Managing Requirements Evolution

using Reconstructed Traceability

and Requirements Views

Managing Requirements Evolution

using Reconstructed Traceability and Requirements Views

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. dr. ir. J.T. Fokkema,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op maandag 12 januari 2009 om 15:00 uur
door

Marco LORMANS

informatica ingenieur
geboren te Heemskerk.

Dit proefschrift is goedgekeurd door de promotor:

Prof. dr. A. van Deursen

Samenstelling promotiecommissie:

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

IPA Dissertation Series 2009-03

The research described in this thesis was mede possible through support of SenterNovem (MOOSE and MERLIN project) and NWO (Reconstructor Project).

About the cover: The cover shows a fragment of the album cover of Joy Division's debut album Unknown Pleasures designed by Joy Division, Peter Saville, and Chris Mathan in 1979.

Printed by Universal Press, Veenendaal

ISBN: 978-90-79982-01-1

Contents

List of Figures

Acknowledgements

Finally, the finish of my biggest project ever is there. Like any project, my PhD research project had its ups and downs. These ups and downs are best described by the emails of the program committee of a conference, which describe the decision to accept or reject a submitted article at the specific conference.

If the article is accepted, it means you have done a good job and your research fills a niche and is appreciated by the international research community. Furthermore, you are one step closer to the finish of your PhD research.

The downside is a rejected article. The quality of your work is not good enough and you have to work harder to find the results that will satisfy the international research community. Writing these acknowledgements means that my research work is finished and the quality was sufficient to qualify for a PhD. I could not have made it to this point without the help of some specific people that I would like to thank here.

First of all, I want to thank Hans Toetenel and Jan Dietz for the job opportunity to participate in an international research project and to write a PhD thesis. I especially want to thank Hans for his supervision during the first phase of the project.

Next, I want to thank the examination committee: Prof. dr. ir. A.J.C. van Gemund, Prof. dr. A. de Lucia, Prof. dr. R.J. Wieringa, Prof. dr. J.C. van Vliet, Prof. dr. ir. A.P. de Vries, and Dr. ir. R. van Solingen for reading and giving feedback on the manuscript. Their comments were very welcome and contributed to the quality of the thesis.

A large part of my research project has been done in close collaboration with industry: the MOOSE and MERLIN projects. These projects gave me the possibility to feel the taste of industry and do industrial case studies. I want thank all project members of both projects for all our inspiring discussions! In particular, I want to thank my project members and co-authors Päivi Parviainen, Maarit Tihinen, Eric Nöcker, Aart de Zeeuw, Andre Stéhouwer, Hans Spanjers, Maarten ter Huurne, and Dan Bendas, for their collaboration in writing our publications. Besides that, special thanks goes to Rob Kommeren (Philips), Lou Somers (Océ Technologies and Technical University Eindhoven), and Durk van der Ploeg (ASML) for providing me an entrance at their companies and providing me the materials for doing the case studies. Finally, one man in particular I want thank and that is Rini van Solingen. He was the project manager of both MOOSE and MERLIN projects, he is co-author of several publications, he was always available to give advice, and last but not least, he was a great company when drinking a beer at our various working trips.

Next, I want to thank my colleagues of the Software Engineering group at Delft University of Technology. In particular, I want to thank Marius Marin, Hylke van Dijk, Hans-Gerhard Gross, and Bas Graaf. Marius for the tough challenges on the tennis court, Hylke for the many brainstorm sessions, and Gerd for the collaboration in doing research and writing publications. Furthermore, Bas with whom I worked together during the whole research project and even my master thesis. He was my roommate, and travel partner for the whole five years of doing research in Delft. Because of him, I have had a lot of fun!

Last but certainly not least, I would like to thank Arie van Deursen. Without his support and patience, I would have never come to the finish line. I am very grateful that he was willing to take up the role as supervisor and promotor for my research project. In particular, I want to thank Arie for his “always being available”. I can remember us talking on the phone late at night or in the weekend when I needed to make a deadline. Arie is a great mentor and it was a great pleasure working with him. Again Arie, thanks for your patience!

Finally, I would like to thank my mom for her support during the years of research and her advice to go and study in Delft in the first place. Her reminders, especially when I left Delft, made me not to forget my to-do-list.

Many thanks to all!

Marco Lormans
Delft, November 2008

Summary

Modern software systems become more complex and the environment where these systems operate in become more and more dynamic. The number of stakeholders increase and the stakeholders' needs change constantly as they need to adjust to the constantly changing environment. A consequence of this trend is that the average number of requirements for a software system increase and change continually.

Requirements engineering is the process in the software development life-cycle that deals with specifying the stakeholders' needs (in requirements) and managing the requirements during the whole software development life-cycle. So, in contrast to the classical way of thinking, that requirements engineering is a up front activity, we see requirements engineering as a life-cycle wide activity.

In modern software engineering the requirements engineering process needs to deal with the first (*Continuing Change*) and second (*Increasing Complexity*) law of software evolution defined by Lehman [Lehman, 1998; Laszlo A. Belady, 1976]. These laws state that 1) software systems must continually be adapted to new stakeholder requirements or changed environments, else they become progressively less satisfactory, and 2) software systems that evolve, become more complex due to these new or changed requirements.

Problem

In this research project, we concentrate our work on managing the requirements evolution process. Our main goal is to answer the question: *How can we develop a requirements management environment to improve the management of requirements evolution?*

We started this research by identifying the problems companies face with respect to requirements engineering in general, and requirements management in specifically. We did that by conducting an inventory at the industrial partners of the MOOSE and MERLIN projects. In particular, we concentrated our research on outsourcing and multi-site projects and focussed on the specific problems these projects face. For example, we studied an industrial case study at Logica, which focussed particularly on developing embedded software systems in an outsourcing context, and a tool developed by Philips, called SOFTFAB, which is designed for supporting distributed software engineering.

The case studies confirmed that monitoring requirements satisfaction is a major problem in practice. Requirements change continuously making the traceability of requirements hard and the monitoring of requirements unreliable. Subsequently, it becomes impossible to manage and monitor the progress of the requirements within the software development life-cycle: are the requirements already covered in the design, and are they covered in the test cases? Ultimately, this can lead to systems that do not comply to the requirements of the stakeholders.

Results

As a solution, we developed a framework, called RES (Requirements Engineering System) that structures the process of requirements evolution. This framework defined the three main processes relevant for managing requirements evolution: 1) the interaction with stakeholders, 2) the consistent processing of changes, and 3) the presentation of information using views.

Next, we developed a methodology, called MAREV (a Methodology for Automating Requirements Evolution using Views) that improves the traceability and monitoring of requirements. This methodology covers all three processes defined in the RES framework and consists of 7 steps.

We have implemented the MAREV methodology in a tool suite, called REQANALYST. It uses Latent Semantic Index (LSI), an Information Retrieval (IR) technique to recover the traceability links of the requirements with other work products produced in the development life-cycle. This traceability information is used to generate "requirements views" that help monitoring and managing the requirements. Finally, our MAREV methodology is applied in several academic and industrial case studies, which resulted in several lessons learned.

Chapter 1

Introduction[1]

This chapter introduces a research project concentrating on improving requirements management. The research project starts with identifying the problems companies face with respect to requirements engineering in general, and requirements management in particular. We learned that monitoring requirements satisfaction is a major problem in practice. Requirements change continuously, making the traceability of requirements hard and the monitoring of requirements unreliable. In this chapter we explain the problem of managing requirements evolution using an example from industry. Then, we define the objective for this research project and finally present a roadmap for this research project and how to tackle this problem.

1.1 Managing Requirements Evolution

Requirements engineering is the process of analyzing and documenting the intended functionality of a software system. Although often thought of as an upfront activity in the construction of a software system, it is in fact a life cycle wide activity. To accommodate this, we need a requirements engineering process that is properly integrated into the wider system development process [Finkelstein, 2004].

The most important aspects for realizing a life cycle wide requirements engineering process are the interaction with involved actors and the evolution of the system. With evolution we primarily mean the changes we need to deal with during the development of a system, e.g., after the first release. The term "changes" is very broadly defined; it includes new environments,

[1]This chapter is based on: Marco Lormans. Monitoring requirements evolution using views. In *Proceedings of the 11th European Conference on Software Maintenance and Reengineering*, Amsterdam, The Netherlands, March 2007

changing expectations, emergent properties, all affecting the requirements of the system.

As an example, in Chapters 3 and 5, we will encounter a traffic monitoring system. Between two releases of this system, more than 500 requirements were significantly modified, and around 80 entirely new requirements were identified. As is often the case, many of these requirements changes originate from increased insight and understanding of the problem domain, rather than from new functionality that is to be added to the system at hand.

During the whole lifetime of the system, all changes (including changes to requirements) need to be processed and synchronized among all stakeholders to keep the development activities consistent. The different perspectives (views) on the system not only need to be combined at the start of development, this harmonization is a continuous activity during the whole life-cycle. Furthermore, the changes and the progress need to be monitored.

This monitoring of requirements concerns the evolution of the requirements and makes sure the whole set of requirements stays consistent, as well as the relations between requirements and other work products such as design decisions, and test cases.

1.2 Problem Statement

Requirements evolve during the development life cycle as a result of volatility and continuous change. If not managed properly, this can cause inconsistencies, which again can lead to time-consuming and costly repairs, such as requirements that are not implemented in the final release of the system. Current requirements management tools support this evolution of requirements insufficiently, as we will see in Chapters 2 and 3. Furthermore, requirements management (including tracing and monitoring requirements) is difficult, time-consuming and error-prone when done manually.

Research Hypothesis An automated requirements management environment, supported by a tool, will improve requirements evolution with respect to keeping requirements traceability consistent, realizing reliable requirements monitoring using views, improving the quality of the documentation, and reducing the manual effort.

1.3 Objectives

In the previous section we have discussed the problem we want to tackle in this thesis and formulated a hypothesis. To address the problem stated in the previous section, we need to answer the following question:

RQ0 *How can we develop a requirements management environment to improve the management of requirements evolution?*

To answer this main research question, we decompose this main problem into four smaller problem areas, each corresponding to a sub-question that we will tackle in this thesis.

1.3.1 Requirements Evolution in Practice

As we already pinpointed in the problem statement, requirements evolution is unavoidable in practice. Managing this requirements evolution is hard. Most companies have no structured approach for dealing with requirements evolution. They simply use an ad-hoc strategy for dealing with this issue. For small projects this does not have to lead to problems, if one person can oversee the whole system. For big projects this requirements evolution will likely lead to unwelcome problems.

In practice, it often happens that functionality is missing or that functionality not documented in the requirements specifications, is implemented in the system. We suspect that one of the causes for this is the evolution of requirements. As the requirements are volatile, the functionality described in the system specification is instable and is exposed to constant change. We would like to know what the impact of requirements evolution is in practice. In this thesis we seek to better understand the approaches currently used in requirements management, and the problems that arise from requirements evolution. This leads to our first sub-question:

RQ1 *How is requirements management currently done in practice and what is the impact of requirements evolution?*

1.3.2 Requirements Monitoring and Views

Monitoring the evolution of requirements is necessary to avoid unexpected issues in the system life-cycle. Monitoring of requirements is typically done by capturing specific information about the requirements in views. The term “view” is often used in the area of software engineering, especially in the area of requirements engineering. Views are generally introduced as a means for separation of concerns [Nuseibeh et al., 1994; Kruchten, 1995] and mostly represent a specific perspective on a system. This perspective is often a subset of the whole system to reduce the complexity.

For example, each stakeholder is interested in a different part of the system. This stakeholder can also be a developer, as he/she is often also only interested in a small part of the complete system. The perspective a view represents can also be an abstraction of the system giving an overview of the whole system without too many details.

A "requirements view" on a system or development process offers a perspective on that system in which requirements assume the leading role [Nuseibeh et al., 1994]. For example, a view can describe project progress in terms of testing; showing the requirements that have been successfully tested, in terms of design; showing the requirements that resulted in a design decision), or in terms of coding; showing the requirements that were actually implemented.

Requirements views are essential for successful project management, in order to monitor progress. However, there is no general agreement on what constitutes a requirements view, and what information it needs to contain. This leads us to our second research question:

RQ2 *Which views do we need to support the process of monitoring requirements evolution?*

1.3.3 Requirements Traceability and Traceability Reconstruction

In order to reconstruct requirements views from project documentation we need traceability support. For example, if we want to show which requirements are covered by a test case, we need to relate each requirement to at least one test case. Traceability support ensures that the requirements will be linked to other work products such as test cases.

In current practice managing traceability support is hard. Within many projects traceability is not implemented. If it is considered, it is often done manually. Today's tools are insufficient to support the traceability activities.

This leads to our third sub-question:

RQ3 *How can we automate the reconstruction of requirements traceability to support requirements evolution?*

1.3.4 Requirements Evolution and Global Software Development

Requirements management is especially important when large software systems with a long life-cycle are concerned. They are often based on previously released products and much software is reused and/or extended.

Besides this, these large software systems are being developed by many different teams on different physical locations. This so called global software development is introduced by many companies and this trend is expected to grow for the coming years.

This trend of global software development and outsourcing causes that requirements management becomes even harder to organise and implement in a project. This leads to our final sub-question:

RQ4 *What is the impact of global software development on the management of requirements evolution?*

1.4 Research Methodology and Evaluation

The research, aimed at addressing the four questions just raised, was initiated as part of an international research project consisting of a consortium containing academic partners and industrial partners. The main reason for this is the fact that software engineering can be considered as an applied science. Most of the previously described problems typically can best be identified in industry.

The research described is done as part of two ITEA projects, MOOSE[1] and MERLIN[2]. MOOSE aims at improving software quality and development productivity for embedded systems, by adapting, tailoring and combining technologies to fit a specific – industrial – situation. MERLIN aims at developing software systems in collaboration.

This research follows a number of steps:

- In order to further explore the problem domain, we carry out a survey among the partners of the MOOSE and MERLIN projects. Furthermore, we conduct a descriptive case study in the area of requirements management in an outsourcing context.
- With this increased insight in the main problems in the area of requirements management as starting point, we define a method for supporting requirements management, for which we develop tool support as well.
- Finally, we evaluate the proposed method and tools by means of a number of case studies.

We conduct the case studies in close collaboration with industry. For every case study, we have an employee of the company actively participating

[1]www.mooseproject.org
[2]www.merlinproject.org

in the research work. This person ensures that our approach will be used by other employers and that we will get feedback about our goals and criteria. For example, in every case study, we ask the users if our approach improves their way of working, the quality of the delivered work, and reduce the effort that is needed to do their job. This approach of working with industry gives us the feedback we need to evaluate our work.

1.5 Outline of Thesis

This thesis investigates the problem of monitoring requirements evolution. To realise our objectives this thesis is organized according to four main themes (related to our research questions). Each theme will be discussed in relation to their corresponding chapter(s).

Exploring the World Chapter 2 starts our investigation with a survey conducted among the industrial partners of the MOOSE project. The main goal of this survey was to explore the domain of embedded software development and identify the main problems concerning requirements engineering and architectural design. The resulting observations served as a roadmap for the rest of our research project.

Chapter 3 zooms in on some specific problems concerning requirements management. In this chapter requirements management systems are introduced and a real-life project is investigated. We focus on the underlying problems causing requirements management to be difficult in practice, and explore what the impact of requirements evolution is.

In these first two chapters we show how requirements management is dealt with in practice and what the implications of requirements evolution are.

Structuring the World Besides exploring the problems of requirements management, Chapter 3 also introduces a framework, called RES (Requirements Engineering Systems) framework, to structure the problems identified in the previous chapters. The RES framework that addresses the concerns with respect to requirements monitoring such as 1) the interaction with stakeholders, 2) consistent processing of changes, and 3) presentation of information using requirements views.

Then, in Chapter 4 a methodology is defined, called MAREV, which helps to structure the requirements management process and supports requirements evolution. Besides that, Chapter 4 presents a tool suite, called REQ-ANALYST, that implements the methodology. All three aspects identified in Chapter 3 are covered by MAREV in seven predefined steps:

1. Defining the traceability meta-model;
2. Identifying the work products;
3. Pre-processing the work products;
4. Reconstructing the traceability links;
5. Selecting the relevant links;
6. Generating requirements views;
7. Addressing requirements evolution.

The first two steps concern the interaction with stakeholders. Steps 3, 4, 5, and 7 address the consistent processing of changes. Finally, step 6 covers the presentation of information.

In Chapter 4 the focus is primarily on the first five steps of the MAREV methodology. The underlying technologies for defining the interaction with stakeholders and the processing of changes are investigated. The first aspect is covered by defining meta-models (step 1 of MAREV) and identifying the concepts in the current way of working (step 2). This approach directly reflects the approach studied in the case study of Chapter 3.

For steps 3 to 5 of MAREV, information retrieval technologies are studied to automate the traceability support process. Currently available approaches are studied and incorporated in MAREV. In particular, we study how Latent Semantic Indexing (LSI) can be used to reconstruct traceability links. In step 3 of MAREV, the predefined concepts are pre-processed for automated traceability reconstruction (step 4). Then, in step 5 the reconstructed candidate links are filtered and only the relevant links are selected as traceability links. Finally, Step 7 deals with the consistent processing of changes to the predefined concepts. To do this, steps 3 to 5 again need to be carried out for the concepts that changed. This is exactly what step 7 needs to support.

In Chapter 4 experiments are carried out with MAREV and REQANALYST. The experiments are done in three case studies; two lab exercises and one industrial case study.

Showing the World In Chapter 5, we focus on step 6 of MAREV. In this step the reconstructed traceability links are used to generate our requirements views. In this chapter a questionnaire is distributed to learn what project members expect from a requirements view. This questionnaire resulted in a number of requirements views that can be useful in practice. These views are incorporated in REQANALYST and used in a case study. The case

Table 1.1: Coverage of research questions in chapters

	2	3	4	5	6
RQ1	X	X			
RQ2		X		X	
RQ3			X	X	
RQ4		X			X

study resulted in a number of observations concerning MAREV and REQ-ANALYST as discussed in Chapter 4 and the expectations of a requirements management system discussed in Chapter 3. Each observation is discussed separately in order to give a clear insight in the applicability of our solution.

Expanding the World In Chapter 6 we discuss a tool suite, called SOFTFAB, developed by Philips Applied Technologies, which has been successfully applied in several multi-site projects. In this chapter, we elaborate on the underlying concepts and define a list of features needed by systems that intend to support distributed software development. We evaluate our list of features and relate them to our requirements for monitoring requirements evolution.

Recommendations to the World Finally, to conclude our research project, Chapter 7 looks back to our initial objectives and explains how these are covered by the thesis. The correspondence between chapters and research questions is summarized in Table 1.1.

1.6 Origin of Chapters

All chapters in this thesis are based on work that is published in various international journals, conferences proceedings, book chapters and workshops. For each chapter, one or more publications have appeared. For each chapter, an overview of the publication(s) is given including an explanation of my contribution per publication and the contribution of each co-author (except for supervisors Arie van Deursen and Hans-Gerhard Gross).

1.6.1 Chapter 1

This first chapter is based on the following publication:

- Marco Lormans. Monitoring requirements evolution using views. In *Proceedings of the 11th European Conference on Software Maintenance and Reengineering*, Amsterdam, The Netherlands, March 2007

This publication is written to give an overview of the whole research project; it describes the goal and context, defines the research questions, and the explains the approach taken in this research project. Furthermore, it aligns all deliverables produced in this project.

1.6.2 Chapter 2

Chapter 2 is based on three publications:

- Bas Graaf, Marco Lormans, and Hans Toetenel. Software technologies for embedded systems: An industry inventory. In *Product Focused Software Process Improvement, 4th International Conference, PROFES2002*, volume 2559 of *Lecture Notes in Computer Science*, pages 453–465. Springer-Verlag, 2002

- Bas Graaf, Marco Lormans, and Hans Toetenel. Embedded software engineering: state of the practice. *IEEE Software*, 20(6):61–69, November – December 2003

- Päivi Parviainen, Maarit Tihinen, Marco Lormans, and Rini van Solingen. Requirements engineering: Dealing with the complexity of sociotechnical systems development. In José Luis Maté and Andrés Silva, editors, *Requirements Engineering for Sociotechnical Systems*. IdeaGroup Inc., 2004. Chapter 1

[Graaf et al., 2002] and [Graaf et al., 2003] give an overview of the state of the practice and [Parviainen et al., 2004] gives an overview of the state of the art. Chapter 2 is primarily based on the work done in the first two publications. These publications are the result of the work by Bas Graaf and myself. Together, we visited all companies and wrote the reports, which finally led to these publications, to which we contributed equally. We chose to use alphabetic ordering of the authors, making Bas Graaf first author.

1.6.3 Chapter 3

For Chapter 3, we used the following publication:

- Marco Lormans, Hylke van Dijk, Arie van Deursen, Eric Nöcker, and Aart de Zeeuw. Managing evolving requirements in an outsoucring context: An industrial experience report. In *Proc. of the Int. Workshop on Principles of Software Evolution (IWPSE'04)*, Kyoto, Japan, 2004. IEEE Computer Society

This publication describes a investigation that I have done within Logica [1]. We investigated all issues related requirements within a project at Logica. It resulted in a conceptual framework for requirements evolution called RES.

1.6.4 Chapter 4

Chapter 4 is based on three publications:

- Marco Lormans and Arie van Deursen. Reconstructing requirements coverage views from design and test using traceability recovery via LSI. In *Proc. of the Int. Workshop on Traceability in Emerging Forms of Software Engineering*, pages 37–42, Long Beach, CA, USA, November 2005
- Marco Lormans and Arie van Deursen. Can LSI help reconstructing requirements traceability in design and test? In *Proc. of the 10th European Conf. on Software Maintenance and Reengineering*, pages 47–56, Bari, Italy, March 2006. IEEE Computer Society
- Marco Lormans and Arie van Deursen. Reconstructing requirements traceability in design and test using latent semantic indexing. *Journal of Software Maintenance and Evolution: Research and Practice*, 2008. Submitted for publication

These three publications are the result of our investigation to learn if and how LSI can help solving our issues regarding automating the process of reconstructing requirements traceability. In these publications we defined the our method MAREV and developed the first release of REQ-ANALYST.

1.6.5 Chapter 5

Chapter 5 is based on the following two publications:

- Marco Lormans, Hans-Gerhard Gross, Arie van Deursen, Rini van Solingen, and Andre Stéhouwer. Monitoring requirements coverage using reconstructed views: An industrial case study. In *Proc. of the 13th Working Conf. on Reverse Engineering*, pages 275–284, Benevento, Italy, October 2006
- Marco Lormans, Arie van Deursen, and Hans-Gerhard Gross. An industrial case study in reconstructing requirements views. *Journal of Empirical Software Engineering*, 2008

[1] At time of executing this project, Logica was still called LogicaCMG

In these two publications we applied our MAREV approach and REQ-ANALYST tool suite in an industrial case study. We tried to tune and improve our approach to make it useful in an industrial environment.

1.6.6 Chapter 6

Chapter 6 the following publication:

- Hans Spanjers, Maarten ter Huurne, Bas Graaf, Marco Lormans, Dan Bendas, and Rini van Solingen. Tool support for distributed software engineering. In *Proceedings of the IEEE International Conference on Global Software Engineering (ICGSE'06)*, pages 187–198, Florianopolis, Brazil, 2006. IEEE Computer Society

This publication describes an exercise we did to investigate the impact of global software development in relation to requirements evolution. We used SOFTFAB as a tool to create an environment for doing our investigation. SOFTFAB is a product developed by Hans Spanjers and Maarten ter Huurne. Bas Graaf and myself are the persons who created the environment to do the experiments with SOFTFAB and SKYFAB. Furthermore, we documented the ideas behind SKYFAB and derived the features needed for successfully implementing a MP-DSE support system environment. Again, Bas Graaf and I contributed equally to the paper, and used the alphabetic ordering of authors. Dan Bendas supported us with the technical issues. Finally, many brainstorm sessions were held with Rini van Solingen about our ideas and he also contributed as reviewer during the production of this publication.

Chapter 2

Embedded Software Engineering: The State of the Practice[1]

The embedded software market has grown very fast the last decade and will continue to do so in the coming years. The specific properties of embedded software, such as hardware dependencies, make its development different from non-embedded software. Therefore we expected very specific software development technologies to be used in this domain. The inventory we conducted at several embedded-software-development companies in Europe remarkably shows that this is not true. However the inventory results concerning requirements engineering and architecture design at these companies do suggest that there is a need for more specifically aimed development technologies. This chapter presents the inventory results and identifies possibilities for future research to customize existing and develop new software development technologies for the embedded-software domain.

2.1 Introduction

Many products today contain software (e.g., mobile telephones, DVD players, cars, airplanes, and medical systems). Because of advancements in information and communication technology, in the future even more products will likely contain software. The market for these 'enhanced' products is forecasted to grow exponentially in the next 10 years [Eggermont, ed.k, 2002]. Moreover, the complexity of these embedded systems is increasing, and the amount and variety of software in these products are growing. This creates a big challenge for embedded-software development. In the years

[1]This chapter was originally published as: Bas Graaf, Marco Lormans, and Hans Toetenel. Embedded software engineering: state of the practice. *IEEE Software*, 20(6):61–69, November – December 2003

to come, the key to success will be the ability to successfully develop high-quality embedded systems and software on time. As the complexity, number, and diversity of applications increase, more and more companies are having trouble achieving sufficient product quality and timely delivery. To optimize the timeliness, productivity, and quality of embedded software development, companies must apply software engineering technologies that are appropriate for specific situations.

Unfortunately, the many available software development technologies do not take into account the specific needs of embedded systems development. This development of embedded systems is fundamentally different from that of non-embedded systems. Technologies for the development of embedded systems should address specific constraints such as hard timing constraints, limited memory and power use, predefined hardware platform technology, and hardware costs. Existing development technologies don't address their specific impact on, or necessary customization for, the embedded domain. Nor do these technologies give developers any indication of how to apply them to specific areas in this domain – for example, automotive systems, telecommunications, or consumer electronics. Consequently, tailoring a technology for a specific use is difficult. Furthermore, the embedded domain is driven by reliability factors, cost factors, and time to market. So, this embedded domain needs specifically targeted development technologies.

In industry, the general feeling is that the current practice of embedded software development is unsatisfactory. However, changes to the development process must be gradual; a direction must be supplied. To achieve this, we need more insight into the currently available and currently used methods, tools, and techniques in industry.

To gain such insight, we performed an industrial inventory as part of the MOOSE [1] project. MOOSE is an ITEA [2] project aimed at improving software quality and development productivity for embedded systems. Not only did we gain an overview of which technologies the MOOSE consortium's industrial partners use, we also learned why they use or don't use certain technologies. In addition, we gained insight into what currently unavailable technologies might be helpful in the future.

2.2 Methods and Scope

The inventory involved seven industrial companies and one research institute in three European countries (see Table 2.1). These companies build a

[1] MOOSE- software engineering MethOdOlogieS for Embedded systems, www.mooseproject.org

[2] ITEA- Information Technology for European Advancement, www.itea-office.org

Table 2.1: Inventoried companies

Company	**Products**
TeamArteche (Spain)	Measurement, control, and protection systems for electrical substations
Nokia (Finland)	Mobile networks and mobile phones
Solid (Finland)	Distributed-data-management solutions
VTT Electronics (Finland)	Technology services for businesses
Philips PDSL (Netherlands)	Consumer electronics
ASML (Netherlands)	Lithography systems for the semiconductor industry
Océ (Netherlands)	Document-processing systems
Logica (Netherlands)	Global IT solutions and services

variety of embedded software products, ranging from consumer electronics to highly specialized industrial machines. We performed 36 one-hour interviews with software practitioners. The respondents were engineers, researchers, software or system architects, and managers, with varying backgrounds. To get a fair overview of the companies involved (most of which are very large), we interviewed at least three respondents at the smaller companies and five or six at the larger companies. These interviews were conducted in the period April–October 2002.

We based the interviews on an outline specifying the discussion topics (see Table 2.2). To be as complete as possible, we based this outline on a reference process model. Because software process improvement methods have such a (ideal) process model as their core, we used one of them. We chose the BOOTSTRAP methodology's (www.bootstrap-institute.com) process model because of its relative emphasis on engineering processes compared to other process models, such as those of the Capability Maturity Model and SPICE (Software Process Improvement and Capability Determination) [Wang et al., 1999]. BOOTSTRAP's other advantage for this inventory is that the BOOTSTRAP Institute developed it with the European software industry in mind.

For every interview we created a report, which the respondent needed to approve. We consolidated the reports for a company into one report. We then analyzed the company reports for trends and common practices. Finally, we wrote a comprehensive report that, for confidentiality reasons, is available only to MOOSE consortium members. That report forms the

Table 2.2: A sample of the interview outline

Here are some discussion topics and questions from the outline we used for the interviews.

Technology

What are the most important reasons for selecting development technologies?

- Impact of introducing new technologies (cost, time, and so on).
- Why not use modern/different technologies?

Software life cycle

Software requirements engineering
How are the requirements being gathered?

- What are the different activities?
- What documents are produced?
- What about tool support?

How are the requirements being specified?

- What specification language?
- What about tool support? (Consider cost, complexity, automation, training, acceptance)
- What notations/diagrams?
- What documents are produced?
- How are documents reviewed?
- What are advantages/disadvantages of followed approaches?

Software architecture design
How is the architecture specified?

- What architecture description language?
- What about tool support? (Consider cost, complexity, automation, training, acceptance)
- Are design patterns used?
- What notations/diagrams?
- What documents are produced?
- How are documents reviewed?
- What are advantages/disadvantages of followed approaches?

basis for this discussion.

2.3 Embedded Software Development Context

When considering the embedded software development process, you need to understand the context in which it is applied. After all, most companies that develop embedded software do not sell it. They sell mobile phones, CD players, lithography systems, and other products. The software in these products constitutes only one (important) part. Embedded software engineering and other processes such as mechanical engineering and electrical engineering are in fact sub-processes of systems engineering. Coordinating these sub-processes to develop quality products is one of embedded system development's most challenging aspects. The increasing complexity of systems makes it impossible to consider these disciplines in isolation.

For instance, when looking at communication between different development teams, we noticed that besides vertical communication links along the lines of the hierarchy of architectures, horizontal communication links existed. Vertical communication occurs between developers who are responsible for systems, subsystems, or components at different abstraction levels (for example, a system architect communicating with a software architect). Horizontal communication occurs between developers who are responsible for these things at the same abstraction level (for example, a programmer responsible for component A communicating with a programmer responsible for component B).

Still, we found that systems engineering was mostly hardware driven – that is, from a mechanical or an electronic viewpoint. In some companies, software architects weren't even involved in design decisions at the system level. Hardware development primarily dominated system development because of longer lead times and logistical dependencies on external suppliers. Consequently, software development started when hardware development was already at a stage where changes would be expensive. Hardware properties then narrowed the solution space for software development. This resulted in situations where software inappropriately fulfilled the role of integrator; that is, problems that should have been solved in the hardware domain were solved in the software domain. Embedded software developers felt that this was becoming a serious problem. So, in many companies this was changing; software architects were becoming more involved on the system level.

Depending on the complexity of the product, projects used system requirements to design a system architecture containing multidisciplinary or monodisciplinary subsystems. A multidisciplinary subsystem will be implemented by different disciplines; a discipline refers to software, or mechanics, or electronics, or optics, and so on. A monodisciplinary subsystem will be implemented by one discipline. Next, the projects allocated system requirements to the architecture's different elements and refined the requirements. This process repeated for each subsystem. Finally, the projects decomposed the subsystems into monodisciplinary components that an individual developer or small groups of developers could develop. The level of detail at which decomposition resulted in monodisciplinary subsystems varied. In some cases, the first design or decomposition step immediately resulted in monodisciplinary subsystems and the corresponding requirements. In other cases, subsystems remained multidisciplinary for several design steps.

This generic embedded systems development process resulted in a tree of requirements and design documents (see Figure 2.1). Each level represented the system at a specific abstraction level. The more complex the system, the more evident this concept of abstraction levels was in the development process and its resulting artefacts (for example, requirements documentation).

In the process in Figure 2.1, requirements on different abstraction levels are related to each other by design decisions, which were recorded in architecture and design documentation. At the system level, these decisions concerned partitioning of the functional and non-functional requirements over software and hardware components. The criteria used for such concurrent design (co-design) were mostly implicit and based on system architects' experience.

2.4 Requirements Engineering Results

Typically, the development of embedded systems involved many stakeholders. This was most apparent during requirements engineering. Figure 2.2 depicts our view of the most common stakeholders and other factors.

In requirements engineering's first phase, the customer determines the functional and non-functional requirements. Depending on the product domain, the customer negotiates the requirements via the marketing and sales area or directly with the developers.

The first phase's output is the agreed requirements specification, which is a description of the system that all stakeholders can understand. This document serves as a contract between the stakeholders and developers. At this point, we noticed a clear difference between small and large projects. In small projects, the stakeholder requirements also served as developer requirements. In large projects, stakeholder requirements were translated into technically oriented developer requirements.

Requirements specify what a system does; a design describes how to

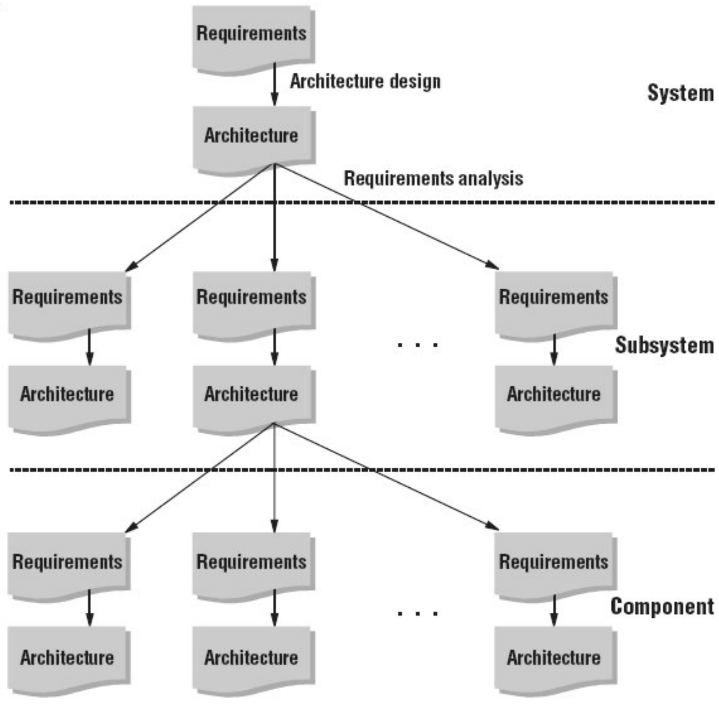


Figure 2.1: The decomposition of the embedded systems development process

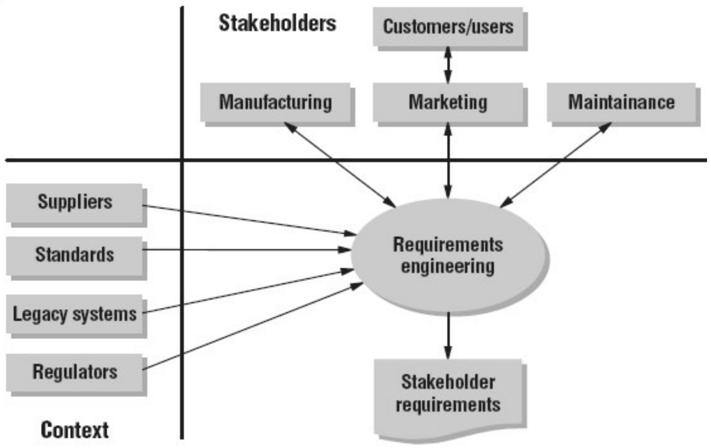


Figure 2.2: Embedded systems-development stakeholders and other factors

realize a system. Software engineering textbooks strictly separate the requirements and the design phases of software development; in practice, this separation is less obvious. In fact, the small companies often put both the requirements and design into the system specification. These companies did not translate the system requirements explicitly into software requirements. The development processes in the larger companies did result in separate requirements and design documents on different abstraction levels. However, in many cases, these companies directly copied information from a design document into a requirements document for the next abstraction level instead of first performing additional requirements analysis.

2.4.1 Requirements Specification

Companies usually specified requirements in natural language and processed them with an ordinary word processor. They normally used templates and guidelines to structure the documents. The templates prescribed what aspects had to be specified. However, not all projects at a company used these templates, so requirements specifications from different projects sometimes looked quite different.

Because embedded software's non-functional properties are typically important, we expected these templates to reserve a section on non-functional requirements next to functional requirements. This wasn't always the case. For example, the requirements specification didn't always explicitly take into account real-time requirements. Sometimes a project expressed them in a separate section in the requirements documents, but often they were implicit. Requirements specification and design also usually didn't explicitly address other typical embedded software requirements, such as those for power consumption and memory use.

Projects that employed diagrams to support requirements used mostly free-form and box-line diagrams in a style that resembles the Unified Modeling Language, dataflow diagrams, or other notations. Project members primarily used general-purpose drawing tools to draw the diagrams. Because of the lack of proper syntax and semantics, other project members often misinterpreted the diagrams. This was especially true for project members working in other disciplines that employ a different type of notation.

UML was not common practice yet, but most companies were at least considering its possibilities for application in requirements engineering. Use cases were the most-used UML constructs in this phase. Some projects used sequence diagrams to realize use cases; others applied class diagrams for domain modelling. However, the interpretation of UML notations was not always agreed on during requirements engineering. It wasn't always clear, for instance, what objects and messages in UML diagrams denote when a sequence diagram specifies a use case realization.

On the lowest levels, projects commonly used pre- and postconditions to specify software requirements. They specified interfaces as pre- and postconditions in natural language, C, or some interface definition language.

Projects rarely used formal specifications. One reason was that formal specifications were considered difficult to use in complex industrial environments and require specialized skills. When projects did use them, communication between project members was difficult because most members did not completely understand them. This problem worsened as projects and customers needed to communicate. In one case, however, a project whose highest priority was safety used the formal notation Z for specification.

2.4.2 Requirements Management

When looking at Figures 2.1 and 2.2, you can imagine that it's hard to manage the different requirements from all these different sources throughout development. This issue was important especially in large projects.

Another complicating factor was that most projects didn't start from scratch. In most cases, companies built a new project on previous projects. So, these new projects reused requirements specifications (even for developing a new product line). Consequently, keeping requirements documents consistent was difficult. To keep all development products and documents consistent, the projects had to analyze the new features' impact precisely.

However, the projects frequently didn't explicitly document relations between requirements, so impact analysis was quite difficult. This traceability is an essential aspect of requirements management. Tracing requirements was difficult because the relations (for example, between requirements and architectural components) were too complex to specify manually.

Available requirements management tools didn't seem to solve this problem, although tailored versions worked in some cases. A general shortcoming of these tools was that the relations between the requirements had no meaning. In particular, tool users could specify the relations but not the rationale behind the link.

When projects did document relations between requirements, they used separate spreadsheets. Some companies were using or experimenting with more advanced requirements management tools such as RequisitePro (IBM Rational), RTM (Integrated Chipware), and DOORS (Telelogic). These experiments weren't always successful. In one case, the tool's users didn't have the right skills, and learning them took too long. Also, the tool handled only the more trivial relations between requirements, design, and test documents. So, developers couldn't rely on the tool completely, which is important when using a tool.

Requirements management also involves release management (managing features in releases), change management (backwards compatibility), and configuration management. Some requirements management tools supported these processes. However, because most companies already had other tools for this functionality, integration with those tools would have been preferable.

2.5 Software Architecture Results

Small projects didn't always consider the explicit development, specification, and analysis of the product architecture necessary. Also, owing to time-to-market pressure, the scheduled deadlines often obstructed the development of sound architectures. Architects often said they didn't have enough time to do things right.

The distinction between detailed design and architecture seemed somewhat arbitrary. During development, the projects interpreted architecture simply as high-level design. They didn't make the distinction between architectural and other types of design explicit, as, for example, Eden and Kazman [2003]. There, the locality criterion is introduced to distinguish architectural design from detailed design. A design statement is said to be local when it can't be violated by mere expansion. The application of a design pattern is an example of a local design statement. Architectural design is not local. For instance, an architectural style can be violated by simple expansion.

2.5.1 Architecture Design

Designing a product's or subsystem's architecture was foremost a creative activity that was difficult to divide into small, easy-to-take steps. Just as system requirements formed the basis for system architecture decisions, system architecture decisions constrained the software architecture.

In some cases, a different organizational unit had designed the system architecture. So, the architecture was more or less fixed – for instance, when the hardware architecture was designed first or was already known. This led to suboptimal (software) architectures. Because software was considered more flexible and has a shorter lead time, projects used it to fix hardware architecture flaws, as we mentioned before.

The design process didn't always explicitly take into account performance requirements. In most cases where performance was an issue, the projects just designed the system to be as fast as possible. They didn't establish how fast until an implementation was available. Projects that took performance requirements into account during design did so mostly through budgeting. For example, they frequently divided a high-level real-time constraint among several lower-level components. This division, however, often was based on the developers' experience rather than well-funded calculations. Projects also used this technique for other non-functional requirements such as for power and memory use.

Projects sometimes considered COTS (commercial off-the-shelf) components as black boxes in a design, specifying only the external interfaces. This was similar to an approach that incorporated hardware drivers into an object-oriented design. However, developers of hardware drivers typically don't use OO techniques. By considering these drivers as black boxes and looking only at their interfaces, the designers could nevertheless include them in an OO design. For the COTS components, the black box approach wasn't always successful. In some cases, the projects also had to consider the components' bugs, so they couldn't treat the components as black boxes.

The software architecture often mirrored the hardware architecture, which made the impact of changes in hardware easier to determine. Most cases involving complex systems employed a layered architecture pattern. These layers made it easier to deal with embedded systems' growing complexity.

2.5.2 Architecture Specification

UML was the most commonly used notation for architectural modelling. On the higher abstraction levels, the specific meaning of UML notations in the architecture documentation should be clear to all stakeholders, which was not always the case. Some projects documented this in a reference architecture or architecture manual (we discuss these documents in more detail later).

The Rational Rose RealTime modelling tool[1] lets developers create executable models and completely generate source code. A few projects tried this approach. One project completely generated reusable embedded software components from Rational Rose RealTime models. However, most of these projects used Rational Rose RealTime experimentally.

For creating UML diagrams, respondents frequently mentioned only two tools: Microsoft Visio and Rational Rose. Projects used these tools mostly for drawing rather than modelling. This means, for instance, that models weren't always syntactically correct and consistent.

Other well-known notations that projects used for architectural modelling were dataflow diagrams, entity-relationship diagrams, flowcharts, and Hatley-Pirbhai diagrams [Hatley and Pirbhai, 1987] to represent control flow and state-based behaviour. Projects often used diagrams based on these notations to clarify textual architectural descriptions in architecture documents. Some projects used more free-form box-line drawings to document and communicate designs and architectures.

One project used the Koala component model [van Ommering et al., 2000] to describe the software architecture. Compared to box-line drawings, the Koala component model's graphical notation has a more defined syntax. Koala provides interface and component definition languages based on C syntax. A Koala architecture diagram specifies the interfaces that a component provides and requires. This project used Microsoft Visio to draw the Koala diagrams.

Projects often used pseudocode and pre- and postconditions to specify interfaces. Although this technique is more structured than natural language, the resulting specifications were mostly incomplete, with many implicit assumptions. This not only sometimes led to misunderstandings but also hampered the use of other techniques such as formal verification.

Some projects referred to a reference architecture or an architecture user manual. These documents defined the specific notations in architectural documents and explained which architectural concepts to use and how to specify them.

2.5.3 Architecture Analysis

Most projects did not explicitly address architecture verification during design; those that did primarily used qualitative techniques. Few projects

[1]www.rational.com/products/rosert

used quantitative techniques such as Petri nets or rate monotonic scheduling analysis [Liu and Layland, 1973]. One reason is that quantitative-analysis tools need detailed information. In practice, projects often used an architecture only as a vehicle for communication among stakeholders.

The most commonly employed qualitative techniques were reviews, meetings, and checklists. Another qualitative technique employed was scenario-based analysis. With this technique, a project can consider whether the proposed architecture supports different scenarios. By using different types of scenarios (for example, use scenarios and change scenarios), a project not only can validate that the architecture supports a certain functionality but also can verify qualities such as changeability.

The respondents typically felt that formal verification techniques were inapplicable in an industrial setting. They considered these techniques to be useful only in limited application areas such as communication protocols or parts of security-critical systems. The few projects that used Rational Rose RealTime were able to use simulation to verify and validate architectures.

2.5.4 Reuse

Reuse is often considered one of the most important advantages of development using architectural principles. By defining clean, clear interfaces and adopting a component- based development style, projects should be able to assemble new applications from reusable components.

In general, reuse was rather ad hoc. Projects reused requirements, design documents, and code from earlier, similar projects by copying them. This was because most products were based on previous products.

For highly specialized products, respondents felt that using configurable components from a component repository was impossible. Another issue that sometimes prevented reuse was the difficulty of estimating both a reuse approach's benefits and the effort to introduce it.

In some cases a project or company explicitly organized reuse. One company did this in combination with the Koala component model. The company applied this model together with a proprietary method for developing product families.

Some companies had adopted a product line approach to create a product line or family architecture. When adopting this approach, the companies often had to extract the product line architecture from existing product architectures and implementations. This is called *reverse architecting*.

In most cases, hardware platforms served as the basis for defining product lines, but sometimes market segments determined product lines. When a company truly followed a product line approach, architecture design took variability into account.

One company used a propriety software development method that enabled large-scale, multisite, and incremental software development. This method defined separate long-term architecture projects and subsystem projects. The company used the subsystems in short-term projects to instantiate products.

Another company had a special project that made reusable components for a certain subsystem of the product architecture. The company used Rational Rose RealTime to develop these components as executable models.

Some companies practiced reuse by developing general platforms on top of which they developed different products. This strategy is closely related to product lines, which are often defined per platform.

2.6 Discussion

You might well ask, are these results representative of the whole embedded software domain? By interviewing several respondents with different roles in each company, we tried to get a representative understanding of that company's embedded software development processes. The amount of new information gathered during successive interviews decreased. So, we concluded we did have a representative understanding for that company.

With respect to embedded software development in general, we believe that the large number of respondents and the companies' diversity of size, products, and country of origin make this inventory's results representative, for Europe at least. However, whether we can extend these results to other areas (for example, the US) is not proven.

Another point for discussion is that the methods, tools, and techniques the companies used were rather general software engineering technologies. We expected that the companies would use more specialized tools in this domain. Memory, power, and real-time requirements were far less prominent during software development than we expected. That's because most general software engineering technologies didn't have special features for dealing with these requirements. Tailoring can be a solution to this problem, but it involves much effort, and the result is often too specific to apply to other processes. Making software development technologies more flexible can help make tailoring more attractive. So, flexible software development technologies are necessary.

We noticed a relatively large gap between the inventory's results and the available software development technologies. Why isn't industry using many of these technologies? During the interviews, respondents mentioned several reasons. We look at three of them here.

The first reason is *compliance* with legacy. As we mentioned before, most projects didn't start from scratch. Developers always have to deal with this legacy, which means that the technology used in current projects should at least be compatible with the technology used in previous products. Also, companies can often use previous products' components in new products with few or no adaptations. This contradicts the top-down approach in Figure 2.1. Unlike with that approach, components at a detailed level are available from the start, before the new product's architecture is even defined. This would suggest a bottom-up approach. However, because most available software development approaches are top-down, they don't address this issue.

Another reason is *maturity*. Most development methods are defined at a conceptual level; how to deploy and use them is unclear. When methods are past this conceptual stage and even have tool implementations, their maturity can prevent industry from using them. This was the case for some requirements management tools. Some respondents said that these tools weren't suited for managing the complex dependencies between requirements and other development artefacts, such as design and test documentation. Also, integrating these tools with existing solutions for other problems such as configuration management and change management was not straightforward.

The third reason is *complexity*. Complex development technologies require highly skilled software engineers to apply them. But the development process also involves stakeholders who aren't software practitioners. For instance, as we mentioned before, project team members might use architecture specifications to communicate with (external) stakeholders. These stakeholders often do not understand complex technology such as formal architecture description languages. Still, formal specifications are sometimes necessary – for example, in safety-critical systems. To make such highly complex technologies more applicable in industry, these technologies should integrate with more accepted and easy-to-understand technologies. Such a strategy will hide complexity.

2.7 Epilogue

In the previous discussion we explained three reasons for industry not to adopt state-of-the-art software development technologies (compliance with legacy, maturity, and complexity). In the remainder of this thesis we take into account these reasons. Therefore, we try to make use of existing standards and technologies, which already have been successfully applied in industrial practice. When these standards and technologies are not directly applicable, we try to tailor them, or develop new technology.

The observations that software development in practice seldom starts from scratch, that it is moving more and more towards larger scale, and that structured reuse becomes an essential activity, makes our focus on managing the evolution of systems (and in particular requirements) a legitimate choice, where many improvements can be realised in practice.

Finally, we have discussed only some of the opportunities for improving embedded software engineering technologies for requirements engineering and architecture design. Our inventory also considered software engineering processes that we have not presented in this chapter. As an example, we also use the results from the inventory to identify cases where new technologies can be applied and tailored towards industrial needs, such as in Chapter 6 dealing with globally distributed software engineering.

Chapter 3

Managing Evolving Requirements in an Outsourcing Context: An Industrial Experience Report[1]

In this chapter we discuss several difficulties managing evolving requirements by means of an industrial case study conducted at Logica. We report on setting up a requirements management system in an outsourcing context and its application in real-life. The experience results in several lessons learned, questions to be answered in the future on how to manage evolving requirements, and solution directions. We propose a conceptual framework of a requirements engineering system tailored for outsourcing environments, which captures the experience results.

3.1 Introduction

In the IT outsourcing services business it is quite common that the requirements of a new product are provided by an external stakeholder, the client. The client wants a new product and needs you to develop it. In order to eliminate risks of budget overruns, your client may insist on a fixed price agreement. The requirements document often forms the contract. This document typically is the outcome of an elicitation process conducted by the client.

A key problem in outsourcing development, however, is the evolution of requirements: no matter how thorough the requirements specification has

[1]This chapter was originally published as: Marco Lormans, Hylke van Dijk, Arie van Deursen, Eric Nöcker, and Aart de Zeeuw. Managing evolving requirements in an outsoucring context: An industrial experience report. In *Proc. of the Int. Workshop on Principles of Software Evolution (IWPSE'04)*, Kyoto, Japan, 2004. IEEE Computer Society

been set up, the requirements for any non-trivial system will change, not only after the system has been built, but also during the process of implementing the system. This evolution of requirements can be due to many reasons, including changing business needs or market and technology developments [Lehman, 1998]. In addition to that, the process of designing, implementing, and writing test cases for requirements will increase insight in the problem domain, which may well lead to modifications on the initial set of requirements.

A major risk of evolving requirements is that the requirements document itself becomes inconsistent. This may lead to a system that cannot be implemented, misinterpretations or false assumptions by developers, and delivery of a system that will not be accepted by the client.

Requirements management is the requirements engineering activity that aims at controlling changes made to requirements. The purpose of this chapter is to analyse the impact of outsourcing on requirements management. To that end, we study what requirements management techniques an IT solution provider can adopt when accepting an outsourcing contract. In particular, we evaluate the various methods and techniques used by Logica— a major international player in IT services and wireless telecoms — in order to manage evolving requirements of a traffic monitoring system they are implementing for an external customer.

This chapter is primarily an experience report. We believe that collecting and organising the experiences obtained by Logica in a case like this is important for a number of reasons. First, our discussion of requirements management problems as occurring in a state of the art industrial software engineering project may help practitioners in obtaining a better understanding of the problems in their own projects. Second, we discuss which requirements management methods and techniques were actually used, and analyse why these worked well or why they were not satisfactory. This provides an evaluation of existing techniques, which will be valuable to researchers as well as practitioners. Last but not least, we establish a connection between the problems we encountered and the published literature in the software evolution and requirements management literature. From this, we derive a number of research questions in the area of requirements evolution.

The case study discussed in this chapter was again carried out as part of the MOOSE project [MOOSE, 2004]. MOOSE is an ITEA project that aims at improving software quality and development productivity for embedded systems, by adapting, tailoring, and combining technologies to fit a specific – industrial – situation.

The remainder of this chapter is organised as follows. In Section 3.2 we provide background information and discuss the key concepts in requirements management. Then, in Section 3.3, we provide an overview of the context of the case study, discussing the application domain, as well as standards and tooling. In Section 3.4 we present issues pertaining to IT outsourcing including the requirements for a requirements management system specifically for outsourcing. In Section **??** we zoom in on the Logica case, and explain how requirements evolution was tackled in this project. We reflect on this case study in Section 3.6, where we provide a discussion of the observations and lessons learned. The discussion ends with a proposal for a conceptual framework of a requirements engineering system in the context of outsourcing, which captures our observations. We conclude this chapter with a summary of the key contributions and directions for future research.

3.2 Requirements Management

Requirements engineering is often split into two main activities: requirements specification and requirements management [Parviainen et al., 2004]. In our definition, requirements specification concerns activities related to elicitation, analysis, documentation, and validation of requirements. It primarily deals with the content of the requirements.

Requirements Management concerns activities related to controlling and tracking of changes to agreed requirements, relationships between requirements, and dependencies between the requirements specifications (documents) and other specifications produced during the systems and software engineering process [Kontonya and Sommerville, 1998]. We purposefully do not adopt the definition of Leffingwell and Widrig [Leffingwell and Widrig, 2000], who include elicitation, organisation, and documentation of requirements under management activities. In our setting, requirements management is primarily a supportive process, which helps to manage evolving requirements throughout the system's lifecycle.

3.2.1 Requirements Management Systems

Because of the growing number and volatility of requirements, requirements management systems (RMS) have been developed. Wiegers discussed various reasons for using a RMS [Wiegers, 1999]. In the context of outsourcing, status tracking as well as effective communication and interaction with stakeholders are important.

Al-Rawas and Easterbrook emphasize the importance of communication in the requirements management process [Al-Rawas and Easterbrook, 1996]. According to them consistency checking of newly introduced requirements often implies re-establishing communication with the stakeholder

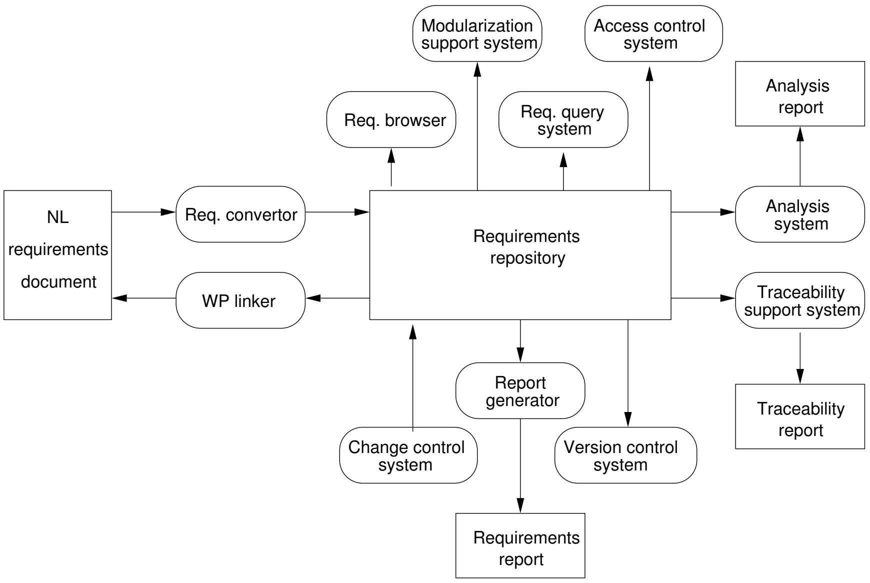


Figure 3.1: A Requirements Management System

of the existing requirements. Traceability is of utmost importance for re-establishing this communication in evolving requirements. The inability to trace these stakeholders and their related information is the crux of the requirements traceability problem [Gotel and Finkelstein, 1994]. In practice keeping the traceability links consistent is a serious challenge [Graaf et al., 2003].

3.2.2 Features of a RMS

A typical RMS stores its requirements repository and provides a number of features to support requirement management activities related to maintenance, evolution, traceability, and change management. Figure 3.1 depicts the typical features of a RMS, based on a diagram of a RMS discussed by Kontonya and Summerville [Kontonya and Sommerville, 1998], extented with features from a discussion by Wiegers [Wiegers, 1999]. This results in the following list of features for any RMS:

- a browser; to support navigation in the set of requirements, e.g., view requirements subsets,

- a query system; to support retrieval of specific requirements from the set of requirements or related requirements,
- a traceability support system; to support management of links to other system elements and the generation of traceability information,
- a report generator; to support generation of all kinds of different reports related to requirements,
- an interface to external documentation; a typical implementation includes a requirements converter and a word processor (WP) linker, to support conversion of the natural language representation (NL) of the requirements to a database format and back again,
- a change control system; to support management of change requests and links to affected requirements,
- a version control system; to support management of different versions of single requirements,
- an analysis system; to perform all kind of analysis on the set of requirements, e.g., impact analysis, status tracking, and determining whether a requirement is an orphan or not,
- an access control system; to control the access rights of users. Not all users are allowed to browse or edit the complete set of requirements,
- a modularisation support system; to support grouping of requirements, e.g., related to a specific quality attribute or piece of functionality.

The list is not exhaustive. The environment and situation determine the features that should be implemented and at what level of detail. For example, in many situations it is not required to implement an access control system.

3.3 Case Study Context

This section introduces the application domain of our case study: a traffic monitoring system (TMS). Besides the TMS we will also introduce MIL-std 498 and the applied tooling at the outsource vendor's site.

3.3.1 Traffic Monitoring System

Our case study involves a traffic monitoring system (TMS [1]), which is an important part of a traffic control and logistics system for a dense traffic system. The main purpose of TMS is to record the positions of vehicles on the net. These recordings are used to adjust the schedules of running and planned vehicles as well as operating the necessary signalling. The system is business critical, but not safety critical.

TMS retrieves its data from multiple measurement sources. Its functionality includes distribution of information, occupation management, and track management. Amongst others, the TMS informs client systems with real-time, consistent, and unambiguous data about vehicle positions. It maintains vehicle movement information (identification and order) at the borders of unmonitored areas and maps this information to actual vehicle movements.

The TMS requirements have been set up by the TMS owner, the design and implementation is done by Logica. The design makes use of various UML models, and includes correctness proofs for critical state diagrams. The implementation is being written in C++.

3.3.2 Documentation Structure and Tooling

The interaction between the outsourcer, the owner of the TMS, and the outsourcing vendor, Logica, is organised around the MIL-std 498 documentation standard. This standard was developed at the United States Department of Defence to realize a common software development standard [Department of Defence, USA, 1994]. Only part of the standard is implemented in our case study (see Section 3.5.1).

The outsource vendor, in our case study, chose to use Rational$^{\text{TM}}$ tooling to support the requirement management activities as well as parts of its development processes. First of all Rational RequisitePro is applied for managing the requirements. It maintains a repository of requirements: their identification, description, and relations, augmented with other information such as their design rationale, and test criteria. The data is stored in an external database, in our case a Microsoft Access database. The tool has a close relation with Microsoft Word to capture and edit the requirements. Rational Rose is used to develop various UML diagrams, e.g., activity diagrams, collaboration diagrams, and class diagrams. These diagrams primarily describe the system design, but are also used to explain the details of a requirement. Unlike textual requirements, the diagrams are not

[1]Details of the case have been modified and made anonymous in order to protect the interests of the customer. We believe that these changes do not materially affect the experiences and results discussed in the chapter.

managed by Rational RequisitePro. Rational SoDA is applied for generating reports according to a prescribed template. The necessary information is taken from Rose and RequisitePro. Finally configuration management, including version and change control, is implemented using Telelogic Synergy, which manages all documents and repositories.

3.4 Outsourcing

In Section 3.2 we discussed requirements management in general, and in the previous section we provided the necessary background material (on traffic monitoring, the MIL-std 498, and tool support) required to analyse the case at hand. In this section, we analyse the implications of adopting explicit, tool-supported requirements management in an outsourcing context. We are not aware of other papers discussing these implications, although some material can be found in [Damian et al., 2003; Prikladnicki et al., 2003].

Outsourcing of system development, integration, and maintenance is an important trend in IT services [Abbas et al., 1997; Spanjers et al., 2006]. For a client, the outsourcer, contracting out work is a cost-effective way to hire expertise at a fixed price and get in return a system (or service) with a predefined quality, which includes timely delivery. The observable characteristics of the system, the requirements, are an essential part of the contract between the client and the outsource vendor. Requirements are accompanied by acceptance tests providing means to validate their correct implementation of the requirements.

The details and structure of the specification of the requirements have significant influence on the way of working for the outsource vendor. The short-term concern for the vendor is to establish a cost-effective way to comply with the obligations of the contract; covering the set of requirements and passing the acceptance tests. The vendor, however, also has long-term concerns, such as winning an extended contract for maintaining the developed system and winning similar contracts with other customers.

A number of issues have to be resolved for successful outsourcing of systems, which by Abbas *et al.* are summarised as: control, ownership, development paradigms, assurance, and system decomposition [Abbas et al., 1997]. Typically these issues are addressed in the contracts and agreements.

Control involves concerns like quality, security, and confidentiality. It also handles the responsibility of integration. Ownership is a complicating factor when changes to requirements or implementations are required. Development paradigms dictate compatibility and communication, while assurance defines acceptance tests. The responsibility of system decom-

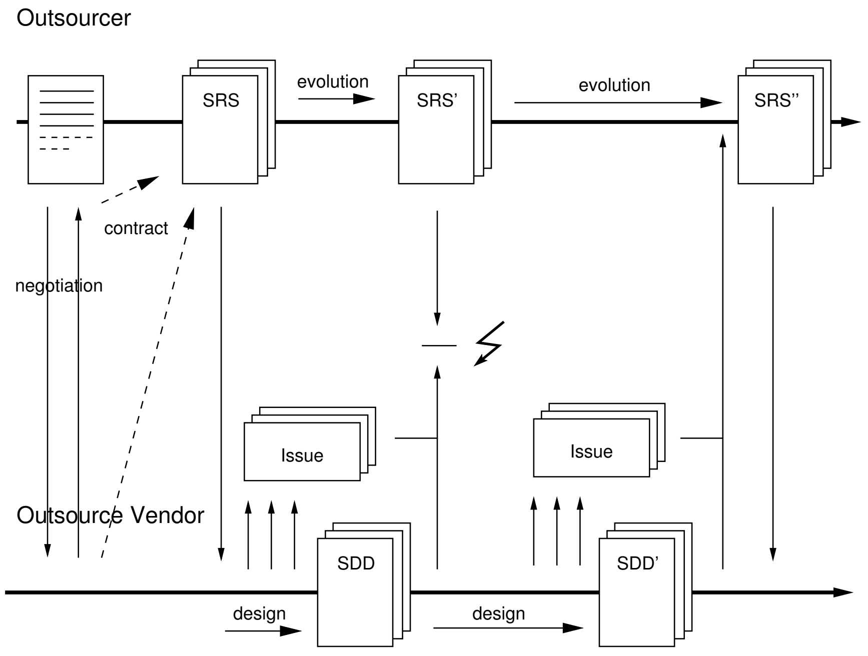


Figure 3.2: Requirements Engineering Process in an Outsourcing Context

position has to be clearly communicated because changes occur in every outsourcing contract. Changing client requirements, clarifications, or design tradeoffs may induce changes that need a modification to the chosen decomposition.

3.4.1 Requirements Engineering Process

In an outsourcing context the responsibilities in the requirements engineering process are distributed over the outsourcer and the outsource vendor. Figure 3.2 outlines the process and identifies possible hazards using the MIL-std 498.

The outsourcer executes the elicitation process and is responsible for the documentation of the requirements in a System Requirements Specification (SRS). This SRS is the basis of a negotiated contract between the outsourcer and the outsource vendor. Given the SRS, the actual development of the product is then done by the outsource vendor and documented in a System Design Description (SDD).

During development of the product the outsource vendor can run into some conflicting or ambiguous requirements. These are noted as *issues* and should be renegotiated with the outsourcer. Concurrently, the outsourcer develops new ideas that should also be implemented in the system (SRS′). The synchronisation of these parallel activities is a potential hazard. There are two basic resolutions: either the outsource vendor holds back the SDD and issues, or the outsourcer holds back the SRS′. In the first case the outsource vendor incorporates the SRS′ into the SDD, forming an updated version of the SDD′. In the second case the outsourcer resolves the issues of the SDD before releasing the updated version of the SRS. This latter case is illustrated in the second iteration of Figure 3.2. A hybrid version for this synchronisation process, although possible, yields a difficult process for keeping all the system artefacts consistent. Note that the evolution has two sources: advancing insights from the outsourcer and advancing insights from the outsource vendor.

3.4.2 Requirements Management Tool Implications

In the ideal case, the input to the RMS is a set of *frozen* TMS requirements, laid down in the contract with the client. With these correct, complete, and non-conflicting requirements, the system is implemented and simply passes the final system acceptance test. A project leader integrates the results of all tests to monitor the progress of the project and may instantiate the final system acceptance test. Passing this test ends the project.

However, practice is obstinate. Generally, the system requirements are incomplete and ambiguous. This has far reaching consequences: the RMS must support *evolution*. During the design process any ambiguity has to be resolved. This may yield an update, insertion, or deletion of an identified requirement. In other words the RMS must support a process that can handle changes effectively, including maintaining a consistent set of requirements.

With respect to the general features of a requirements management system, a RMS in the context of outsourcing must pay special attention to:

- Change management; project members have to be aware of the up-to-date baseline of requirements, evolutions, and anticipated changes.
- Quality assurance; the coherence and applied terminology of the requirements must be verified before delivering a design. Conflicting requirements need to be resolved in interaction with the outsourcer.
- Issue tracking; during the project questions and change requests are communicated (possibly over multiple channels). Their status and history need to be recorded and accessible.

- Test reporting; tests are induced by the requirements. The outsource vendor should be able to show via reporting if and how the requirements are fulfilled.
- Status reporting; status and other attributes of requirements should be translated to numbers and should be used in status reporting.
- Flexible modularisation; the system decomposition of the outsourcer does not necessarily comply with the preferred decomposition of the system by the outsource vendor. With flexible modularisation support the vendor can choose to internally use a different modularisation than used externally in the communication with the outsourcer.

The above features concern communication and interaction with stakeholders and is primarily the result of interviews with members of the development team of Logica.

3.5 Case Study Implementation

This section describes the requirements management process as implemented at Logica for the TMS case. Two important, previously introduced, concepts of this RMS are: the document structure MIL-std 498 and the tooling by Rational. In this section we will discuss how Logica applies these concepts. We first introduce the applicable parts of MIL-std 498, and then describe the traceability model that helps realising the requirements for the RMS of the outsource vendor. After that the tooling is discussed, and some case statistics presented.

3.5.1 Document Structure

The prime responsibility of the client is to provide the requirements, the prime responsibility of the vendor is to implement these requirements. Only part of the MIL-std 498 is implemented to provide the requirements in our case study. The essentials are captured in Figure 3.3.

The client provides the Operational Concept Description (OCD), System/Subsystem Specification (SSS), System/Subsystem Design Description (SSDD), and the System Requirements Specification (SRS) (see Figure 3.3). The client also provides the corresponding acceptance tests at super-system level including the System Test Plan (STP). All documents are plain-text Microsoft-Word documents.

The documents are the input for the development team of Logica. They split the documents into smaller units, and add design information. The SRS is used for creating a System Design Description (SDD) and System

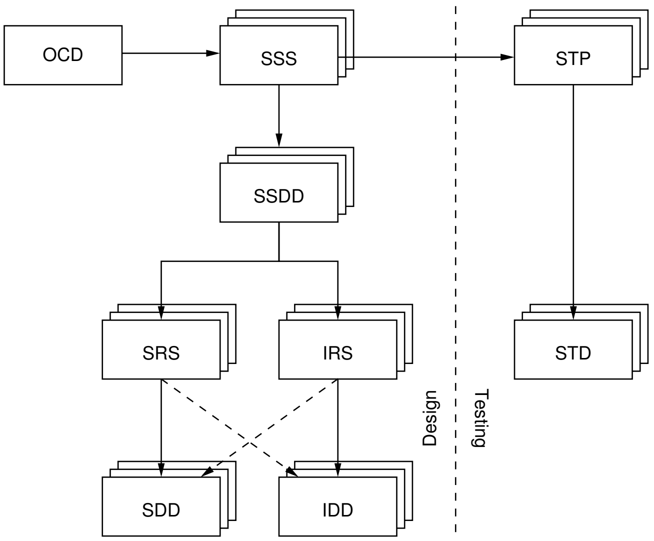


Figure 3.3: MIL-std 498 outline

Test Descriptions (STD). In fact the client and Logica used the SRS, SDD, and STD as system specifications rather than software specifications as is usual in the MIL-std 498.

3.5.2 Requirements Traceability Model

The traceability model consists of a number of different requirement types. We identified a number of them, which we believe are typical for the outsourcing business. The requirement types are: system requirement, issue, design decision, assumption, and non conformance.

A *system requirement* is a traditional requirement type that describes *what* the system should be able to do, but not *how* the system will do it. Logica categorised its system requirements into functional, non-functional, and design constraints, which are part of the SRSs and IRSs.

An *issue* is a type of requirement introduced by Logica that marks a point of attention that needs further investigation and possibly negotiation with the outsourcer. Issues are the communicating vehicle with the client and other stakeholders. An issue often affects multiple requirements as we will, e.g., see in Section 3.5.7.

A *design decision* records how the system will implement one or more requirements. The decisions recorded in a design decision can be very diverse, for example, decisions related to a technique, certain structure, or Commercial off-the-shelf (COTS) product.

An *assumption* articulates an implicit (external) constraint. Non compliance to these constraints would yield an incorrect functioning system. The relation between the design decision and assumption should be documented very clearly. Assumptions can result in additional system requirements.

The concept of a *non conformance* explicitly documents exceptions in choices made earlier in the development process. A reason for introducing such a non conformance can be that more information becomes available during the project. Design decisions, assumptions, and non conformances are all part of the SDDs, and IDDs.

A *design rationale* records the argumentation for the design choice taken, but it can also be used to record the argumentation behind an assumption or non conformance. An example of a design rationale is the argumentation for preferring a specific piece of middleware, in order to avoid high costs for licensing. Design rationales are defined as an attribute of a requirement and are *not* a requirement type.

The above types of requirements primarily relate to the design of the system. Similarly, Logica also defined requirement types for testing, namely *test criterion*, *test case*, and *test procedure*. A test criterion describes the conditions when a requirement has been successfully implemented in the system. It is possible to have more than one test criterion per requirement. A criterion can be met through one or more test cases, which are atomic tests. A test case describes the exact test situation, including preconditions, actions to be taken, and the condition under which the test case is successfully executed. Finally, the test procedure is a sequence of multiple test cases. The test cases and the test procedure are part of the STDs.

All aforementioned types of requirements are stored as uniquely identifiable entities in the RMS except for the design rationale, which is, as already mentioned, defined as an attribute of a requirement. Adding design rationales to the set of identifiable entities helps in two ways: it provides relations for traceability and it provides the necessary context for proper modifications.

The traceability relationships between the entities discussed above are summarised in Figure 3.4. The solid lines indicate the primary traceability links allowed to be set in the RMS and are n-to-n relations. Thus, each system requirement is linked to one or more design decisions, as well as to one or more test criteria. Likewise, each system requirement can be decorated with one or more issues concerning that requirement. The dashed lines give additional traceability links, that bypass the primary traceability relation. Thus, in principle test criteria come from system requirements, but in some cases a test criterion can be derived from a design decision as well.

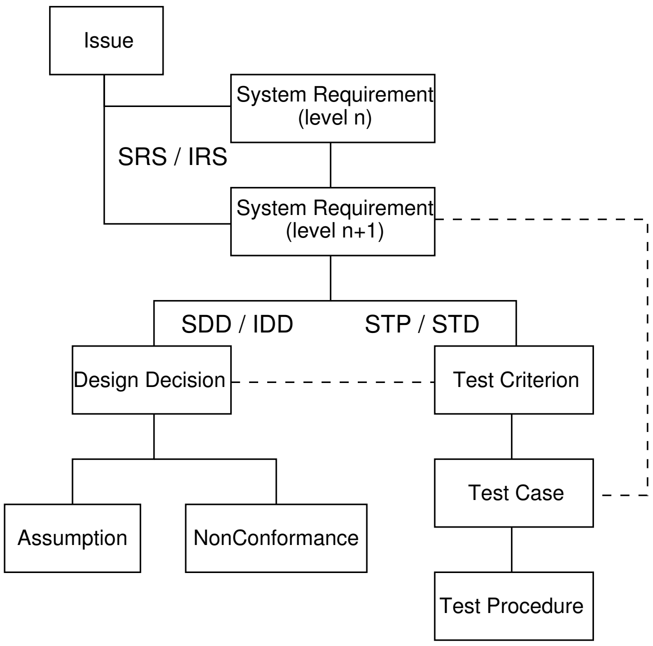


Figure 3.4: Requirement Traceability Model

Likewise, a system requirement should in principle be accompanied by a test criterion, but in some cases it is easier to directly provide the test case instead. Although advisable, it is not always possible to come up with a test criterion for every system requirement.

3.5.3 Instantiating the RMS

In order to adopt RequisitePro (see Section 3.3.2) in the TMS setting, it needs to be configured so that the described document structure and the traceability model fit in. Moreover, its configuration should support the appropriate entities and relationships as occurring in the traceability model.

For the TMS case, the documents were divided into three levels of abstraction. The first and highest level of abstraction includes the OCD, the SSS, the SSDD, and the STP documents. The second level includes all the SRS, and IRS documents. The final and most detailed level includes the SDD, the IDD, and the STD documents. For every requirement attributes have been defined. Besides the obligatory internal id, also attributes such as design rationale, priority, status, and stability are defined in the tool. A particularly important attribute is the identifier provided for each requirement by the client – these identifiers provide the traceability to the requirements documents from the client. Thus, the internal identifier generated by the RMS is effectively ignored, and replaced by the client-provided identifier.

Observe that similar steps are required if another requirements management tools had been chosen, for example, DOORS from Telelogic.

3.5.4 Populating the RMS

After the RMS has been properly configured, it needs to be populated with the actual requirements. RequisitePro supports an interactive process in which paragraphs in MS Word documents can be marked as requirements, after which the type of requirement and the values for the corresponding attributes can be provided. In addition to that, traceability links to other requirements can be declared. Since the original MS Word documents have no strict structure, this process is largely manual, and hard to automate.

3.5.5 Updating the Requirements

Users of RequisitePro can modify requirements by navigating through the set of requirements and selecting a particular requirement for modification, which opens the requirement in the originating Microsoft Word document. Executing such an update, e.g., repair a typo or change a requirement statement, marks that particular requirement as changed. The traceability model now helps to trace the impact of that change through the rest of the system.

Unfortunately, in the TMS setting the client was the owner of the requirements documents , and changes made by the vendor will lead to inconsistencies with the version maintained by the client. Hence, changes to the requirements must come from the client, who provides a full new version of the requirements documents. The new set then must be analysed by hand, and changes to the underlying requirements and traceability links as stored in the RMS must be carefully executed. Although some help can be provided by taking the differences between two documents, this remains a cumbersome and error prone process.

3.5.6 Report Generation

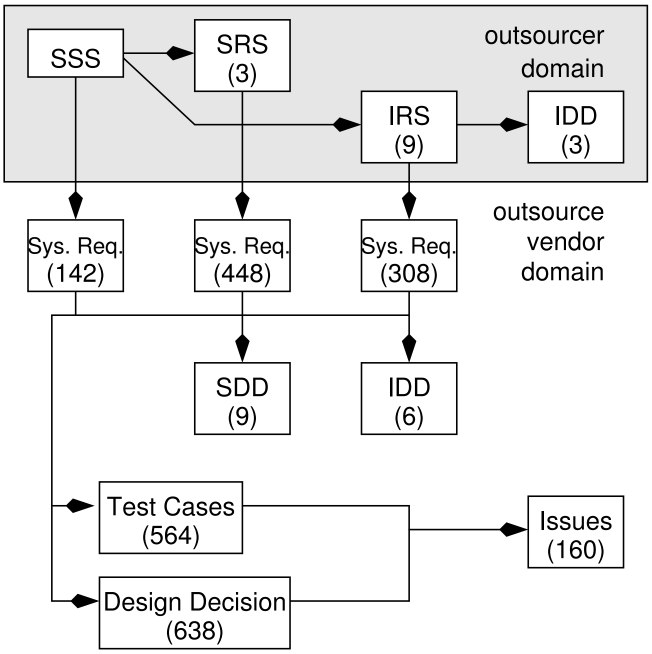
Logica defined various reports to be generated from the RMS to support the various project team members. These reports are generated to support reviews, testing, design, and project management. Reports are defined for actors such as the project manager, test manager, designers, and requirements manager. Examples of these reports are generating an overview of all issues including their status for the project manager, generating an overview of all system requirements covered by test procedures for the test manager, generating traceability matrices for the designer, and generating an overview of all requirements concerning a subsystem for the requirement manager.

The generated reports can be used for a basic form of systematic analysis or, for example, to verify the consistency, or correctness of requirements. RequisitePro offers limited functionality for this; if the offered support is insufficient separate software operating directly on the underlying database and the traceability links contained therein should be written (which amounts to bypassing RequisitePro).

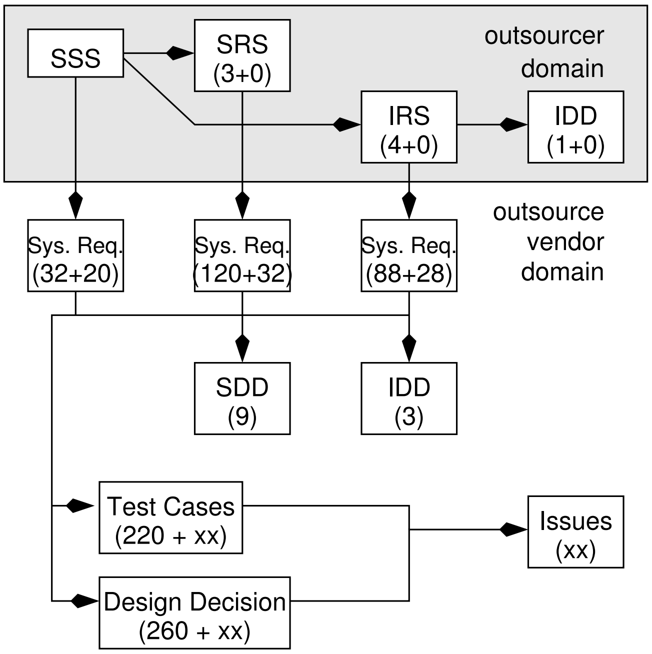
3.5.7 Case Statistics

The outsourcer provided the requirements documentation which is decomposed into 1 SSS, 3 SRS, 9 IRS, and 3 IDD documents. Subsequently the outsource vendor used the traceability model of Figure 3.4 to structure the requirements from the SRS and IRS documents.

The development process involved only one iteration at the time of writing. During the first phase 160 issues emerged which had to be resolved with the client. Issues ranged from clarifications through inconsistencies and from adding and removing modules to restructuring of the modularisation. The development process particularly did not involve analysis of the



(a) Phase I, (s) indicates the number of requirements



(b) Phase II, (s + t) indicates s updated and t new requirements; xx is unknown

Figure 3.5: Case Statistics

received requirements, which potentially could have resolved a number of issues in an early stage of the development. An example of an issue is the following: One of the requirements explicitly mentioned the version number of a middleware component, however during design a different (older !) version of this component was believed to have significant advantages. The corresponding design decision thus had a different number than the originating requirement. The issue is easily resolved but has to be clearly communicated in order to keep the involved SSS consistent.

After the first phase the issues were fed back to the client. The subsequent contract renegotiations yielded a new set of SRS documents that roughly increased the number identified requirements by 80 and significantly modified 560 requirements. Figure 3.5a shows the statistics of the case study for the first phase, and Figure 3.5b shows the situation after the first iteration. As an example 120 system requirements that originate from an SRS were updated or deleted, and 32 new ones were added. As a result the requirement database holds about 2400 related requirements. The system holds about 4700 traceability links.

Importing and updating the requirements from the respective SRS and IRS documents is a partly manual process as described in the previous subsection. The initial import as well as the update after renegotiations both took 5–6 men days.

3.6 Discussion

Having described the way in which requirements management was conducted for the TMS system, we can now discuss selected observations, and distil a number of key lessons learned. We propose a requirements engineering framework that addresses most of our identified concerns.

3.6.1 Observations

Import of Requirements

Importing and updating of requirements is related to change management. The *import of the semi-structured requirements* into the RMS is a troublesome process, which is why an automatic process is preferred. It takes quite some manual work to meet the requested quality. Although the input requirements are structured, they are not formalised. Diagrams, for instance, play an important role in clarifying requirements. But since they lack formalisation, diagrams have been left out of the requirements management system.

Update of Requirements

Updating the set of requirements is an entirely manual process. It starts from the differences between the updated and initial version, and changes are then updated in the RMS manually. The update process typically interferes with the tracing capabilities of the toolset; all links are invalidated, which makes the update a cumbersome and error-prone process.

Change Management

This problem partly stems from the fact that the change management system does not distinguish between *types of changes*. Thus fixing a typo propagates through the traceability links in exactly the same manner as a drastic change such as decimating the available latency for some action. Categorisation or modularisation of types of changes should be supported by the RMS.

Issue Management

With respect to issue tracking, and quality assurance, we observed that Logica uses *issues* to communicate with its outsourcer. Resolved issues are signed by both parties to protect them from potential legal conflicts. This process is satisfactory for both parties, although the problematic update inside the RMS remains. A risk exists that resolved issues are not correctly updated in the RMS, which renders the set of requirements inconsistent or even incorrect.

Reporting

Status reports are an example of a report type that is generated from the RMS to support the communication with stakeholders. These reports are used in project management for status tracking and issue tracking. Other reports are used for reviewing and analysing requirements for correctness, possible ambiguities, and consistency. The generation of these reports requires tailoring. The available tool, Rational SoDA, has several limitations, such as poor quality reports, limited flexibility, and long processing times. These limitations in *reporting* of Rational SoDa, make Logica to consider developing a new tool, which uses a more advanced query system than currently offered by RequisitePro, to overcome these issues.

Navigation and Browsing

As a consequence of this lack of proper reporting capabilities, developers use the *navigation facilities* of RequisitePro instead of a generated report to locate information about requirements. The browser offers the opportunity to search through the complete set of requirements. Access control was not necessary to be implemented in this particular case study. All members of the development team within Logica are allowed the read and edit the complete set of requirements in the outsource vendor domain.

Flexibility in Generating Views

Furthermore, the browser, although flexible, is not able to provide all required views. For example a view of the decomposition of a high-level requirement throughout the system shows the complete traceability path, whereas developers may only want to see part of it. *Flexibility in generating views* is essential, as views play a key role in the communication and interaction with stakeholders. This especially holds in an outsourcing context where clients have very specific demands for communication.

Tracking and Tracing

Tracking and tracing are important assets of an RMS. The traceability model helps to avoid a 'spaghetti' of traceability links. There is however the risk of inconsistency in the links, in part because of the manual process of providing the links. Links should also have a rationale that explains why a particular link exists. Making a link a first class element of the traceability model would solve this issue and potentially decrease the risk of an inconsistent set of requirements.

Status Control

The current traceability model (Figure 3.4) supports detailed attributes for tracking, including a status attribute. In practice, however, the status attribute of every requirement is not used. It is for Logica sufficient to report the status of a complete requirements document, e.g., a SRS. This means that *status control* per requirement is not implemented at Logica. Still, the need is present to report the status of individual requirements.

Requirements Analysis

The RMS has very limited support for *analysis of requirements*. Although the traceability model facilitates impact analysis and coverage control, it lacks support for other types of analysis such as conformance checking. *Conformance, and correctness checking* are now implement by a review process. This again emphasises the importance of high quality report generation.

Flexibel Modularisation

Flexibility of modularisation is an important feature of an RMS in the context of outsourcing. In our case study, the requirements were already grouped into subsystems by the client. The outsource vendor executed the design step from SRS to SDD. During this design step developers objected to the initial modularisation. However, since the client owns the high-level design documents, their objections had to be negotiated with the client, which takes time. Three possible solutions are: 1) the outsourcer hands over ownership of the high level design documents; 2) the high level design documents are discarded and issues are resolved in the domain of the RMS; or 3) the RMS supports flexible modularisations, which makes it easier to align and synchronise.

Alignment of Development Environments

This previous observation is directly related to our final observation. Both the client and the outsource vendor have configured their own *development environment*, while they are developing one system in collaboration. Taking into account the previous observations concerning the import of requirements, and reporting from a RMS, our final observation is that the synchronisation and alignment data within different environments is hard to manage.

In the next section, we will map these observation to lessons learned for developing software systems in an outsourcing context.

3.6.2 Lessons Learned

From our observations the following lessons should be taken in consideration when developing software systems in an outsourcing context:

1. Flexible modularisation and generation of views needs to be supported. Current tool support is not sufficient to satisfy these needs.

2. Explicit issue notes are an effective means for communication. Their role needs to be explicit in the requirements management process and the tracking of these issues should be supported.

3. Tracking can be implemented effectively through an appropriate traceability model. Links between requirements in such a model must be first class elements.

4. Due to the distributed requirement engineering process in an outsourcing context, synchronisation of activities must be addressed ex-

plicitly. These activities include modifying and updating the set of requirements.

3.6.3 A Conceptual Framework for Reconciling Requirements Evolution and Outsourcing

We now propose a conceptual framework of a Requirements Engineering System (RES) that implements the lessons learned discussed above. The primary purpose of this framework is to bridge the gap between the need for evolution of requirements, and the evolution impediments that are generated by the adoption of outsourcing. Parts of our proposed framework recur in existing commercial tool sets but none of them satisfies all features [Graaf et al., 2003].

We refer to the proposed framework as a requirements engineering system because typical specification activities such as analysis are also incorporated. The heart of our framework, depicted in Figure 3.6, is the requirements model with corresponding traceability model. In general this model is modularised according to a template, e.g. MIL-std 498 or IEEE-std 830-1998, used as document structure by the client. For every requirement entity attributes are defined and these entities are related according to the traceability model.

The requirements model allows for a structured analysis and updates through automatic tools. Dynamic modularisation is an important aspect of the proposed framework. The document structure is one modularisation, the (software) system decomposition is another. The underlying traceability model can generate multiple *views*, each taking a different perspective with corresponding partitioning and clustering of requirements and traceability links. This makes a view an ideal means for communicating issues with the outsourcer.

Traceability links themselves are first class entities in this model. Thus, in the traceability model the traceability links have a unique identifier as well as other attributes such as design rationale, and maybe even a description.

Another important aspect of the framework is the ability to transform the contents of the requirements model to an external structured document based on a template (e.g., by means of *forms*). Outsourcers mostly provide unstructured or semi-structured documents to the outsource vendor.

Incorporating the various changes originating from the unstructured environment poses a challenge for the structured environment. The outsource vendor and the customer operate in parallel, and in different development environments. Outsource vendors are not in the position to force outsourcers to adapt to their requirements engineering process, giving rise

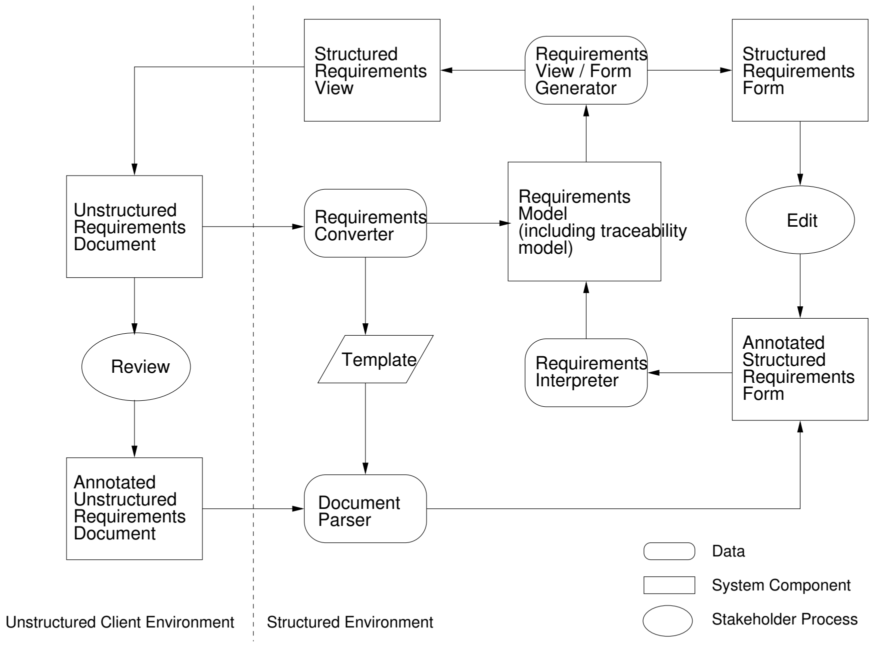

Structured Requirements View
Requirements View / Form Generator
Structured Requirements Form
Unstructured Requirements Document
Requirements Converter
Requirements Model (including traceability model)
Edit
Review
Template
Requirements Interpreter
Annotated Structured Requirements Form
Annotated Unstructured Requirements Document
Document Parser
Data
System Component
Stakeholder Process
Unstructured Client Environment
Structured Environment

Figure 3.6: Proposed Requirements Engineering Framework

to the need for a conversion process. In our framework the conversion is done only once. After this conversion process requirements management and engineering is done in the structured environment. Therefore we need to reproduce a look-a-like of the initial unstructured document that is structured and can be edited. We refer to these as *forms*. To produce these forms a template is extracted during the conversion process, which is used to parse the annotated requirements documents, so the changes can automatically be imported into the RES. Forms are used to interact with the stakeholder, e.g., to facilitate editing. Once edited, the changes made can easily be interpreted and imported in the RES ensuring consistency of the set of requirements.

Views cannot be edited by the stakeholders and are primarily a means for communication and analysis. The process described is repeatable during the development life-cycle as well as during maintenance of the system and supports the management of evolving requirements.

To summarize, our RES framework addresses three essential processes with respect to managing requirements evolution:

- The interaction with stakeholders. This process is covered by the requirements converter, the editable forms, and the document parser. It together creates the possibilities to import and export data in a structured way and corresponds with some of the required the features of an RMS discussed in Section 3.2.2.

- The consistent processing of changes. The requirements model provides all features necessary for covering this process. It covers traceability support, an analysis system, modularisation support, etc...

- The presentation of information using views. This last process addressed by RES is covered by the requirements view / form generator together with its corresponding views and forms.

If we map the RES framework to the steps Logica implemented in Section 3.5, we can recognize that each of the steps is covered by the RES framework. The first process of RES covers the implementation steps “instantiating” and “populating” the RMS (Sections 3.5.3 and 3.5.4). The second process of RES concern the “document structure”, “traceability model”, and “updating the requirements” (Sections 3.5.1, 3.5.2, and 3.5.5). Finally, the third process covers Section 3.5.6, “report generation”.

This mapping shows that our RES framework should cover the problem of managing evolving requirements in an outsourcing context. The actual value of the RES framework stays an open issue for the moment and future work should prove the real value of the RES framework in practice.

3.6.4 External Validity

We conclude our discussion by analyzing the validity of the proposed RES framework in a wider context. In the previous section, we stated that RES covers the software development process for an outsourcing context. However, this is based on the lessons learned from a single case study at Logica. Issues to consider when generalizing the lessons learned include:

1. The process adopted in the case did not include steps that are specific or proprietary to Logica. All steps of the process can be identified in standard handbooks on software engineering such as [Pressman, 2005; Sommerville, 2001; van Vliet, 2008]. Therefore, we expect the approach to be applicable in other companies as well.

2. The framework is independent of actual tools used. While the case at hand made use of Rational RequisitePro, other tools, such as Telelogic DOORS or Borland Together would easily fit as well.

3. In the case study at hand, the document structure was based on MIL-std 498. The RES framework does not prescribe a specific document structure, such as MIL-std 498. However, it does assume a certain documentation discipline.

4. In outsourcing projects, structured documentation is more likely to occur than in regular projects. As the framework requires a certain documentation discipline, it is less suitable to, for example, extreme programming projects that use test cases as their primary form of documentation.

3.7 Contributions and Future Work

In this chapter we have discussed how Logica has implemented requirements management in order to support requirements evolution for the traffic monitoring system, which Logica is implementing for an external customer. The customer took care of requirements elicitation and analysis, and decided to outsource the development to Logica. The focus of this chapter is on analysing the implications of outsourcing on requirements management methods and tools.

We consider the following to be our key contributions. First of all, we discussed how requirement management was tackled in a fixed price contract between a client who created the requirements in the first place and an outsourcing vendor who was responsible for building the system. Relevant described results include the use of *issues*, the adopted traceability model, and the discussion of features that a requirements management system applied in an outsourcing context should have. We believe that sharing these experiences from Logica is valuable for both researchers and practitioners in the area of requirements management.

Second, we identified a number of problems with the requirements management methods and tools adopted. These problems concern the transition of requirements from the unstructured to the structured domain, the need to redo this transition upon requirements evolution, inadequate reporting facilities, and lack of sufficiently flexible modularisation support.

Last but not least, we proposed a framework that can be used as a research vehicle for addressing the concerns raised in this chapter. Our framework facilitates multiple views, imports by means of forms and templates, and explicitly separates the structured from the unstructured environment. The elaboration of this model is the focus of our current research.

The proposed RES framework and the observations give rise to several topics to investigate further in future research:

- How to determine the template (structure) of a semi-structured requirements document,
- How to specify the traceability model such that it has clear semantics,
- How to generate views for analysis, and conformance checking,
- How to interpret an annotated form,
- How can we parse an annotated document to a form according to the derived template, and
- Does this framework reduce the risks of introducing errors during evolution of requirements.

These research opportunities should be investigated in close cooperation with industry. Logica and other members of the MOOSE and MERLIN consortium [MOOSE, 2004; MERLIN, 2005] should provide a fertile experimental ground to arrive at answers to these questions.

3.8 Epilogue

In this chapter, we studied an industrial case study in detail. This investigation confirms the results from Chapter 2. Two of the three reasons given in Chapter 2 can be identified again in this case study, namely maturity and complexity. Compliance with legacy was not an issue in this case study

Firstly, the case study showed us that it is indeed difficult to apply a commercial tool suite in practice. The issues identified in Section 2.6 of Chapter 2, such as difficulties in managing complex dependencies in a RMS, and the issues of change management, are problematic in this chapter as well.

Secondly, the complexity reason can indirectly be identified in this case study as well. The customers (external stakeholders) of Logica, are not interested in the complex tooling of Logica. So, the complex development environment of Logica needs to be hidden from its customer. Consequently, Logica has difficulties managing the requirements and keeping the documents consistent as describe in the case study.

The RES framework is a result of the three reasons for not using state-of-the-art technology discusses in Chapter 2 and the lessons learned of the case study studied in this chapter. The first process of RES should make sure that any legacy approach can be imported in the RES framework structure. Furthermore, complexity is hidden in the internal templates and requirements model. Maturity is not covered by the RES framework as the framework does not fill in any technology. It only defines process.

In the next Chapter 4, we will focus on the requirements model of our RES framework, and in particular the traceability model. We will investigate the possibilities to automate the reconstruction of the traceability model. A method will be defined that covers this reconstruction process and it will be implemented in a tool suite.

Chapter 4

Reconstructing Requirements Traceability in Design and Test using Latent Semantic Indexing[1]

Managing traceability data is an important aspect of the software development process. In this chapter we define a methodology, consisting of seven steps, for reconstructing requirements views using traceability data. One of the steps concerns the reconstruction of the traceability data. We investigate to what extent Latent Semantic Indexing (LSI), an information retrieval technique, can help recovering the information needed for automatically reconstructing traceability of requirements during the development process. We experiment with different link selection strategies and apply LSI in multiple case studies varying in size and context. We discuss the results of a small lab study, a larger case study and a large industrial case study.

4.1 Introduction

For many organisations, the purpose of requirements management is to provide and maintain clear agreements between the different stakeholders involved in developing a product, while still allowing requirements evolution. Requirements management is a supportive process that starts in the orientation phase and continues during the rest of the product development life-cycle.

[1]This chapter was originally published as: Marco Lormans and Arie van Deursen. Can LSI help reconstructing requirements traceability in design and test? In *Proc. of the 10th European Conf. on Software Maintenance and Reengineering*, pages 47–56, Bari, Italy, March 2006. IEEE Computer Society

Managing requirements in such a way that useful information can be extracted from this requirements set, is hard in practice [Graaf et al., 2003; Huffman Hayes et al., 2006]. This extracted information can be used for many applications such as generating requirements views or impact analysis [Cleland-Huang et al., 2003]. The requirements views we will encounter are coverage views, which include whether or not a requirement is covered by an acceptance test, by a design artefact, by a system test, and so on. These requirement views can provide a major asset for developers and project managers, offering them a way to monitor the progress of the development process.

Obtaining accurate information requires that an up-to-date traceability matrix is maintained, establishing links between, for example, requirements and test cases. Keeping the traceability links consistent during development of the product is a time consuming, error-prone, and labour-intensive process demanding disciplined developers [Brinkkemper, 2004; Gotel and Finkelstein, 1994; Lormans et al., 2004]. Currently available tools do not support the feature of automatically recovering traceability links [Alexander, 2002; Graaf et al., 2003; Huffman Hayes et al., 2006].

In this chapter, we investigate to what extent relevant traceability links can be reconstructed automatically from available documents using latent semantic indexing (LSI). LSI is an information retrieval method assuming there is a latent semantic structure for every document set [Deerwester et al., 1990]. LSI creates a "semantic" subspace of terms and documents closely associated using statistical techniques. This subspace can be used for retrieving information and, in our case, for reconstructing traceability links. For this reason, our approach assumes a document-oriented requirements engineering process based on natural language, which can identify semantic similarities between different documents produced during the development of the product.

The long term objective of our research is to determine how industry can benefit from using LSI to track and trace requirements and eventually generate various requirements views. In this chapter, we describe three exploratory case studies, that give answers to the following questions:

1. Can LSI help in reconstructing meaningful traceability relations between requirements and design, and between requirements and test cases?

2. What is the most suitable strategy for mapping LSI document similarities to reconstructed links?

3. What are the most important open issues that need to be resolved before LSI can be applied successfully in an industrial context?

To answer these questions we offer, in every case study, an analysis why the particular links can be reconstructed and what the requirements are for documenting the related development work products.

The three case studies in which we applied LSI, vary in size and context. The first is a lab study, Pacman, used in a testing course at Delft University of Technology. Available documentation includes use cases, design decisions, acceptance test cases, as well as a Java implementation with a JUnit test suite. The Pacman case gives us the opportunity to explore all the possibilities of the techniques in a controlled environment. In this study, we varied the different parameters of our analysis to come to a setting giving the best results. The second case study is part of a software engineering course at Eindhoven University of Technology. In this course, a group of students need to develop a complete new software system from scratch and properly document requirements, design and test cases including traceability links. The last case study is an industrial case study carried out at Philips Applied Technology. In this study, requirements, design decisions, and corresponding test suite for a Philips DVD recorder were analyzed.

The remainder of this chapter is organized as follows. In Section 4.2 we give an overview of background information and discuss related work, followed by a brief survey of latent semantic indexing in Section 4.3. In Section 4.4 we describe our link reconstruction methodology, MAREV, and in Section 4.5 we describe the link selection strategies we use in this methodology. Next, in Section 4.6 we describe the tool we developed to support our approach. The three cases studies are presented in Section 4.7. In Section 4.8 we compare and discuss the results of the case studies. We conclude the chapter by summarizing the key contributions and offering suggestions for future research.

4.2 Background and Related Work

4.2.1 Requirements Views

The different perspectives on requirements are often represented using views. Views capture a subset of the whole system in order to reduce the complexity from a certain perspective. For example, Nuseibeh *et al.* discuss the relationships between multiple views of a requirements specification [Nuseibeh et al., 1994]. This work is based on the viewpoints framework presented by Finkelstein *et al.* in [Finkelstein et al., 1992]. This framework primarily helps organizing and facilitating the viewpoints of different stakeholders.

Von Knethen also discusses view partitioning, but from a different perspective [von Knethen, 2001]. She considers views on the system, distinguishing, e.g., the static structure from the dynamic interactions in the system.

If we look beyond requirements, the concept of "view" also appears in the area of architectural design. Kruchten introduced his "4 + 1 view model" for architecture, where he defined five different concurrent perspectives on a software architecture [Kruchten, 1995]. Each view of this model addresses a specific set of concerns of interest to different stakeholders. Other examples are the "Siemens' 4 views" by Hofmeister *et al.* [Hofmeister et al., 1999], the IEEE standard 1471 [IEEE, 2000], and the views discussed by Clements *et al.* in their book "Documenting Software Architectures" [Clements et al., 2002, 2003]. Finally, Van Deursen *et al.* discuss a number of specific views for architecture reconstruction [van Deursen et al., 2004].

Although much research is done in the area of (requirements) views, there is no general agreement on what these views should look like or what information they should contain. Every project setting seems to have its own specific information needs concerning requirements.

4.2.2 Requirements Traceability and Reference Models

Managing different requirements perspectives (views) can be supported through appropriate meta-models, as shown by Nissen *et al.* [Nissen et al., 1996]. An important area of research in the domain of traceability is developing these meta-models. These so called reference models discussed in [Ramesh and Jarke, 2001; von Knethen, 2001; Toranzo and Castro, 1999; Maletic et al., 2003; Zisman et al., 2003] define the development artefacts including their attributes, and the traceability relations that are allowed to be set between these artefacts.

Von Knethen proposes (conceptual) traceability models for managing changes on embedded systems [von Knethen, 2001; von Knethen et al., 2002]. These models help estimating the impact of a change to the system or help to determine the links necessary for correct reuse of requirements. According to Von Knethen, defining a workable traceability model is a neglected activity in many approaches. Our earlier research confirms the importance of defining a traceability model [Lormans et al., 2004]. The initial experiments concerned a static traceability model. New insights suggest a dynamic model, in which new types of links can be added as the way of working evolves during the project. The need for information as well as the level of detail changes [Dömges and Pohl, 1998].

4.2.3 Traceability Reconstruction

To reconstruct coverage views from project documentation we need some traceability support. Several traceability recovery techniques already exist, each covering different traceability issues during the development lifecycle. Some discuss the relations between source code and documentation, others the relations between requirements on different levels of abstraction.

Antoniol *et al.* use information retrieval (IR) methods to recover the traceability relations from C++ code onto manual pages and from Java code to requirements [Antoniol et al., 2002]. Marcus and Maletic, and Di Penta *et al.* use latent semantic indexing for recovering the traceability relations between source code and documentation [Marcus and Maletic, 2003; Marcus et al., 2005a; Di Penta et al., 2002]. The IR methods in these cases are mostly applied to reverse engineering traceability links between source code and documentation in legacy systems.

IR methods can also be used for recovering traceability links between the requirements themselves [och Dag et al., 2004; Huffman Hayes et al., 2003]. In these cases, traceability recovery is mainly used for managing the requirements after development when all the documentation needs to be finalized and released. Both Natt och Dag *et al.* and Huffman Hayes *et al.* have developed a tool to support their approach. In [och Dag et al., 2005] Natt och Dag *et al.* discuss their approach and tool, called ReqSimile, in which they have implemented the basic vector space model and applied it in an industrial case study. Huffman Hayes *et al.* have implemented various methods for recovering the traceability links in their tool called RETRO [Huffman Hayes et al., 2005, 2006]. They also applied their approach in an industrial case study.

De Lucia *et al.* present an artefact management system, which has been extended with traceability recovery features [De Lucia et al., 2004, 2005]. This system manages all different artefacts produced during development such as requirements, designs, test cases, and source code modules. De Lucia *et al.* also use LSI for recovering the traceability links. In [De Lucia et al., 2006b], they improved their traceability recovery process and propose an incremental approach. In this approach they incrementally try to identify the “optimal” threshold for recovering traceability links.

Cleland-Huang *et al.* define three strategies for improving the dynamic requirements traceability performance: hierarchical modelling, logical clustering of artefacts and semi-automated pruning of the probabilistic network [Cleland-Huang et al., 2005]. They are implementing their approach in a tool called Poirot [Lin and *et al.*, 2006]. Furthermore, like De Lucia *et al.*, they have defined a strategy for discovering the optimal thresholds [Zou et al., 2004].

Finally, IR techniques are also used for improving the quality of the requirements set. Park *et al.* use the calculated similarity measures for improving the quality of the requirements specifications [Park et al., 2000].

4.3 Latent Semantic Indexing

Latent Semantic Indexing (LSI) is an information retrieval technique based on the vector space model and assumes that there is an underlying or latent structure in word usage for every document set [Deerwester et al., 1990]. This is particularly caused by classical IR issues as *synonymy* and *polysemy*. Synonymy concerns the fact that there are many ways to refer to the same object. Users in different contexts, or with different needs, knowledge, or linguistic habits will describe the same information using different terms. Polysemy involves the fact that most words have more than one distinct meaning. In different contexts or when used by different people the same term takes on varying referential significance [Deerwester et al., 1990].

LSI uses statistical techniques to estimate the latent structure of a set of documents. A description of terms and documents based on the underlying latent semantic structure is used for representing and retrieving information. This way LSI partially overcomes some of the deficiencies of assuming independence of words, and provides a way of dealing with synonymy automatically.

LSI starts with a matrix of terms by documents. Subsequently, it uses Singular Value Decomposition (SVD) to derive a particular latent semantic structure model from the term-by-document matrix [Berry et al., 1999; Salton and McGill, 1986]. Any rectangular matrix, for example a $t \times d$ matrix of terms and documents, X, can be decomposed into the product of three other matrices:

$$X = T_0 S_0 D_0^T$$

such that T_0 and D_0 have orthonormal columns and S_0 is diagonal (and D_0^T is the transpose of D_0). This is called the singular value decomposition of X. T_0 and D_0 are the matrices of left and right singular vectors and S_0 is the diagonal matrix of singular values.

SVD allows a simple strategy for optimal approximate fit using smaller matrices. If the singular values in S_0 are ordered by size, the first k largest values may be kept and the remaining smaller ones set to zero. The product of the resulting matrices is a matrix X' which is only approximately equal to X, and is of rank k. Since zeros were introduced into S_0, the representation can be simplified by deleting the zero rows and columns of S_0 to obtain a new diagonal matrix S, and deleting the corresponding columns of T_0 and D_0 to obtain T and D respectively. The result is a reduced model:

$$X' = TSD^T \approx X$$

which is the rank-k model with the best possible least square fit to X [Deerwester et al., 1990].

Note that the choice of k is critical: ideally, we want a value of k that is large enough to fit all the real structure in the data, but small enough so we do not also fit the sampling error or unimportant details. Choosing k properly is still an open issue in the factor analytic literature [Deerwester et al., 1990]. Our choice will be discussed when we apply LSI in our case studies.

Once all documents have been represented in the LSI subspace, we can compute the similarities between the documents. We take the cosine between their corresponding vector representations for calculating this similarity metric. This metric has a value between [-1, 1]. A value of 1 indicates that two documents are (almost) identical.

These measures can be used to cluster similar documents, or for identifying traceability links between the documents. We can also define new queries and map these into the LSI subspace. In this case, we can identify which existing documents are relevant to the query. This can be useful for identifying requirements in the existing document set.

Finally, LSI does not rely on a predefined vocabulary or grammar for the documentation (or source code). This allows the method to be applied without large amounts of pre-processing or manipulation of the input, which drastically reduces the costs of traceability link recovery [Maletic et al., 2003; De Lucia et al., 2004]. However, some text transformations are needed to prepare the documentation to form the corpus of LSI. This user-created corpus will be used as the input for creating the term-by-document matrix.

4.4 MAREV

The long term objective of our work is an approach that supports large organizations in the software industry in managing requirements throughout the life-cycle of, for example, consumer electronics products or document systems such as copiers. Such products need to fulfil hundreds or thousands of requirements. Furthermore, these requirements can change over time when new product versions are created and shipped.

Our focus is on reconstructing *requirements views*, i.e., views on the set of requirements that can be used to monitor the progress in requirements development, design, and testing. In this chapter we focus on the reconstruction of requirements traceability needed to generate the requirements views.

In order to answer the questions raised in the introduction, we conducted three case studies, which are described in Section 4.7. In this section we discuss 1) the steps of MAREV [1], our methodology to reconstruct the traceability links and generate requirements views, and 2) the approach we used to assess the reconstructed links. In Section 4.5 we describe the link selection strategies (step 5) in more detail and in Section 4.6 we discuss the tool that we developed in order to carry out these steps.

4.4.1 Link Reconstruction Steps

We have developed an approach for reconstructing our requirements views automatically. In this particular case we experimented with LSI for reconstructing the traceability links (step 4), which resulted in reasonably good traceability recovery results. The steps are:

1. Defining the underlying traceability model;
2. Identifying the concepts from the traceability model in the available set of documents;
3. Pre-processing the documents for automated analysis;
4. Reconstructing the traceability links;
5. Selecting the relevant links;
6. Generating requirements views;
7. Tackling changes.

In this chapter, we will primarily focus on techniques for executing step 4 and 5 handling the traceability recovery and selection of correct links. Of course, step 1, 2 and 3 are of major importance for executing step 4 and 5 successfully. We have defined some requirements views for step 6, and finally, for step 7, we describe how to tackle requirements evolution (changes in requirements). The last two steps remain future work for now. We will discuss all steps briefly and then focus on the steps 4 and 5 in the case studies.

[1]MAREV: A Methodology for Automating Requirements Evolution using Views

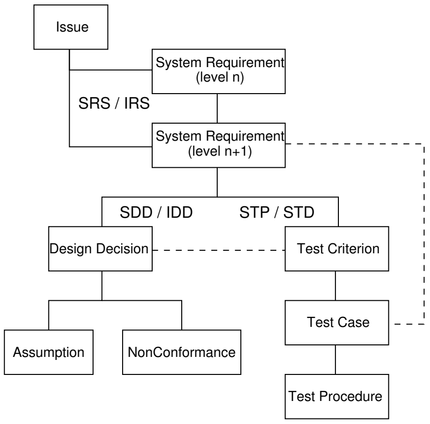


Figure 4.1: Traceability Model

Traceability Model Definition

Traceability relations establish links between requirements on the one hand and various types of development work products on the other. A traceability model defines the work products and the types of links that are permitted within the development process.

The choice of traceability model mainly depends on the purposes for which it is to be used. For example, Ramesh and Jarke [Ramesh and Jarke, 2001] discuss a range of different traceability models. Other examples of reference models can be found in [Maletic et al., 2003; von Knethen, 2001; Toranzo and Castro, 1999; Zisman et al., 2003].

An example of a traceability model relevant for coverage monitoring is shown in Figure 4.1. This model and the work products, including their dependencies, contained in it reflect the way of working at a big industrial company, Philips Applied Technologies, in the embedded systems domain. For example, it distinguishes between a *customer requirement* (cast in terms familiar by customer representatives) and *technical requirements* (cast in terms familiar by developers). Moreover, the model supports *evaluation* of requirements: after shipping the product, field studies are conducted in order to evaluate the working of the requirement in real life. The evaluation results are taken into account when shipping a new version of the product. This traceability model enables us to derive, for example, the following coverage information that can be included in a requirements view:

- **Identification coverage**; The information in this view indicates the links between customer requirements and technical requirements. The technical requirements specify the product that actually will be built. Every system requirement should have a link with at least one customer requirement, and vice versa.

- **Design coverage**; Design coverage captures the information to ensure that the requirements in the system's requirements specification are addressed in the design. This view shows how well the design reflects the requirements. Note that the presence of a link does not mean that these requirements are correctly designed or implemented. Having a requirements coverage of 100% after the design phase tells management that the system should have all functionality covered in the design as agreed in the contract.

- **Test case coverage**; A comparable situation applies to the requirements coverage in the test specifications. Most test specifications are created in the design phase in parallel with the design. This view shows how well the test specification reflects the requirements. Again this does not mean the functionality is correctly implemented. Having a coverage of 100% tells management that all functionality will be tested in the test phase.

- **Test pass coverage**; A system test is an internal test, often called factory test, to check if the system is working correctly. If all system tests pass, the development team can show the customer that all functionality is implemented and that all functionality is working correctly as agreed. This view shows which requirements are tested and ready for the customer acceptance test.

- **Acceptance coverage**; The customer can approve the results by means of the final acceptance test. This view shows which requirements are accepted by the customer and are ready for release.

- **Evaluation coverage**; After delivery, the evaluation coverage view indicates which requirements have been evaluated and are suitable for reuse in ongoing and future projects.

The above discussed examples of coverage views each reflect a link in the traceability model depicted in Figure 4.1. Of course other examples can be defined such as combinations of these views, e.g., capturing information about the coverage in the test specification and the actual execution of these system tests.

The work products and traceability links that actually need to be captured in the traceability model depend on project-specific information needs [Dömges and Pohl, 1998], but also on factors such as schedule and budget [Ramesh et al., 1995].

Concept Identification

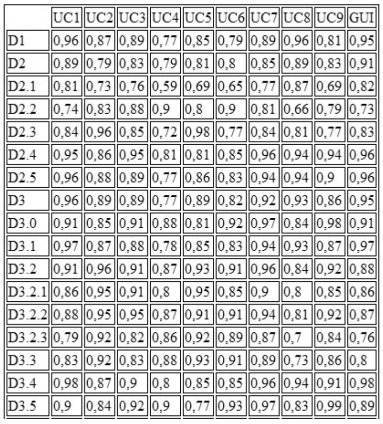
Every concept contained in the traceability model should be uniquely identified in the available documentation. Since the documents are typically semi-structured (typically being just MS Word files), this requires a certain amount of manual processing. The more systematically the documents are organized (for example through the use of templates such as MIL-std 498 or IEEE-1233-1998), the easier it is to make this structure explicit and identify the texts for each of the entities from the traceability model.

In general, identifying individual requirements and test cases in the documentation is relatively easy compared to identifying the design artefacts. Requirements and test cases in most development approaches are tagged with a unique identifier. For design decisions it is often not so clear how they should be documented and identified. Key decisions are often captured in diagrams, e.g., in UML. Here, we encounter the well known problem of establishing traceability relations between requirements and design [Settimi et al., 2004]. Solutions exist to make architectural knowledge more explicit such as capturing architectural assumptions explicitly [Lago and van Vliet, 2005]. Unfortunately these solutions are often not yet used in practice [Graaf et al., 2003].

Text Pre-processing

After defining the entities and the texts belonging to each of them, some pre-processing of these texts is needed. The first step is extracting the texts from the original documents, bringing them in the (plain text) input format suitable for further automated processing. This often is a manual or semi-automatic task. In the semi-automatic case, scripting techniques using, e.g., Perl, can be used to transform the original text into the format needed for further processing. Whether such scripting techniques can be used depends very much on the document structure of the original documentation. The next step is to conduct typical IR steps such as lexical analysis, stop word elimination, stemming, index-term selection, and index construction.

The collection of documents (the "corpus") to be used as input for LSI may be larger than the texts corresponding to the entities from the traceability model. In fact, LSI analysis may benefit from including additional documents containing texts about, for example, the application domain. It allows LSI to collect more terms that are typically used in combination, helping LSI to deal with, for example, synonyms. If such extra documents are used, these documents need to be preprocessed as well.



	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	GUI
D1	0,96	0,87	0,89	0,77	0,85	0,79	0,89	0,96	0,81	0,95
D2	0,89	0,79	0,83	0,79	0,81	0,8	0,85	0,89	0,83	0,91
D2.1	0,81	0,73	0,76	0,59	0,69	0,65	0,77	0,87	0,69	0,82
D2.2	0,74	0,83	0,88	0,9	0,8	0,9	0,81	0,66	0,79	0,73
D2.3	0,84	0,96	0,85	0,72	0,98	0,77	0,84	0,81	0,77	0,83
D2.4	0,95	0,86	0,95	0,81	0,81	0,85	0,96	0,94	0,94	0,96
D2.5	0,96	0,88	0,89	0,77	0,86	0,83	0,94	0,94	0,9	0,96
D3	0,96	0,89	0,89	0,77	0,89	0,82	0,92	0,93	0,86	0,95
D3.0	0,91	0,85	0,91	0,88	0,81	0,92	0,97	0,84	0,98	0,91
D3.1	0,97	0,87	0,88	0,78	0,85	0,83	0,94	0,93	0,87	0,97
D3.2	0,91	0,96	0,91	0,87	0,93	0,91	0,96	0,84	0,92	0,88
D3.2.1	0,86	0,95	0,91	0,8	0,95	0,85	0,9	0,8	0,85	0,86
D3.2.2	0,88	0,95	0,95	0,87	0,91	0,91	0,94	0,81	0,92	0,87
D3.2.3	0,79	0,92	0,82	0,86	0,92	0,89	0,87	0,7	0,84	0,76
D3.3	0,83	0,92	0,83	0,88	0,93	0,91	0,89	0,73	0,86	0,8
D3.4	0,98	0,87	0,9	0,8	0,85	0,85	0,96	0,94	0,91	0,98
D3.5	0,9	0,84	0,92	0,9	0,77	0,93	0,97	0,83	0,99	0,89

Figure 4.2: Example of Similarity Matrix

Link Reconstruction

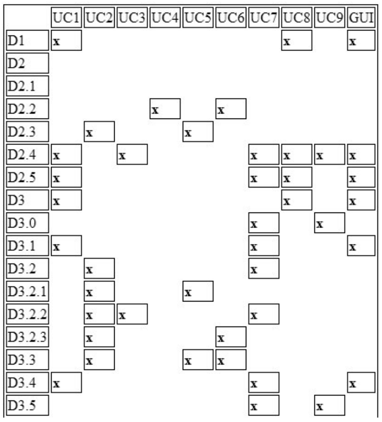
After generating the term-by-document matrix we can reconstruct the traceability links using LSI. First of all, this creates the rank-k model on the basis of which similarities between documents can be determined. Here we need to choose the number for k. Secondly, for every link type in the traceability model (for example tracing requirements to designs) a similarity matrix is created containing the similarities between all elements (for example between every requirement and design artefact).

The result of our LSI analysis is a similarity matrix containing the recovered links, represented as their similarity measures. Figure 4.2 shows an example of a similarity matrix calculated for 10 use cases and 3 design components. This similarity matrix allows us to judge the quality for every recovered link.

Link Selection

Once LSI has created the similarity matrix, a choice has to be made if the similarity number is indeed a traceability link or not. There are several different strategies for doing this: the cut point, cut percentage, constant threshold, and variable threshold strategy [Antoniol et al., 2002; Marcus and Maletic, 2003; De Lucia et al., 2004].

All these strategies have their strengths and shortcomings. In order to benefit from the specific characteristics of applying LSI to traceability reconstruction, we propose two new link selection strategies: a one and



	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	GUI
D1	x							x		x
D2										
D2.1										
D2.2				x		x				
D2.3		x			x					
D2.4	x		x				x	x	x	x
D2.5	x						x	x		x
D3	x							x		x
D3.0							x		x	
D3.1	x						x			x
D3.2		x					x			
D3.2.1		x			x					
D3.2.2		x	x				x			
D3.2.3		x				x				
D3.3		x			x	x				
D3.4	x						x			x
D3.5							x		x	

Figure 4.3: Example of Traceability Matrix

two dimensional vector filter strategy, which we will discuss in detail in Section 4.5.

The result of the link selection step is a traceability matrix containing the links not filtered by the chosen link selection strategy. The link selection strategy and its corresponding parameters determine which similarity measures will become the final traceability links. In Figure 4.3, an example of a reconstructed traceability matrix is shown using our two dimensional vector filter strategy. In Section 4.5, we will explain, using an example, how to construct this traceability matrix with our link selection strategies.

Requirements View Generation

The final step is to use the reconstructed traceability links to obtain requirements views. Currently we have defined a number of different views concerning the status and coverage of requirements, as well as a view to browse the reconstructed traceability links.

For example, given the presence of a link, the status of a requirement can be appropriately set. Moreover, management information can be obtained by computing percentages of requirements that have reached a certain status.

The reconstructed traceability matrix can also be used to calculate coverage metrics. Currently, for every link defined in the traceability model we calculate the percentage of all requirements covered by the specific link. Thus, we get a list of requirements coverage percentages in the design, test cases, and so on. Another view shows the requirements that are not covered by, e.g., the design or test cases. The developers can use this information to check if the requirements are indeed not covered and can undertake appropriate action.

Tackling Requirement Changes

Finally, when we have reconstructed a traceability matrix and are able to generate requirements views, we need to maintain the traceability matrix. In other words, we need to extend our approach so it is able to tackle requirements evolution. We use an incremental approach for handling this.

In step 5 we have a validated traceability matrix. This matrix can be used to generate requirements views. This validated traceability matrix is also the new reference traceability matrix. So, when a requirement changes during a project, steps 3 – 5 are executed again (automatically) and the change can be validated by the user using the reference traceability matrix and the newly generated traceability matrix.

In a new project (started from scratch), the traceability matrix grows as the project grows. The incremental approach causes the validation steps to be small, minimizing the manual effort required. When applying the MAREV approach to a project that is already running, the first full execution of all steps in the MAREV approach requires quite some manual validation effort. However, once the first run of MAREV has been done, the incremental approach only requires little manual effort in the next runs.

4.4.2 Assessment of the Results

In order to assess the suitability of the reconstructed links, we conduct a qualitative as well as a quantitative analysis of the links obtained.

The qualitative assessment of the links is primarily done by experts exploring the documents. The structure of the documents set is of major influence on this process. It helps significantly if the documents are structured according to an (international) standard or template such as IEEE standard 830-1998, IEEE standard 1233-1998, ESA [European Space Agency, 1991] or Volere [Robertson and Robertson, 2000]. Beforehand, such a structure helps choosing the concepts and pre-processing the documents. Afterwards it helps in assessing the reconstructed traceability links as it is easier to browse through the documents. A tool for exploring the links in order to support the qualitative assessment is discussed in Section 4.6.

The quantitative assessment consists of measuring two well-known IR metrics: *recall* and *precision* [Salton and McGill, 1986; Rijsbergen, 1979; Frakes and Baeza-Yates, 1992; Baeza-Yates and Ribeiro-Neto, 1999]:

$$recall = \frac{|correct \cap retrieved|}{|correct|}$$

$$precision = \frac{|correct \cap retrieved|}{|retrieved|}$$

The number of *correct* traceability links are specified in a reference traceability matrix provided by the experts developing the system. The number of *retrieved* traceability links is derived from the LSI analysis.

Both metrics have values between [0, 1]. A recall of 1 means that all correct links were reconstructed, however the total set of links can contain incorrect links. A precision of 1 indicates that all reconstructed links are correct, but there can be correct links that were not reconstructed. The link selection strategy and its corresponding parameters influence the performance indicators recall and precision, as we will see in the case studies.

4.5 Link Selection Strategies

For selecting the relevant links in a similarity matrix several link selection strategies are available. In their application of LSI, De Lucia *et al.* present a number of strategies for selecting traceability links. The following are discussed [De Lucia et al., 2004]:

1. *cut point*; In this strategy we select the top μ links regardless of the actual value of the similarity measure [Antoniol et al., 2002; Marcus and Maletic, 2003]. This strategy always returns exactly μ traceability links.

2. *cut percentage*; In this strategy we select a percentage p of the ranked list to be considered as links regardless of the actual value of the similarity measure. This strategy always returns exactly the $p\%$ of the total reconstructed candidate links.

3. *constant threshold*; In this strategy we select those links that have a similarity measure higher than c, where c is a constant (a commonly used threshold is 0.7). Note that the number of returned links is flexible.

4. *variable threshold*; In this strategy, proposed by De Lucia *et al.*, we select those links that have a similarity measure higher than ε, where ε is calculated through a percentage q of the difference between the

maximum and minimum similarity measures of the total set of similarity measures, e.g., the best $q\%$ of the interval defined by the maximum and minimum [De Lucia et al., 2004]. This strategy is useful if the difference between the maximum and the minimum is low.

5. *scale threshold*; In this strategy, proposed by De Lucia *et al.*, the links are obtained as a percentage of the maximum similarity measures, i.e., $\varepsilon = c * MaxSimilarity$, where $0 \leq c \leq 1$ [Antoniol et al., 2002; De Lucia et al., 2004]. This measure is most useful if the maximum similarity is low.

In our case studies, we have experimented with each of these strategies, which again all have their strengths and shortcomings. Except for strategy constant threshold, all strategies return at least one or more traceability links as correct links, while in our case studies situations exist where no links should be found, e.g., when the quality of the document set is poor. Note however, that it is possible for individual rows or columns to have no links, since the threshold is calculated using the complete set of similarity measures in the matrix.

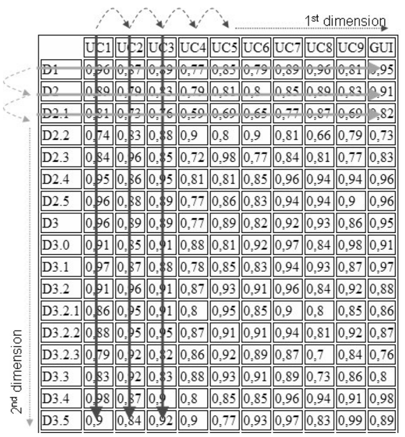
Furthermore, the first two strategies do not take the similarity measure into account and make a selection independent of the calculated result. They simply select the μ best or $p\%$ best similarity measures as traceability links. A typical question is what number should we choose for the μ and p? In most cases, we do not know the exact number of traceability links to return and it is hard to predict this number.

The last two strategies define an interval containing the selection of similarity measures that are correct traceability links. Both strategies are very vulnerable for extremes. For example, if the minimal similarity measure is very low with respect to the other measures, it is possible that the top 20% contains almost all measures.

To deal with these issues, we have experimented with a new approach, that tries to take advantage of the specific characteristics of our setting. For requirements traceability purposes, it is not very likely that there are, e.g., requirements that link to all test cases, or design decisions that may be inspired by all requirements together. For that reason, we propose a strategy that works on a per column basis.

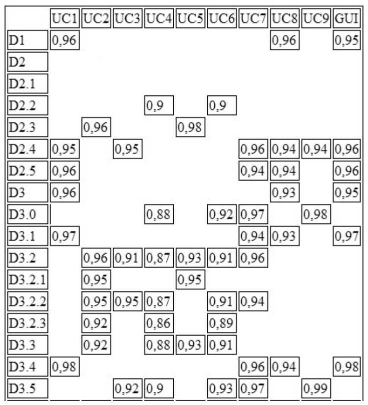
4.5.1 One Dimensional Vector Filter Strategy

This strategy takes into account each column of the similarity matrix separately (see 1st dimension in Figure 4.4a). Each column vector of the similarity matrix is taken as a new set of similarity measures. Then, for each column, it combines the constant and variable threshold approaches: if


1st dimension
2nd dimension

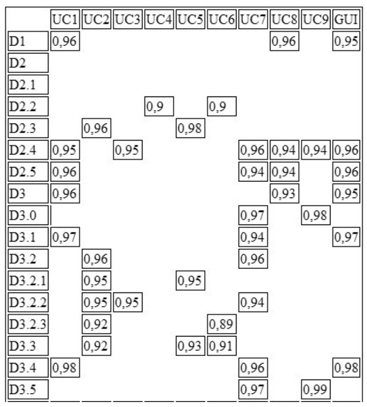
	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	GUI
D1	0,96	0,87	0,89	0,77	0,85	0,79	0,89	0,96	0,81	0,95
D2	0,89	0,79	0,83	0,79	0,81	0,8	0,85	0,89	0,83	0,91
D2.1	0,81	0,73	0,76	0,59	0,69	0,65	0,77	0,87	0,69	0,82
D2.2	0,74	0,83	0,88	0,9	0,8	0,9	0,81	0,66	0,79	0,73
D2.3	0,84	0,96	0,85	0,72	0,98	0,77	0,84	0,81	0,77	0,83
D2.4	0,95	0,86	0,95	0,81	0,81	0,85	0,96	0,94	0,94	0,96
D2.5	0,96	0,88	0,89	0,77	0,86	0,83	0,94	0,94	0,9	0,96
D3	0,96	0,89	0,89	0,77	0,89	0,82	0,92	0,93	0,86	0,95
D3.0	0,91	0,85	0,91	0,88	0,81	0,92	0,97	0,84	0,98	0,91
D3.1	0,97	0,87	0,88	0,78	0,85	0,83	0,94	0,93	0,87	0,97
D3.2	0,91	0,96	0,91	0,87	0,93	0,91	0,96	0,84	0,92	0,88
D3.2.1	0,86	0,95	0,91	0,8	0,95	0,85	0,9	0,8	0,85	0,86
D3.2.2	0,88	0,95	0,95	0,87	0,91	0,91	0,94	0,81	0,92	0,87
D3.2.3	0,79	0,92	0,82	0,86	0,92	0,89	0,87	0,7	0,84	0,76
D3.3	0,83	0,92	0,83	0,88	0,93	0,91	0,89	0,73	0,86	0,8
D3.4	0,98	0,87	0,9	0,8	0,85	0,85	0,96	0,94	0,91	0,98
D3.5	0,9	0,84	0,92	0,9	0,77	0,93	0,97	0,83	0,99	0,89

(a) Reconstructed Similarity Matrix



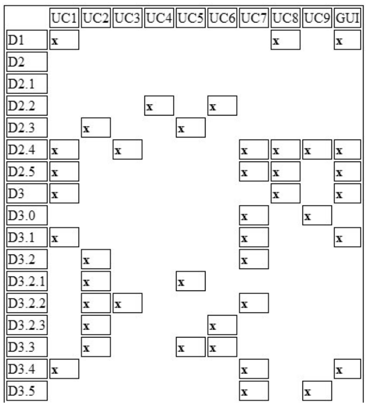
	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	GUI
D1	0,96							0,96		0,95
D2										
D2.1										
D2.2				0,9		0,9				
D2.3		0,96			0,98					
D2.4	0,95		0,95				0,96	0,94	0,94	0,96
D2.5	0,96						0,94	0,94		0,96
D3	0,96							0,93		0,95
D3.0				0,88		0,92	0,97		0,98	
D3.1	0,97						0,94	0,93		0,97
D3.2		0,96	0,91	0,87	0,93	0,91	0,96			
D3.2.1		0,95			0,95					
D3.2.2		0,95	0,95	0,87		0,91	0,94			
D3.2.3		0,92		0,86		0,89				
D3.3		0,92		0,88	0,93	0,91				
D3.4	0,98						0,96	0,94		0,98
D3.5			0,92	0,9		0,93	0,97		0,99	

(b) Applied One Dimensional Vector Filter Strategy



	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	GUI
D1	0,96							0,96		0,95
D2										
D2.1										
D2.2				0,9		0,9				
D2.3		0,96			0,98					
D2.4	0,95		0,95				0,96	0,94	0,94	0,96
D2.5	0,96						0,94	0,94		0,96
D3	0,96							0,93		0,95
D3.0							0,97		0,98	
D3.1	0,97						0,94			0,97
D3.2		0,96					0,96			
D3.2.1		0,95			0,95					
D3.2.2		0,95	0,95				0,94			
D3.2.3		0,92				0,89				
D3.3		0,92			0,93	0,91				
D3.4	0,98						0,96			0,98
D3.5							0,97		0,99	

(c) Applied Two Dimensional Vector Filter Strategy



	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	GUI
D1	x							x		x
D2										
D2.1										
D2.2				x		x				
D2.3		x			x					
D2.4	x		x				x	x	x	x
D2.5	x						x	x		x
D3	x							x		x
D3.0							x		x	
D3.1	x						x			x
D3.2		x					x			
D3.2.1		x			x					
D3.2.2		x	x				x			
D3.2.3		x				x				
D3.3		x			x	x				
D3.4	x						x			x
D3.5							x		x	

(d) The Final Reconstructed Traceability Matrix

Figure 4.4: Applying the One and Two Dimensional Vector Filter on the example Similarity Matrix using $c = 0.7$ and $q = 20\%$

there are measures above the constant threshold c, we take the best $q\%$, e.g., 20% of the similarity measures in that column.

The constant threshold is used to indicate if there is any similarity between this specific work product (in the example a use case) and the other work products (in the example the design artefacts)(see Figure 4.4a). If all similarity measures in the column vector are smaller than c, there is not enough similarity between the work products and thus there are no traceability links at all. This way we can guarantee a certain level of quality for our reconstructed traceability links.

If there are measures greater than the constant threshold we take the variable threshold for selecting the traceability links. With the variable threshold, a similarity interval is defined by the minimum and maximum similarity measures of the column vector (taking the original column vector, including the possible measures smaller than c). We use q to calculate the variable threshold ε per column. This threshold ε retains the best $q\%$ of the similarity measures in that vector representation and selects them as traceability links independent of the other column vectors.

In the example, depicted in Figure 4.2 and Figure 4.4a, we can see that the constant threshold ($c = 0.7$) has no influence on the result. All the similarity measures and higher than $c = 0.7$. Consequently, every use case in this example has at least one link (in this case, the variable threshold always returns a link per column).

The variable threshold ε is calculated per column, and differs for each column; UC1 $\Rightarrow \varepsilon = 0.916$, UC2 $\Rightarrow \varepsilon = 0.914$, UC3 $\Rightarrow \varepsilon = 0.912$, UC4 $\Rightarrow \varepsilon = 0.838$,UC5 $\Rightarrow \varepsilon = 0.922$, etc... We see that the relatively high variable threshold of UC5 would cause UC4 not to return any links, and that the relative low variable threshold of UC4 would cause that only 2 links of UC2 are filtered (see Figure 4.4a and Figure 4.4b). Besides that, every column can have a different number of returned links. Note that the standard variable threshold strategy would use a single ε for all columns.

With respect to applying only the variable threshold strategy, we will see in our case studies that our strategy increases the precision of the result without affecting the recall. The variable threshold ε in case of taking $q = 20\%$ results in $\varepsilon = 0.91$ for all columns. Consider, as an example, the effect of this threshold on UC4. With this threshold, UC4 would have no links, while for the given example we know there should be two links returned. This decreases the recall with respect to using our one dimensional vector filter strategy. On the other hand, our strategy does filter more of the relative high similarity measures of UC9. Our strategy with a $\varepsilon = 0.930$ for UC9 returns only three links, while with the “normal” variable threshold ($\varepsilon = 0.91$) it returned five links. As we will see in the case studies, the correct link is indeed in that set of three links.

The benefits of our one dimensional vector filter strategy, compared to the strategies discussed by De Lucia *et al.* [De Lucia et al., 2004], are the following:

- Our strategy is flexible in the number of returned candidate traceability links. Thus, it does not always return an absolute number of links, like the cut point and cut percentage strategies.

- Our strategy takes into account the calculated similarity measures and uses a constant threshold to guarantee a certain level of quality. It is possible that with our strategy no traceability links are returned.

- Our strategy is less vulnerable for extremes in the total set of similarity measures. It only takes a subset (the column vector) to set the variable threshold. For each individual work product, it returns a more precise set of traceability links that is not affected by the similarity measures in the other column vectors.

We have shown some arguments why our strategy improves the link selection step compared to the other available strategies. However, there are still two problems with this strategy: 1) it does not consider the other dimension (row) of the similarity matrix (in our case example the design vectors) and 2) it always returns a link for each column if the constant threshold is too low.

The first is a problem because of the following. Imagine the situation that a design vector has relatively high values for the similarity measures compared to the other design vectors in the matrix, e.g., D3.2 compared to D2.2. In this case, this design artefact returns many traceability links using our one dimensional vector filter strategy; the similarity measures are higher than the constant threshold c and also belong to the interval defined by ε of each column. This is an undesirable situation as one design artefact (or one test case) should not cover all (or most of the) use cases.

The second is a problem because it should be possible for columns to return no links. For example, when a use case is not yet covered in the design, the column of that use case should not return any links. Both problems are solved using our second strategy, which is an extension to the one dimensional vector filter strategy.

4.5.2 Two Dimensional Vector Filter Strategy

This two dimensional vector filter strategy is basically the same as our one dimensional vector filter strategy except that it is executed on both dimensions of the similarity matrix (see Figure 4.4a). It also filters the relatively weak similarity measures of the row (in our example the design vectors). In general, this should improve the quality of the reconstructed traceability links; the precision further increases.

When applying our two dimensional vector filter strategy in our example we see that for D3.2, four extra links are filtered. The same results can be observed for, e.g., D3.2.2 and D3.5 (see Figure 4.4c).

However, with this second strategy the risk increases that the filter is too precise and also eliminates correct links, thus decreasing recall. If we look again at UC4, we see there only remains one link after applying our two dimensional vector filter strategy. After applying the two dimensional vector filter strategy, we transform the remaining similarity measures to traceability links. Finally, these form the traceability matrix depicted in Figure 4.4d.

The additional benefits of our two dimensional vector filter strategy with respect to the benefits discussed in Section 4.5.1 are the following:

- It returns a more precise result for each pair of work products (in our example; use case - design artefact pairs).
- It possible that a column returns no links even if the constant threshold has no influence on the result. The second filter dimension (per row) makes this possible.

4.6 The ReqAnalyst Tool Suite

In order to support the traceability reconstruction approach, we developed a tool called ReqAnalyst [1]. The objectives of this tool are:

- To offer a test bed for experimenting with different traceability reconstruction approaches and algorithms;
- To support the application of these approaches to industrial projects.

The tool has been implemented in Java, and follows the Extract-Query-View approach adopted by many reverse engineering tools [van Deursen and Moonen, 2006]. In this approach we first extract the relevant data from the provided documents. This data, the work products and if available the reference traceability matrices, are stored in a database. For reconstructing the traceability links, queries can be done on the database. The reconstructed information combined with the data from the database is used to generate the requirements views.

[1]This tool suite replaces the tool support (TMG toolbox, Trace Reconstructor and Trace Explorer) used in our paper at CSMR 2006 [Lormans and van Deursen, 2006]. The tool is available from http://swerl.tudelft.nl/bin/view/Main/\reqanalyst.

4.6.1 Tool Requirements

In order to make REQANALYST useful in practice, it needs to fulfil the following requirements. One of the key requirements for REQANALYST is that it should reduce the effort for maintaining consistent traceability support and reduce the search time for changes, improve impact analysis and coverage analysis. Besides that, REQANALYST should be able to easily support different development environments and different domains with a minimum of tailoring effort. This includes environments such as global distributed software development, offshoring and outsourcing. Also the deployment of REQANALYST should be simple for such heterogeneous environments.

The input for REQANALYST consists of the work products that need to be traced and of which the requirements views should be generated. The tool should be flexible in the structure of these work products, minimizing the amount of tailoring required to offer a document as input to REQANALYST. In addition to that, it should be able to cluster the work products in an easy and flexible way.

Furthermore, REQANALYST should be scalable. It should be able to handle a large number of work products, but it should also be easily expandable with respect to the number of predefined requirements views (or other views, if necessary).

Since we anticipate that the maintenance of such a traceability matrix cannot be fully automated, REQANALYST should support manual traceability identification as well. In particular, it should be possible to read in a hand-written matrix, to compare the manual with the automatically obtained results, and to easily inspect the documents for which the two matrices differ.

In order to support the evaluation of reconstruction approaches, the latter comparison feature can be used for conducting a qualitative analysis of the reconstruction results. In order to support a quantitative analysis as well, the tools should be able to compute precision and recall figures from the traceability matrices.

4.6.2 Technology Used

REQANALYST is implemented using standard web-technology. For storing the data we use a MySQL[1] database. On top of the database we have implemented a Java web application using Java Servlets (for collecting data and link reconstruction) and Java Server Pages (for presenting the results). The choice for building a dynamic web application in Java made it easy to

[1]http://www.mysql.com

fulfil a number of the practical tool requirements mentioned in the previous subsection, such as ease of deployment.[2] Furthermore, the use of a browser provides access to the tool from any location, making it suitable for global distributed software development.

4.6.3 Functionality and Implementation

The functionality of the present version of REQANALYST is still relatively simple. REQANALYST currently is primarily a research prototype, allowing us to experiment with the use of LSI for requirements coverage view reconstruction.

A REQANALYST session starts by choosing a project, which can be a new one, or one that has been stored in the database already. Once the user has chosen a project, REQANALYST shows a menu with the steps that can be executed next.

REQANALYST first of all offers a menu to extract the data from the provided documentation. The work products and the reference traceability matrices can be extracted. Secondly, it provides a menu for setting the parameters of the LSI reconstruction and the choice for a link selection strategy.

Once the tool has executed a reconstruction an intermediate menu appears showing the reconstructed traceability matrix and some options for generating various requirements views. These views should make it possible to obtain continuous feedback on the progress of ongoing software development or maintenance projects. Furthermore, they facilitate communication between project stakeholders and different document owners. In addition to that, REQANALYST offers views that support the comparison of traceability matrices obtained in different ways, for example manual versus automatically via LSI. Examples are shown in Figures 4.7 and 4.8 discussed in the next section.

4.7 Case Studies

We have conducted three case studies where we applied our approach for reconstructing requirements traceability using LSI. The case studies vary in size and context. The first case study, Pacman, is a small case we developed within our university. This case study gives us the opportunity to explore all the possibilities of the techniques in a controlled environment. We varied the different parameters of our analysis to come to a setting giving the best results. The second case study, called Calisto, is somewhat

[2]For our case studies we used the Apache Tomcat 5.5 web server for deployment

Case Studies	Pacman 2.2	Calisto	Philips
Number of Requirements (work products)	14	12	7
Number of Design Artefacts	24	48	16
Number of Test Cases	20	79	326
Total number of indexed terms	1366	2251	2502
Number of “requirement - design artefact” links	28	59	nk
Number of “requirement - test case” links	19	80	nk

Table 4.1: Case Study Statistics

bigger. Although developed within a university, the system at hand was developed for an external (industrial) client. The last case study involves an industrial project carried out at Philips Applied Technologies. This case study represents a real life project for commercial purposes.

In our case studies, we will focus mainly on two types of traceability links; links between requirements and design, and links between requirements and test. The corresponding traceability model is shown in Figure 4.5. By combining these two link types we can furthermore obtain traceability from design decisions to system tests, as indicated by the dashed line in the figure.

An impression of the size of the cases is provided by Table 4.1. It shows the number of work products involved relevant to our traceability model for each case, as well as the number of indexed terms for the total set of documents, including additional context (e.g. Javadoc). Besides that, it shows the number of links between the different work products as set in the provided reference traceability matrices[1].

For each case study, we will conduct link reconstruction using the following link selection strategies: constant threshold, variable threshold, one dimensional vector filter and two dimensional vector filter, and reflect on the lessons learned from this case.

4.7.1 Case Study I: Pacman 2.2

Our first results are obtained from a lab experiment executed at Delft University of Technology. The system at hand is a simple version of the well-known Pacman game that is used by students in a lab course for testing ob-

[1]For the Philips case study, we do not have the reference traceability matrices. So we do not know the number of links and cannot calculate the link density (nk – not known).

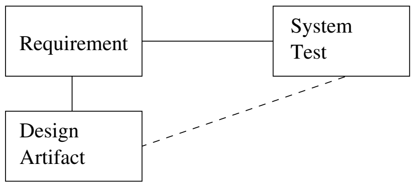


Figure 4.5: Traceability Model for our Case Studies

ject oriented software following Binder's testing approach [Binder, 2000]. An initial implementation for the system is given, and students are expected to extend the test suite according to Binder's test design patterns, and enhance the system with additional features (which they should test as well).

Case Configuration

The available documentation for Pacman consists of

- A requirements specification including a concise domain analysis, ten use cases, and a description of the user interface.
- A design document listing the design decisions at architectural as well as detailed design level. This covers the (model-view-controller) layering used as reflected in the package structure, the static view explaining the main classes and their associations, a dynamic view summarizing the system's main state machine, and a description of the implementation of the user interface.
- A testing document explaining the acceptance test suite for the application. For each use case one or more test cases are provided as well as a test case for validating the proper working of the user interface.

Pacman is shipped with a traceability matrix. As can be seen from the above description, Pacman's documentation has been organized with traceability in mind. Thus, for the acceptance test suite, there is a natural mapping from test case to use case. For the design it is somewhat harder to setup the documentation with clear traceability objectives. As an example, to what requirements should the decision to opt for a model-view-controller architecture be linked?

For the requirements specification the use cases are chosen as main requirement entities. Besides the use cases we also included the domain

UC7	Suspend
Actor	player
1.	Entry condition: The player is alive and playing
2.	The player presses the quit button in order to suspend playing the game
3.	During suspension, no moves are possible, neither from the player nor from the monsters
3a.	The user can press undo in order to undo monster and player moves
4.	Pressing the start button re-activates the game

Figure 4.6: Full text for use case UC7 of the Pacman case study

analysis, user interface, and the requirements for the development environment. The design is listed according to its design decisions, which we used as design entities in our analysis. Finally, every test case is considered as a test case entity for our analysis. In total there are 14 requirement entities, 24 design artefacts, and 20 test cases. In Figure 4.6 we show an example of a use case description. The documents were provided in plain text, and the traceability matrix as an MS Excel spreadsheet. They could be directly passed as input to ReqAnalyst.

As corpus, the collection of all documents was used, including the Javadoc of the implementation. This resulted in a corpus of almost 1366 terms. Furthermore, for c we took the value 0.7. The other two values k and q we varied to get an impression of the impact of these values.

Case Results

The recall (R) and precision (P) for this case study are shown in Table 4.2 for the constant threshold, variable threshold, one dimensional vector filter and two dimensional vector filter strategies discussed in Section 4.5. For the two dimensional vector filter strategy we also recorded the link density (LD). In Figure 4.7 and Figure 4.8 the reconstructed traceability matrices of the Pacman case study are shown using the various filter strategies. Figure 4.7 shows the reconstructed matrices of the links between the requirements and the design, and Figure 4.8 shows the reconstructed matrices of the links between the requirements and the test cases.

The reconstructed matrices are compared with the provided reference traceability matrix. The correctly reconstructed links are coloured grey and each the cell contains an “X”. The empty cells are correctly *not* reconstructed links. According to the reference traceability matrix these cells should not return a link.

Link Type			Constant Threshold		Variable Threshold		One Dim. Vector Filter		Two Dim. Vector Filter	
	c	q	R	P	R	P	R	P	R	P
Use case to design	0.7	10%	1.0	0.09	0.36	0.25	0.29	0.13	0.21	0.15
	0.7	20%	1.0	0.09	0.54	0.15	0.64	0.17	0.46	0.17
	0.7	30%	1.0	0.09	0.82	0.13	0.82	0.16	0.71	0.17
	0.7	40%	1.0	0.09	0.93	0.11	0.89	0.13	0.82	0.14
	0.7	50%	1.0	0.09	0.93	0.09	0.93	0.11	0.86	0.12
	0.7	60%	1.0	0.09	1.0	0.09	0.97	0.10	0.93	0.10
Use case to test	0.7	10%	1.0	0.07	0.42	0.36	0.58	0.27	0.53	0.42
	0.7	20%	1.0	0.07	0.74	0.24	0.68	0.19	0.68	0.27
	0.7	30%	1.0	0.07	0.95	0.17	0.84	0.18	0.79	0.21
	0.7	40%	1.0	0.07	0.95	0.13	0.95	0.14	0.95	0.16
	0.7	50%	1.0	0.07	0.95	0.10	0.95	0.11	0.95	0.13
	0.7	60%	1.0	0.07	1.0	0.09	0.95	0.10	0.95	0.11

Table 4.2: Recall and precision for the reconstructed traceability matrices of Pacman 2.2 with rank-k subspace of 20% and $c = 0.7$

The cells that are coloured light grey are invalid compared to the provided reference traceability matrices. These cells containing “fp” are the false positives. These links should not be reconstructed as traceability links and are therefore incorrectly reconstructed. The dark grey cells containing “fn” are the false negatives (missing links). A link should have been reconstructed between these two particular work products, but our approach did not select this candidate link as a traceability link. In REQANALYST, each cell in these matrices is clickable leading to the text of both documents. This makes it easy to analyze why a certain reconstructed link was present or absent.

Results “Requirements – Design”

The results in Table 4.2 show a relatively low precision of the links between the requirements and design. This is caused by the many false positives. The constant threshold strategy returns the most false positives (and this way returns the lowest precision). The threshold of $c = 0.7$ has almost no influence on the result (see Figure 4.7a). Most similarity measures are above 0.7.

If we apply the variable threshold strategy we filter many of the false positives. This strategy generally increases the precision, but decreases the recall, e.g., for $q = 30\%$. Figure 4.7b shows that 5 correct links are filtered using these settings. We can also see that many of the false positives are located in specific rows and columns. In the case of $q = 30\%$, design artefacts D0, D3.3 and D3.7 and requirement artefacts DA, UC7 and GUI return many false positive.

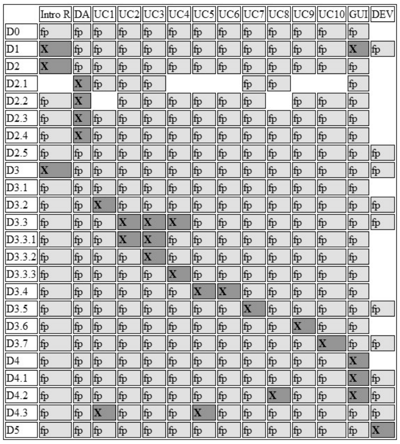
Our one dimensional vector filter strategy filters many of these false positives in the columns of the traceability matrix. For example for the column with label DA (Domain Analysis) it filters an additional 8 false positives compared to the variable threshold strategy (see Figure 4.7b and Figure 4.7c). The same can be observed for UC7 (4 additional false positives) and GUI (6 additional false positives). In this case, with $q = 30\%$ the filter increases the precision and does not influence the recall (see Table 4.2 with $q = 30\%$). However, the filter has limited influence on the rows containing many false positives such as D0, D3.3 and D3.7.

Using our two dimensional vector filter strategy also affects the rows of the matrix. Compared with the one dimensional vector filter strategy we filter an additional 1 false positive for D0, 3 false positives for D3.3, and 4 false positives for D3.7. In this case we did increase the precision a little, but also decreased the recall; 3 correct links are now filtered (dark grey cells containing “fn”).

The two dimensional vector filter strategy did also filter one additional false positive in UC7. Still UC7 contains many false positives. The quantitative analysis did not help us to understand this phenomenon so we needed to explore the text. We used the “browse results view” of REQ-ANALYST for this. We investigated the text of the correct links and the returned links with the best score. We manipulated the text to improve our understanding of these links. Improving the similarity measure of the correct links was not that difficult, but understanding why the other links had such a high similarity score was not always that obvious.

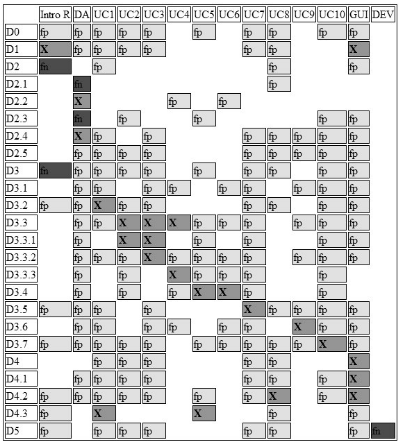
To improve the correct similarity measure of UC7 (See Figure 4.6) the state conditions were made more explicit in the design text. So documenting that a state has changed, e.g., to “playing state” again, is not sufficient. Explicitly documenting that the player is “alive and playing” helps to link the design artefact to the use case.

Furthermore, in the design artefact the term “pause” was used for indicating a suspension. So we also introduce the term “pause” in the use case description. The last step of the use case description was changed in: “4. Pressing the start button ends the pause, and re-activates the game”. These changes in the text increased the similarity measure of the correct link. However, this did not influence the total result of use case UC7. UC7 still returned 12 false positives. The other similarity measures did not sufficiently decrease for the link selection strategy to filtered them.



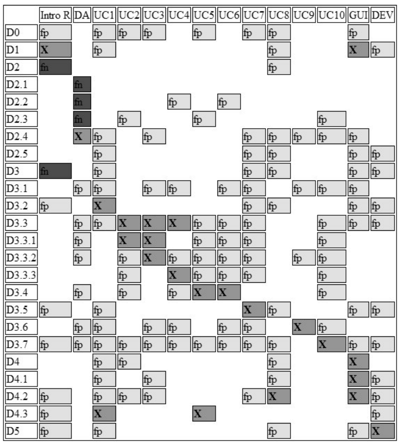
	Intro R	DA	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	GUI	DEV
D0	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	
D1	X	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	X	fp
D2	X	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	
D2.1		X	fp	fp	fp				fp	fp			fp	
D2.2	fp	X		fp	fp	fp	fp	fp	fp		fp	fp	fp	
D2.3	fp	X	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	
D2.4	fp	X	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	
D2.5	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp
D3	X	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp
D3.1	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	
D3.2	fp	fp	X	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp
D3.3	fp	fp	fp	X	X	X	fp	fp	fp	fp	fp	fp	fp	fp
D3.3.1	fp	fp	fp	X	X	fp	fp	fp	fp	fp	fp	fp	fp	
D3.3.2	fp	fp	fp	fp	X	fp	fp	fp	fp	fp	fp	fp	fp	
D3.3.3	fp	fp	fp	fp	fp	X	fp	fp	fp	fp	fp	fp	fp	
D3.4	fp	fp	fp	fp	fp	fp	X	X	fp	fp	fp	fp	fp	
D3.5	fp	fp	fp	fp	fp	fp	fp	fp	X	fp	fp	fp	fp	fp
D3.6	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	X	fp	fp	
D3.7	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	X	fp	fp
D4	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	X	
D4.1	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	X	fp
D4.2	fp	fp	fp	fp	fp	fp	fp	fp	fp	X	fp	fp	X	fp
D4.3	fp	fp	X	fp	fp	fp	X	fp	fp	fp	fp	fp	fp	fp
D5	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	X

(a) Constant Threshold Strategy



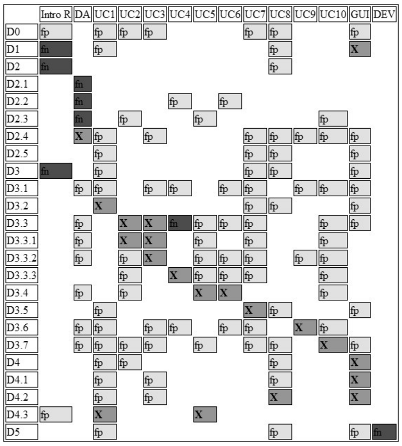
	Intro R	DA	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	GUI	DEV
D0	fp	fp	fp	fp	fp		fp		fp	fp		fp	fp	
D1	X	fp	fp	fp	fp				fp	fp			X	
D2	fn		fp							fp			fp	
D2.1		fn								fp				
D2.2		X				fp		fp						
D2.3		fn		fp			fp					fp	fp	
D2.4		X	fp		fp				fp	fp	fp	fp	fp	
D2.5		fp	fp	fp	fp				fp	fp	fp	fp	fp	
D3	fn	fp	fp	fp	fp		fp		fp	fp		fp	fp	
D3.1		fp	fp		fp	fp		fp	fp		fp	fp	fp	
D3.2	fp	fp	X	fp	fp				fp	fp		fp	fp	
D3.3		fp	fp	X	X	X	fp	fp	fp		fp	fp	fp	
D3.3.1		fp		X	X		fp		fp			fp	fp	
D3.3.2		fp	fp	fp	X	fp	fp	fp	fp		fp	fp	fp	
D3.3.3		fp		fp		X	fp	fp	fp			fp		
D3.4		fp		fp		fp	X	X	fp			fp		
D3.5	fp	fp	fp		fp				X	fp	fp	fp	fp	
D3.6		fp	fp		fp	fp		fp	fp		X	fp	fp	
D3.7	fp	fp	fp	fp	fp		fp	fp	fp	fp	fp	X	fp	
D4			fp	fp	fp				fp	fp			X	
D4.1		fp	fp	fp	fp				fp	fp		fp	X	
D4.2	fp	fp	fp	fp	fp		fp		fp	X		fp	X	
D4.3	fp		X				X			fp			fp	
D5	fp		fp	fp	fp				fp	fp			fp	fn

(b) Variable Threshold Strategy



	Intro R	DA	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	GUI	DEV
D0	fp		fp	fp	fp		fp		fp	fp			fp	
D1	X		fp							fp			X	fp
D2	fn									fp				
D2.1		fn												
D2.2		fn				fp		fp						
D2.3		fn		fp			fp					fp		
D2.4		X	fp		fp				fp	fp	fp	fp	fp	
D2.5			fp						fp	fp			fp	fp
D3	fn		fp						fp	fp			fp	fp
D3.1		fp	fp		fp	fp		fp	fp		fp	fp	fp	
D3.2	fp		X						fp	fp			fp	fp
D3.3		fp	fp	X	X	X	fp	fp	fp			fp	fp	fp
D3.3.1		fp		X	X		fp	fp	fp			fp		
D3.3.2		fp		fp	X	fp	fp	fp	fp		fp	fp		
D3.3.3				fp		X	fp	fp	fp			fp		
D3.4		fp		fp		fp	X	X				fp		
D3.5	fp		fp						X	fp			fp	fp
D3.6		fp	fp		fp	fp		fp	fp		X	fp		
D3.7	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp		X	fp	fp
D4			fp	fp						fp			X	
D4.1			fp		fp					fp			X	fp
D4.2	fp		fp	fp	fp				fp	X			X	fp
D4.3	fp		X				X							fp
D5	fp		fp							fp			fp	X

(c) One Dimensional Vector Filter Strategy



	Intro R	DA	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	GUI	DEV
D0	fp		fp	fp	fp				fp	fp			fp	
D1	fn		fp							fp			X	
D2	fn									fp				
D2.1		fn												
D2.2		fn				fp		fp						
D2.3		fn		fp			fp					fp		
D2.4		X	fp		fp				fp	fp	fp	fp	fp	
D2.5			fp						fp	fp			fp	
D3	fn		fp						fp	fp			fp	
D3.1		fp	fp		fp	fp		fp	fp		fp	fp	fp	
D3.2			X						fp	fp			fp	
D3.3		fp		X	X	fn	fp	fp	fp			fp	fp	
D3.3.1		fp		X	X		fp		fp			fp		
D3.3.2		fp		fp	X		fp	fp	fp		fp	fp		
D3.3.3				fp		X	fp	fp	fp			fp		
D3.4		fp		fp			X	X				fp		
D3.5			fp						X	fp			fp	
D3.6		fp	fp		fp	fp		fp	fp		X	fp		
D3.7		fp	fp	fp	fp		fp		fp	fp		X	fp	
D4			fp	fp						fp			X	
D4.1			fp		fp					fp			X	
D4.2			fp		fp					X			X	
D4.3	fp		X				X							
D5			fp							fp			fp	fn

(d) Two Dimensional Vector Filter Strategy

Figure 4.7: Reconstructed traceability matrices between requirements and design using different link selection strategies with rank-k subspace of 20%, $c = 0.7$ and $q = 30\%$

Results “Requirements – Test”

Looking at the links between the requirements and test cases we observed similar results. The constant threshold strategy does not have much influence and only filters a small number of candidate links resulting in many false positives (see Figure 4.8a).

The variable threshold strategy does a much better job and filters many of the false positives. Again we can see that a number of rows and columns cause the many false positives. In this case the rows Intro TC, TC6 and TC11, and the columns DA, UC10 and GUI (see Figure 4.8b).

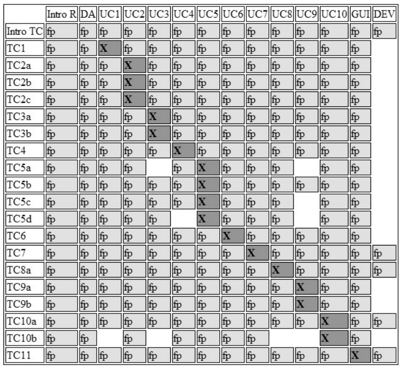
Our one dimensional vector filter strategy again filters many additional false positives, but in this case it also filters some correct links causing the recall to decrease (see Table 4.2 with $q = 30\%$). Two correct links are filtered (see Figure 4.8b and Figure 4.8c). For the variable threshold strategy the threshold $\varepsilon = 0.88$. For the column UC5 the threshold $\varepsilon = 0.91$, and for column UC10 the threshold $\varepsilon = 0.92$. So, for both UC5 and UC10 the threshold is higher filtering more cells in that column. Cell $\langle UC5, TC5a\rangle$ has a similarity of 0.9 (< 0.91) and because of that it will be (incorrectly) filtered using the one dimensional vector filter strategy. The same holds for cell $\langle UC10, TC10a\rangle$, which has a similarity of 0.91 and the threshold for that column is 0.92.

The two dimensional vector filter strategy shows the expected result. It filters some additional false positives in the rows of the matrix increasing the precision. Unfortunately, again one additional correct link is filtered (see Figure 4.8c and Figure 4.8d).

Lessons Learned

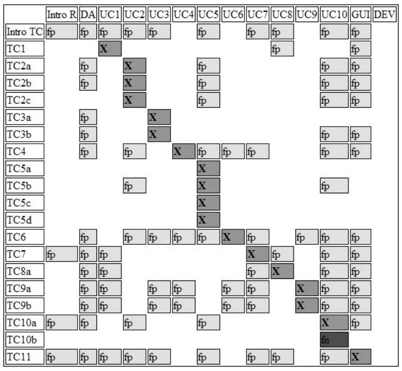
The key lessons learned from this case study are:

- Reconstructing traceability between use cases and test cases is easier than between use cases and design.
- The design activity and traceability activity is a hard combination. The designer should structure the design decisions so that clear traceability can be established.
- For larger case studies we do not expect results to become better than for Pacman. Pacman is designed to incorporate traceability and for most industrial projects this only limitedly done [Graaf et al., 2003; Gotel and Finkelstein, 1994].
- Eliminating false positives in columns with many hits is effectively done by the one dimensional vector filter strategy.



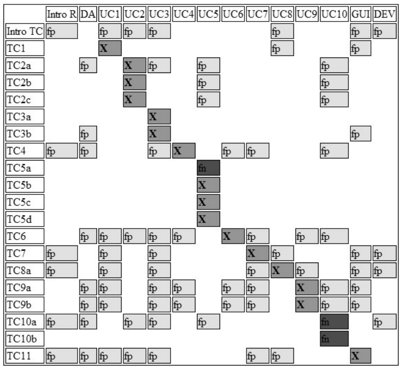
	Intro R	DA	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	GUI	DEV
Intro TC	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp
TC1	fp	fp	X	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	
TC2a	fp	fp	fp	X	fp	fp	fp	fp	fp	fp	fp	fp	fp	
TC2b	fp	fp	fp	X	fp	fp	fp	fp	fp	fp	fp	fp	fp	
TC2c	fp	fp	fp	X	fp	fp	fp	fp	fp	fp	fp	fp	fp	
TC3a	fp	fp	fp	fp	X	fp	fp	fp	fp	fp	fp	fp	fp	
TC3b	fp	fp	fp	fp	X	fp	fp	fp	fp	fp	fp	fp	fp	
TC4	fp	fp	fp	fp	fp	X	fp	fp	fp	fp	fp	fp	fp	
TC5a	fp	fp	fp	fp		fp	X	fp	fp	fp		fp	fp	
TC5b	fp	fp	fp	fp	fp	fp	X	fp	fp	fp	fp	fp	fp	
TC5c	fp	fp	fp	fp	fp	fp	X	fp	fp	fp		fp	fp	
TC5d	fp	fp	fp	fp	fp		X	fp	fp	fp		fp	fp	
TC6	fp	fp	fp	fp	fp	fp	fp	X	fp	fp	fp	fp	fp	
TC7	fp	fp	fp	fp	fp	fp	fp	fp	X	fp	fp	fp	fp	fp
TC8a	fp	fp	fp	fp	fp	fp	fp	fp	fp	X	fp	fp	fp	fp
TC9a	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	X	fp	fp	
TC9b	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	X	fp	fp	
TC10a	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	X	fp	fp
TC10b	fp	fp		fp		fp	fp	fp	fp			X	fp	
TC11	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	fp	X	fp

(a) Constant Threshold Strategy



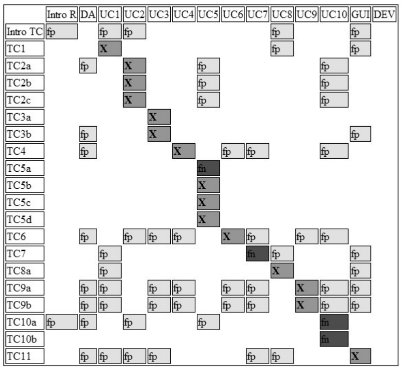
	Intro R	DA	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	GUI	DEV
Intro TC	fp	fp	fp	fp	fp		fp		fp	fp		fp	fp	
TC1			X							fp			fp	
TC2a		fp		X			fp					fp	fp	
TC2b		fp		X			fp					fp	fp	
TC2c				X			fp					fp	fp	
TC3a		fp			X									
TC3b		fp			X							fp	fp	
TC4		fp		fp		X	fp	fp	fp			fp	fp	
TC5a							X							
TC5b				fp			X					fp		
TC5c							X							
TC5d							X							
TC6		fp		fp	fp	fp	fp	X	fp		fp	fp	fp	
TC7	fp	fp	fp						X	fp		fp	fp	
TC8a		fp	fp						fp	X		fp	fp	
TC9a		fp	fp		fp	fp		fp	fp		X	fp	fp	
TC9b		fp	fp		fp	fp		fp	fp		X	fp	fp	
TC10a	fp	fp		fp			fp					X	fp	
TC10b												fn		
TC11	fp	fp	fp	fp	fp		fp		fp	fp		fp	X	

(b) Variable Threshold Strategy



	Intro R	DA	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	GUI	DEV
Intro TC	fp		fp	fp	fp					fp			fp	fp
TC1			X							fp			fp	
TC2a		fp		X	fp		fp					fp		
TC2b				X			fp					fp		
TC2c				X			fp					fp		
TC3a					X									
TC3b		fp			X								fp	
TC4	fp	fp			fp	X		fp	fp			fp		
TC5a							fn							
TC5b							X							
TC5c							X							
TC5d							X							
TC6		fp	fp	fp	fp	fp		X	fp		fp	fp		
TC7	fp		fp		fp				X	fp			fp	fp
TC8a	fp		fp		fp				fp	X	fp		fp	fp
TC9a		fp	fp		fp	fp		fp	fp		X	fp	fp	
TC9b		fp	fp		fp	fp		fp	fp		X	fp	fp	
TC10a	fp	fp		fp	fp		fp					fn		fp
TC10b												fn		
TC11	fp	fp	fp	fp	fp				fp	fp			X	

(c) One Dimensional Vector Filter Strategy



	Intro R	DA	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	GUI	DEV
Intro TC	fp		fp	fp						fp			fp	
TC1			X							fp			fp	
TC2a		fp		X			fp					fp		
TC2b				X			fp					fp		
TC2c				X			fp					fp		
TC3a					X									
TC3b		fp			X								fp	
TC4		fp				X		fp	fp			fp		
TC5a							fn							
TC5b							X							
TC5c							X							
TC5d							X							
TC6		fp		fp	fp	fp		X	fp		fp	fp		
TC7			fp						fn	fp			fp	
TC8a			fp							X			fp	
TC9a		fp	fp		fp	fp		fp	fp		X	fp	fp	
TC9b		fp	fp		fp	fp		fp	fp		X	fp	fp	
TC10a	fp	fp		fp			fp					fn		
TC10b												fn		
TC11		fp	fp	fp	fp				fp	fp			X	

(d) Two Dimensional Vector Filter Strategy

Figure 4.8: Reconstructed traceability matrices between requirements and test cases using different link selection strategies with rank-k subspace of 20%, $c = 0.7$ and $q = 30\%$

- Eliminating false positives in columns, as well as rows with many hits is effectively done by the two dimensional vector filter strategy.

4.7.2 Case Study II: Calisto

In this section we discuss our results from the second case study. This case study involves software developed by students from Eindhoven University of Technology in a software engineering project where the students needed to carry out a complete development life-cycle.

In this project an Interface Specification Tool is constructed. This tool is designed to support the ISpec approach, a specification method in the context of component technology [Jonkers, 2000]. The purpose of the tool is to create Interface Specification Documents as well as exporting these documents to other software components.

Case Configuration

The provided documentation for Calisto consists of:

- A user requirements specification (URD), which states what the product is supposed to do according to the client. It is used as the contract between the client and the developers of the product.

- A software requirements specification (SRS), which formally describes the functionality of the product to be made. The document translates all user requirements into software requirements. It defines a logical model that contains the functionality that was found in the user requirements. The functional requirements are described using the classes defined in the logical model including attribute and method descriptions.

- An acceptance test plan (ATP), which describes the plan for testing the developed software tool against the user requirements. It lists the test cases that should cover the user requirements.

The documents all comply with the equally named specifications from the Software Engineering Standard, as set by the European Space Agency (ESA) [European Space Agency, 1991]. We consider the SRS as a design document as it specifies classes and interfaces. Thus, in our analysis we will refer to the software requirements as our design artefacts.

All requirements have a unique identifier; the user requirements comply to the prefix URCARxx and the software requirements to the prefix SRFURxx were xx is a unique number. The test cases are directly related

Link Type			Constant Threshold		Variable Threshold		One Dim. Vector Filter		Two Dim. Vector Filter	
	c	q	R	P	R	P	R	P	R	P
Req. to design	0.4	10%	0.54	0.12	0.07	1.0	0.19	0.39	0.15	0.69
	0.4	20%	0.54	0.12	0.10	0.60	0.27	0.28	0.22	0.43
	0.4	30%	0.54	0.12	0.15	0.35	0.41	0.21	0.31	0.29
	0.4	40%	0.54	0.12	0.31	0.26	0.51	0.17	0.44	0.27
	0.4	50%	0.54	0.12	0.41	0.15	0.53	0.14	0.49	0.18
	0.4	60%	0.54	0.12	0.58	0.12	0.54	0.13	0.51	0.15
	0.3	50%	0.69	0.11	0.41	0.15	0.63	0.14	0.54	0.19
	0.3	60%	0.69	0.11	0.58	0.12	0.68	0.12	0.58	0.15
Req. to test	0.4	10%	0.94	0.16	0.05	1.0	0.23	0.44	0.20	0.70
	0.4	20%	0.94	0.16	0.16	1.0	0.51	0.41	0.45	0.69
	0.4	30%	0.94	0.16	0.36	0.69	0.71	0.28	0.61	0.50
	0.4	40%	0.94	0.16	0.60	0.35	0.83	0.21	0.75	0.37
	0.4	50%	0.94	0.16	0.79	0.20	0.85	0.17	0.79	0.25
	0.4	60%	0.94	0.16	0.96	0.14	0.94	0.16	0.89	0.20

Table 4.3: Recall and precision for the reconstructed traceability matrices of Calisto with rank-k subspace of 20%

to the user requirements as they have the same unique identifier, namely URCARxx.

We did the analysis including with the code included in as well as excluded from the corpus. In the first case the corpus consisted of almost 5500 terms, in the second case it consisted of almost 2300 terms. The second case did contain additional context from the provided documents. This additional text includes the introductions to specific groups of requirements or "glue" text to make the document readable and not just a list of requirements. In this chapter we discuss the results of the second case not including the code. We started with the same values for k, c and q as in the Pacman case.

Case Results

The precision and recall for all link selection strategies for the Calisto case study are summarized in Table 4.3. In this case study we observed that the constant threshold has a major impact on the results. When using the commonly accepted threshold of $c = 0.7$ LSI returns only few links. Using a threshold of $c = 0.4$ makes that the constant threshold has almost no influence on the results, but gives the best results. Filtering only on the constant threshold (c = 0.4) will cause the recall of design never to exceed 0.54 and the recall of test never to exceed 0.94.

Remarkable is the difference between the variable threshold strategy and one dimensional vector filter strategy for both link types. This can be explained by the distribution and the stretch in the data set. For example when we take the reconstructed links between requirements and design for q = 10%. In the case of applying the variable threshold strategy the threshold ε = 0.86 explaining the low recall and high precision. When applying the one dimensional vector filter strategy the ε = 0.57 for a specific column. The lower threshold returns more links increasing the recall and decreasing the precision compared to the variable threshold.

As for Pacman, we can see that the precision obtained using our two dimensional vector filter strategy is higher in all cases – in fact the improvement is even higher than we had for the Pacman case. However, the recall was consequently lower with respect to the first strategy using similar parameters. This can be explained as follows. First, again we have certain design artefacts containing many false positives. For example, one has 7 and another has 5 false positives. The second strategy reduced the number of false positives to 0 for the first case (causing the increase in precision). For the second case 4 false positives are filtered, but in this case also 2 correct links are filtered. This causes the recall to decrease.

When looking at the results of the reconstruction of the links between the requirements and design we can identify a separation between the part that describes the classes (functionality) and the part that describes the additional non-functional aspects such as portability and maintainability. In the part that describes the functionality of the system we have quite a few missing links (causing the low recall). In the part that describes the non-functional aspects we have many false positives (causing a decrease in precision). Looking at the text we see that the structure of the description is not that different, so this cannot be the reason for this separation. The cause for this separation should then be in the description itself. Indeed the non-functional aspects are more extensively described by text, as the classes are more described by diagrams, pseudo-code and data types.

Lessons Learned

- Reconstructing traceability between requirements and test cases again performs better than between requirements and design.

- In this case the description of the design was often captured in diagrams, pseudo-code or data types. This information is ignored by our analysis emphasizing the difficulties of tracing requirements to design.

- Eliminating columns with many hits is effectively done by the one dimensional vector filter strategy.

- Eliminating rows with many hits is effectively done by the two dimensional vector filter strategy.

- The Software Engineering Standard by the European Space Agency influences the choice for the work products to be analysed and has a direct impact on the result (see description of functional and non-functional aspects)

- It is indeed hard to get better results in a real-life industrial project compared to the Pacman case study. However, the results for "requirements – test" return comparable results in both case studies. With a similar recall the precision for Calisto is even better.

4.7.3 Case Study III: Philips Applied Technologies

For most products Philips Applied Technologies develops, almost 80–90% is reused from previous projects. The majority of new products has only limited new functionality that needs to be developed from scratch. The existing functionality is delivered by various Philips units.

In this case study the document set of an extension of a DVD+RW recorder is analyzed for requirements coverage. We want to know if all the requirements agreed in the contract are covered in the product. That is, we trace the requirements in the rest of the work products.

During product development a large number of requirements initially identified cannot be traced back to test cases or design documents: in a way they "get lost". This gets even worse when the system evolves over time. First ad-hoc attempts in two case studies showed that less than 10% of the total requirements can be recovered from the design and test documents (see Section 4.7.3). Furthermore, as the system evolves, new requirements are introduced in the system that cannot be traced back to the original requirements specifications.

Case Configuration

In this case the total set of documentation consists of one general document, which describes the document structure for this component. Furthermore there is one requirements document, which describes the requirements of the component, and an architecture document, which describes the delta that is introduced due to the new functionality. Finally, there are 5 interface specifications, 11 component specifications, which together form the design of the component and one test specification containing all the test scenarios and test cases.

In total, 20 documents are analyzed in this case study. These documents are all Microsoft Word documents and are based on the IEEE-1233-1998 standard for system requirements specifications [IEEE, 1998a]. Furthermore, they are not explicitly related. Thus, no reference traceability matrix is provided or anything comparable.

In this case it was not very obvious how to identify the concepts in the documentation. The requirements all have a unique identifier, but one of the problems is that the requirements specification consists of a requirements hierarchy. We need to choose the right granularity for our requirement concept. In this case study we took the highest level of requirements including their sub-levels as documents in the LSI analysis. This resulted in 7 high-level requirements as input for the LSI analysis.

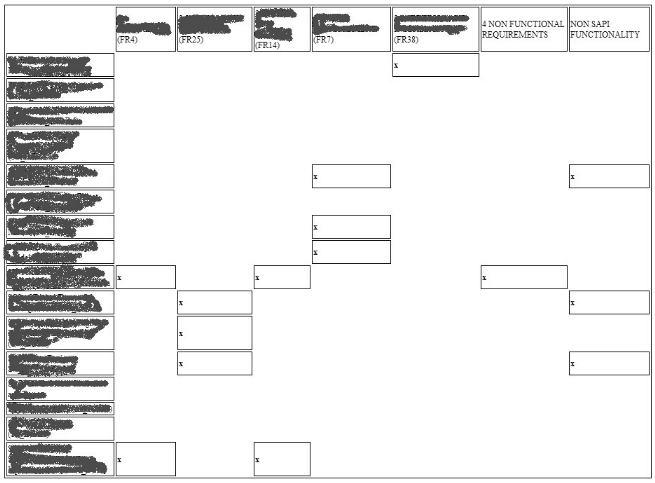
For design artefacts our choice was not very obvious either. Every document contains the design of one component or interface in the system. Taking a component as design artefact makes sense as the internal structure of the design documents is not really suitable for subdividing into smaller parts. So each design document will be a design artefact for the LSI analysis. In total this makes it 16 design artefacts.

The identification of concepts in the test specification was not really difficult. Test scenarios and test cases were easily recognizable in the test specification and therefore very suitable as input for the LSI analysis. In total we have 326 test cases. After pre-processing the documents this resulted in a corpus of more than 2500 terms representing the engineering domain.

Preliminary Analysis of Documents

Before we executed the LSI analysis on the document set we carried out a simple exploring analysis on the documents using Xpath expressions [Clark and DeRose, 1999]. In practice often simple search facilities are used to reconstruct links, for example, when a new requirement is agreed and needs to be processed in the documentation [och Dag et al., 2005]. In this first analysis we transformed the Microsoft Word documents to XML format and did some searching on the document set using Xpath expressions. Somewhat surprisingly, this analysis showed no direct links between the requirements documents and any other document. The unique identifiers were not traceable in the rest of the documents. Querying with the labels of a requirement identified only few links.

Still traceability should be incorporated somehow in the document set; after all this is the documentation set of one and the same product extension. Taking a closer look at the documents showed that this is indeed true.



	[redacted] (FR4)	[redacted] (FR25)	[redacted] (FR14)	[redacted] (FR7)	[redacted] (FR38)	4 NON FUNCTIONAL REQUIREMENTS	NON SAPI FUNCTIONALITY
[redacted]					x		
[redacted]							
[redacted]							
[redacted]							
[redacted]				x			x
[redacted]							
[redacted]				x			
[redacted]				x			
[redacted]	x		x			x	
[redacted]		x					x
[redacted]		x					
[redacted]		x					x
[redacted]							
[redacted]							
[redacted]							
[redacted]	x		x				

Figure 4.9: Reconstructed Traceability Matrix for Philips with a rank-k subspace of 20% and applying the Two Dimensional Vector Filter Strategy using c = 0.4 and q = 20%

When analyzing the documents and focusing on a specific requirement it showed that this requirement is transformed to a different technical term during architectural design. Components in the architecture get a name not directly related to the requirement.

From this experiment we learned that it is often very hard for non-experts to understand the documentation and use it effectively. This preliminary analysis emphasizes again that you need an expert to set up the traceability meta-model (define the concepts that need to be traced) and to judge the reconstructed traceability matrix.

Case Results

Executing the LSI analysis resulted in more informative results. The first remarkable result is that the similarity measures between the requirements and the design were much better than the similarity measures between the requirements and the test cases. The first two case studies showed the opposite results. A reason for this is the choice of the granularity of the concepts for analysis; high-level requirements and complete design components in combination with the structure of these documents. Every design component starts with a general description of the component. Part of that general description includes its functionality. This functionality matches the functionality described in the requirements description. Figure 4.9 shows a reconstructed traceability matrix for the requirements and design components. It has a link density of 13% and should give a direction for the experts to find their requirements.

The similarity measures between requirements and test cases showed results that were not as good. In this case there is a mismatch in the granularity of the documents used for analysis. The requirements were the same high-level requirements, but the test cases can be considered more as unit tests. These unit tests are written according to the designs and not the requirements. Clustering the 325 test to 16 higher level “test categories”, did not improve the reconstruction results. The quality of the similarities between de requirements and test cases was insufficient. It seems that the terminology has changed during the design phase making it difficult to reconstruct traceability between requirements and test.

Finally, in this case we were not able to compare the results with a provided traceability matrix, so we had to consult experts knowing the system. Disappointing is the fact that it is hard to validate that the reconstructed links are indeed correct. We found several links that are correct, but we did not come to an agreement for all links. For this reason we could not calculate the recall and precision.

Lessons Learned

The key lessons learned from this case study are:

- The choice for the work products to be traced is essential. The expert needs to decide on the work products that are most suitable to trace and the level of granularity to get the best results.
- The IEEE-1233-1998 standard for system requirements specifications shows a possibility to improve the reconstruction of traceability links between requirements and design. In IEEE-1233-1998 it is mandatory to provide a general description of the component.
- Again we see that is it hard to get better results in a real-life industrial project compared to the Pacman case study, which perhaps can be considered as an upper bound of the quality that LSI-based link reconstruction can achieve.

4.8 Discussion

4.8.1 Link Selection

With respect to the link selection strategies it is very hard to conclude when a strategy performs better. It very much depends on the application objectives. In the case studies we showed the results of applying LSI using different parameter values and compared the available link selection strategies. If we set our objective to realizing 100% recall, we can say that our two dimensional vector filter strategy performs best. With a recall of 100% the corresponding precision is higher for all cases.

We have also seen that the strategies cannot always reduce the number of false positives successfully. A good example of this is use case UC7 in the Pacman case. The returned links were all quite similar (between 0.91 and 0.96). The infinity of possible links in our strategies can be a disadvantage. In this case the cut point and cut percentage strategy will be more successful. They will, regardless of the actual value, simply select only the best x similarity measures (where x is a natural number).

Another point of attention is the constant threshold, which can also be disadvantageous. The constant threshold protects against losing too much quality in similarity measures, since every link should at least have a similarity measure $\geq c$. In some cases a correct link will be filtered only because of the constant threshold. Changing the values for the variable threshold will not help to recover these links. In this case a recall of 100% can never be reached as can be seen in the Calisto case and in the Philips Applied Technologies case concerning the requirements coverage in the test cases. In this case all similarity measures are simply lower than the constant threshold. The opposite is also possible, in the Pacman case the constant threshold was hardly of any influence.

A solution can be to make the constant threshold dependent on the minimal and maximal similarity measures or the mean of the entire matrix. This way it is more related to the actual output and not chosen randomly. It also represents the quality of the data. If for example the maximum similarity measure of the matrix is 0.67 and the constant threshold is 0.7 no links will be found with both strategies. Taking the mean of the total data set as constant threshold ensures links will be found. Note that this does not mean that for every requirement a link will be found, so for individual requirements the idea behind the constant threshold is kept. The analist together with the expert should decide on the quality of the data set.

4.8.2 Link Types

Furthermore, we observe a systematic difference in the performance of LSI for the different link types; requirements in design and requirements in test cases. The links between requirements and test case in general performed better than the links between requirements and design. The Philips Applied Technologies case was an exception, which can be explained by the wrong choice of granularity of the test cases. In general, the reason why test cases perform better is unclear and is still an open issue that remains to be answered.

However, one of the reasons LSI performs worse for "requirement – design" relations is the fact that for designs many diagrams are used for documentation (see again [Settimi et al., 2004]). Additionally, the diagrams are often badly described. Information retrieval techniques are less suitable for traceability reconstruction in this case. So, it very much depends on the provided document structure. If many diagrams are used to capture the information of the design, these should also be accompanied with a clear description. When defining the traceability model these things need to be considered and decided upon.

4.8.3 Link Density

We expect that there is a general range in the number of links that should be in an "ideal high quality" traceability matrix. We call this metric the link density for a traceability matrix of size $N \times M$.

Initially, we calculated the link density by dividing the total number of links set in the reference traceability matrix by the total number of links that can be set (the total set of possible candidate links). For example, in our Pacman case we have 28 "requirement - design artefact" links and we divide that number by 336 (14 requirements x 24 design artefacts). For both the Pacman and the Calisto case study we calculated the link density [1]. We see that the link density for our case studies is always between 7% and 10% (see Table 4.4). This observation can be an indication that the link density should always be situated between the 5% and 15% of the total candidate links when doing "adequate" traceability. Maybe this link density can be used as a guideline for traceability reconstruction. For example, in the Philips case where we do not have a reference traceability matrix. If you reconstruct a traceability matrix and the link density is 30%, it means that around 15–20% of the returned links are probably false positives.

This initial number gives only a general view on a traceability matrix. It gives an indication of how many links there should be in total. This ini-

[1]Again for the Philips case study we cannot calculate these numbers as we do not have a reference traceability matrix (nk – not known).

Case Studies	Pacman 2.2	Calisto	Philips
Number of Requirements (work products)	14	12	7
Number of Design Artefacts	24	48	16
Number of Test Cases	20	79	326
Number of “requirement - design artefact” links	28	59	nk
Number of “requirement - test case” links	19	80	nk
Link density between “requirement - design artefact” links	0.08	0.10	nk
Link density between “requirement - test case” links	0.07	0.08	nk
Link density relation “requirement:design artefact”	1:1,7	1:4	nk
Link density relation “requirement:test case”	1:1,4	1:6,6	nk

Table 4.4: Link Density Statistics

tial number does not give any direction on the structure of the traceability matrix, while in most cases we know that a traceability matrix is structured around its diagonal; the first requirement is implemented in the first design artefact and tested by the first test case, and so on...

In the ideal case we want to know the specific relation of the link density between the requirements and all other work products (design artefacts and test cases). For example, each requirement has on average a link with 2 test cases. If we assume this number is constant, we know that every additional requirements has 2 corresponding test cases. In Table 4.4 we calculated this link density relation for our case studies. It shows that each requirement in the Calisto case study has on average 4 links to a design artefact. Future work should confirm if there is indeed an “ideal” constant link relation between requirements and other work products as we propose here.

4.8.4 Reserach Issues

On the basis of our case studies we can identify the following research issues that need to be addressed before LSI can be applied for the purpose of link reconstruction in an industrial context:

- How can we take advantage of the (few) cross references that are typically included already in the documents? For example, does it make sense to give requirement identifiers used in design documents a high weight in the term-by-document matrix?

- How should the hierarchical structure of, for example, design or requirements documents be dealt with in link reconstruction? In our present experiments we created a flat set of documents. How can LSI be adjusted so that if possible links in the most detailed documents are taken into account, moving up higher in the hierarchy (aggregating the paragraphs) when this does not lead to a sufficient number of results?

- What documents should be included in the corpus? For example, do we get better requirements-to-design links when we also include the test documents in the corpus? Why (not)?

- Can we come up with link reconstruction techniques tailored towards specific types of links? For example, do we need different strategies for reconstructing requirements-to-design and for requirements-to-test links?

- Are the recall and precision that are achieved sufficient for practical purposes? Can we make sufficiently accurate predictions of certain coverage views based on the (incomplete) links we can reconstruct?

- Is the link density a good measure to characterize a "good" traceability matrix? If so, what will be the range for the number (5–15%) for various link types?

4.9 Contributions and Future Work

The objective of this chapter was to investigate the role latent semantic indexing can play in order to reconstruct traceability links. From the previous discussion we can conclude that LSI can indeed help increasing the insight in a system by means of reconstructing the traceability links between the different work products produced during development. We consider following to be our main contributions:

- We have provided a methodology, MAREV, for automating the process of reconstructing traceability and generating requirements views.

- We defined a new two-dimensional vector filter strategy for selecting traceability links from an LSI similarity matrix.

- We provided a tool suite, REQANALYST, for reconstructing traceability links including support for quantitative and qualitative assessment of the results.
- We applied our approach in three case studies of which one was an industrial strength case in the consumer electronics domain.
- For each of the case studies, we offered an analysis of factors contributing to success and failure of reconstructing traceability links.
- We identified the most important open research issues pertaining to the adoption of LSI for link reconstruction purposes in industry.

Our future work will be concerned with the open issues listed in the discussion section. Furthermore, we would like to extend our work along three lines. First, we want to study other links than those between requirements on the one hand and test cases and design decisions on the other. Furthermore, we are in the process of extending our experimental basis. In particular, we are working on two new case studies in the area of consumer electronics and traffic monitoring systems. In these case studies we want to focus more on our last step: the generation of requirements views that are useful in practice. Finally, we will further explore how to improve the performance of LSI and the link selection strategies in real-life applications. All can be implemented in REQANALYST.

4.10 Epilogue

This chapter presented our main results, the MAREV approach and the REQANALYST tool suite, as potential solutions for the observations, and lessons learned in Chapters 2 and 3. Futhermore, it can be considered as an implementation of the RES framework also presented in Chapter 3.

In Table 4.5 we depict a mapping of the steps of MAREV to the lessons learned, and the RES framework discussed in Chapter 3. The reasons for not using state-of-the-art technologies, as discussed in Chapter 2, are deliberately left out of the table as this mapping is already discussed in Section 3.8.

When we look at Table 4.5, we see that all lessons learned and RES processes are covered by the MAREV steps. So, we can conclude that MAREV takes into account the lessons learned from our industrial case study, and covers the three main processes of the RES framework.

The one thing that remains is applying MAREV and REQANALYST in an industrial case study. In the next Chapter 5, we again use the TMS case, discussed in Chapter 3, to apply our approach and assess its value for industry.

MAREV Steps	Lessons Learned (Ch3)	RES framework (Ch3)
Define Traceability Model	Lesson 3	Process 2
Identify Work Products	Lesson 1 & 2	Process 1 & 2
Preprocess Work Products	Lesson 1 & 2	Process 1
Reconstruct Traceability	Lesson 3 & 4	Process 2
Select Links	Lesson 3 & 4	Process 2
Generate Views	Lesson 1	Process 3
Tackle Changes	Lesson 3 & 4	Process 2

Table 4.5: Mapping of results Chapter 3

Chapter 5

An Industrial Case Study in Reconstructing Requirements Views[1]

Requirements views, such as coverage and status views, are an important asset for monitoring and managing software development projects. We have developed a method that automates the process of reconstructing these views, and we have built a tool, REQANALYST*, that supports this method. This chapter presents an investigation as to which extent requirements views can be automatically generated in order to monitor requirements in industrial practice. The chapter focuses on monitoring the requirements in test categories and test cases. In order to retrieve the necessary data, an information retrieval technique, called Latent Semantic Indexing (LSI), was used. The method was applied in an industrial study. A number of requirements views were defined and experiments were carried out with different reconstruction settings for generating these views. Finally, we explored how these views can help the developers during the software development process.*

5.1 Introduction

A "requirements view" on a system or development process offers a perspective on that system in which requirements assume the leading role [Nuseibeh et al., 1994]. A requirement view can be a combination of artefacts such as requirements and design information, showing how a requirement is transformed into a design artefact, and indicating how and where a requirement is covered by specific design artefacts, or where it is located in

[1]This chapter was originally published as: Marco Lormans, Hans-Gerhard Gross, Arie van Deursen, Rini van Solingen, and Andre Stéhouwer. Monitoring requirements coverage using reconstructed views: An industrial case study. In *Proc. of the 13th Working Conf. on Reverse Engineering*, pages 275–284, Benevento, Italy, October 2006

the system architecture. Examples are coverage views, such as “which design artefacts address which requirement?”, or status views, such as “which requirements are already implemented?” The various requirements views help to avoid inconsistencies within the documentation of one kind of work product (requirements specification) or between the documentation of different types of work products (requirements specification and architectural design document) [Easterbrook and Nuseibeh, 1995]. Requirements views help in improving the coherence between the work product documents, and lead to higher overall quality of the work products.

Requirements views are essential for successful project management, and for monitoring the progress of product development. In an outsourcing context, reporting progress in terms of requirements is particularly important, since the customer is much less aware of the system breakdown or of implementation issues, and more likely to be interested primarily in his requirements.

Unfortunately, capturing, monitoring, and resolving multiple views on requirements is difficult, time-consuming as well as error-prone when done by hand [Nissen et al., 1996]. The creation of requirements views necessitates an accurate traceability matrix, which, in practice, turns out to be very hard to obtain and maintain [Gotel and Finkelstein, 1994; Dömges and Pohl, 1998; Graaf et al., 2003; Ramesh et al., 1995; Lindvall and Sandahl, 1996]. The tools currently available on the market, such as Telelogic DOORS and IBM Rational RequisitePro, are often not sufficient: keeping the traceability consistent using these tools is hard and involves significant effort [Lormans et al., 2004; Alexander, 2002].

To remedy this problem, a significant amount of research has been conducted in the area of reverse engineering of traceability links from available software development work products [De Lucia et al., 2004; och Dag et al., 2005; Huffman Hayes et al., 2006]. Our own line of research has focused on the use of information retrieval techniques, in particular *latent semantic indexing* (LSI) [Deerwester et al., 1990], for this purpose, and on the application of the reconstructed matrices for view reconstruction, specifically. We incorporated our ideas in a method, called MAREV, and implemented the method in a tool, called REQANALYST [Lormans and van Deursen, 2005, 2006, 2008].

While significant progress in this area has been documented, a number of open research issues exist, which we seek to explore in this chapter. An initial question to be addressed is not related to the case study performed. It is about which requirements views are most needed in practice. To answer this question, a questionnaire was sent out to a dozen practitioners, and from the answers three important groups of views were distilled, which are described in detail.

As unit of analysis, one development project of Logica, an international

IT services supplier, was scrutinized for the case study. The primary question addressed through this exploratory case study was how and to which extent requirements views can be reverse-engineered from existing work products. An important question hereby is whether the approach we proposed [Lormans and van Deursen, 2005, 2006, 2008] may be used to reconstruct these views. To answer this question, it is described how our own prototype tool (REQANALYST) has been extended to support these views, offering project stakeholders means to inspect the system and development progress in terms of these views. Another question to be addressed through the case study is whether these reconstructed views can help in a real life software development process.

In the software development project under investigation in this case study, a traffic monitoring system (TMS) is developed, and it is outsourced to Logica. In the project, progress reporting to the customer must be done in terms of requirements, making accurate requirements views an essential success factor. The chapter discusses the way of working in this project, and looks at how and to which extent reconstructed links can be used to support and enhance the way of working. In the case study, the focus lies on requirements views that are related to testing artefacts.

The remainder of this chapter is organized as follows. Section 5.2 discusses existing work in the area of requirements views and reverse engineering of traceability matrices. Section 5.3 summarizes the methodology for generating requirements views, called MAREV. Sections 5.4, 5.5, and 5.6 present the requirements views aimed at, the way they are implemented in the REQANALYST tool, and the case study performed at Logica, respectively. The chapter concludes with a discussion, a summary of contributions, and suggestions for future research.

5.2 Related Work

5.2.1 System Views

The term 'view' is often used in the area of software engineering, especially in the area of requirements engineering. Views are generally introduced as a means for separation of concerns [Nuseibeh et al., 1994] and mostly represent a specific perspective on a system. This perspective is often a subset of the whole system in such a way that its complexity is reduced. Each stakeholder is interested in a different part of the system. A stakeholder may be a developer who is only interested in a small part (a component, for example) of the complete system. The perspective that a view represents, can also be an abstraction of the system. It can give an overview of the whole system without providing too many details. Such a view from the top can be useful for a project manager or a system architect.

Nuseibeh *et al.* discuss the relationships between multiple views of a requirements specification [Nuseibeh et al., 1994; Easterbrook and Nuseibeh, 1995]. Most systems that are developed by multiple participants have to deal with requirements that overlap, complement and contradict each other. Their approach focuses on identifying inconsistencies and managing inconsistencies in the requirements specification. It is based on the viewpoints framework presented by Finkelstein *et al.* [Finkelstein et al., 1992]. This framework helps in organizing and facilitating the viewpoints of different stakeholders.

Zachman proposes "The Architecture Framework" focusing on information system views [Zachman, 1987]. Hay uses the six views of this framework for requirements analysis [Hay, 2003]. In his approach, he uses the framework to define the requirements analysis process, which can be seen as the process of translating business owners' views into an architect's view.

The concept of a "view" also appears in other areas of software engineering such as architectural design. Kruchten introduced his "4 + 1 view model for architecture", where he defined five different concurrent perspectives on a software architecture [Kruchten, 1995]. Each view of this model addresses a specific set of concerns of interest to different stakeholders. Other examples are the "Siemens' 4 views" by Hofmeister *et al.* [Hofmeister et al., 1999], the IEEE standard 1471 [IEEE, 2000], and the views discusses by Clements *et al.* in their book "Documenting Software Architectures" [Clements et al., 2002, 2003]. Van Deursen *et al.* also discuss a number of specific views for architecture reconstruction [van Deursen et al., 2004].

Finally, Von Knethen discusses view partitioning [von Knethen, 2001]. She considers views on the system, distinguishing, for instance, the static structure from the dynamic interactions in the system. These views support the process of impact analysis in two ways: they improve (1) the planning (estimating costs) as well as (2) the implementation of changes. Furthermore, the views allow the system to incorporate changes in a consistent way.

Although, much research has been done in the area of system views, there is no general agreement on what such views should look like, or which information they should contain. Every project setting seems to have its own specific information needs. Thus, views must be flexible in meeting these needs.

5.2.2 Document Standards, Templates and Reference Models

Another approach for separating concerns is to use a well structured document set, conforming to known templates such as MIL-std 498 [Department of Defence, USA, 1994], Volere [Robertson and Robertson, 2000], IEEE-std-830 [IEEE, 1998b], or IEEE-std-1233 [IEEE, 1998a]. These templates help in getting an overview of what the system does, but they are often not sufficient. Project managers, but also other team members, need fast access to this data, and, preferably, they would like only a subset of the whole pile of documents produced during the development life-cycle. Current templates are not sufficiently flexible, and they are difficult to keep consistent during development.

Nissen *et al.* show that meta-models help managing different requirements perspectives [Nissen et al., 1996]. The meta-models define which information is available and how it is structured in the life-cycle. This comprises the development artefacts, including their attributes, and additionally, the traceability relations permitted to be set between these artefacts. If the information is not stored sometime in the life-cycle, it can never be extracted and used in a view. An important area of research is developing these meta-models [Ramesh and Jarke, 2001; von Knethen, 2001; Toranzo and Castro, 1999; Maletic et al., 2003; Zisman et al., 2003], constraining the views to be generated.

Von Knethen proposes traceability models for managing changes on embedded systems [von Knethen, 2001; von Knethen et al., 2002]. These models help estimating the impact of a change on the system, or help to determine the the links necessary for correct reuse of requirements. According to Von Knethen, defining a workable traceability model is a neglected activity in many approaches. Our earlier research confirms the importance of defining a traceability model [Lormans et al., 2004]. Some initial experiments concerned a static traceability model. New insights suggest a dynamic model, in which new types of links can be added as the way of working evolves during the project. The need for information as well as the level of detail change constantly in big development projects [Dömges and Pohl, 1998].

5.2.3 Traceability Support and Recovery

Traceability support is required in order to reconstruct requirements views from project documentation. Several traceability recovery methods and supporting tools already exist, each covering different traceability issues during the development life-cycle. Some discuss the relations between source code and documentation, others address the relations between requirements on different levels of abstraction.

De Lucia *et al.* present an artefact management system, which has been extended with traceability recovery features [De Lucia et al., 2004, 2007]. This system manages different artefacts produced during development such as requirements, designs, test cases, and source code modules.

The Information Retrieval (IR) technique that De Lucia *et al.* use for recovering the traceability links is Latent Semantic Indexing (LSI). Furthermore, they propose an incremental traceability recovery process in which they try to identify the optimal threshold for link recovery in an incremental and iterative way [De Lucia et al., 2006b]. The threshold determines which links should be considered as candidate links by a tool and which not.

Natt och Dag *et al.* [och Dag et al., 2005] and Huffman Hayes *et al.* [Huffman Hayes et al., 2006] use traceability reconstruction primarily for managing requirements of different levels of abstraction, such as reconstructing links between business and system requirements. Both, Natt och Dag *et al.* and Huffman Hayes *et al.*, have developed a tool to support their approaches. In [och Dag et al., 2005], Natt och Dag *et al.* discuss their approach and tool, ReqSimile, that implements the basic vector space model which also forms the basis for latent semantic indexing. They report their experiences in [och Dag et al., 2005], and the results are comparable to what we found.

In their tool called RETRO, Huffman Hayes *et al.* have implemented various methods for recovering traceability links [Huffman Hayes et al., 2006]. They also applied their approach in an industrial case study.

Cleland-Huang *et al.* define three strategies for improving dynamic requirements traceability performance: hierarchical modelling, logical clustering of artefacts and semi-automated pruning of the probabilistic network [Cleland-Huang et al., 2005]. They are implementing their approach in a tool called Poirot [Lin and *et al.*, 2006]. They have also defined a strategy for discovering the optimal thresholds for determining candidate links [Zou et al., 2004].

Antoniol *et al.* [Antoniol et al., 2002] use information retrieval methods to recover the traceability relations between C++ code and documentation pages, and between Java code and requirements. Marcus and Maletic [Marcus and Maletic, 2003], and Di Penta *et al.* [Di Penta et al., 2002] use information retrieval techniques for recovering the traceability relations between source code and documentation. In addition, Di Penta *et al* [Di Penta et al., 2002] augmented their traceability approach with models of programmer behaviour. The IR methods in these cases are mostly applied for reverse engineering traceability links between source code and documentation in legacy systems.

Marcus *et al.* [Marcus et al., 2005b] discuss how to visualize traceability links, and they introduce a tool, TraceViz, that implements their proposed requirements for traceability visualization. IR techniques are also used for improving the quality of the requirements set. Finally, Park *et al.* use the calculated similarity measures for improving the quality of the requirements specifications [Park et al., 2000].

None of the discussed traceability reconstruction methods support the generation of requirements views for monitoring the requirements in the other work products. One reason for this is that current methods do not explicitly discuss the links that can be reconstructed and cannot be reconstructed. This makes it hard to define specific views and retrieve the information needed to manage a project with respect to evolving requirements.

5.3 MAREV and REQANALYST

In our earlier work, we have proposed an approach for reconstructing requirements views [Lormans and van Deursen, 2005] and experimented with the reconstruction of traceability links in several case studies [Lormans and van Deursen, 2006, 2008]. The method is called MAREV: Methodology for Automating Requirements Evolution using Views. Besides that, the method has been implemented in a tool called REQANALYST. This section provides a brief overview of the tool as well as the underlying method.

5.3.1 MAREV: A Methodology for Automating Requirements Evolution using Views

MAREV consists of the following seven steps (see also Lormans and van Deursen [2005, 2006, 2008]).

Step 1: Defining the Traceability Meta-model

The underlying traceability meta-model defines the types of work products (e.g. business requirements, system requirements, design artefacts, or test cases, and so on) and the type of links that are permitted within the development life-cycle. The choices made for defining the meta-model largely depend on the needs of the application domain. Examples can be found in [Ramesh and Jarke, 2001; von Knethen, 2001; von Knethen et al., 2002; Toranzo and Castro, 1999; Maletic et al., 2003; Zisman et al., 2003].

Step 2: Identifying the Work Products

The work products are identified in the provided project documentation or configuration management system, and mapped onto the traceability meta-model. Each work product is given a type and unique identifier if it has not already been assigned one. This unique identifier is a code plus a unique number, for example, a functional requirement description can have an identifier of the type "FR*xx*", where *xx* represents the number. This results in a set of functional requirement descriptions with the unique identifiers "FR01", "FR02", and so on. This step must be executed for every work product defined in the traceability meta-model. If requirements management tools such as Telelogic's DOORS are used, unique identifiers are provided automatically.

Step 3: Pre-processing the Work Products

The work products are pre-processed to support automated analysis for them. The text of each work product needs to be extracted and transformed into plain text. This step includes typical information retrieval activities such as lexical analysis, stemming, and so on.

Step 4: Reconstructing the possible Traceability Links

The likely traceability links are reconstructed for which Latent Semantic Indexing [Deerwester et al., 1990] is used. The result of this step is the complete set of candidate traceability links.

Latent Semantic Indexing (LSI) is an information retrieval technique based on the vector space model. It assumes that there is an underlying or latent structure in word usage for every document set [Deerwester et al., 1990]. LSI uses statistical techniques to estimate this latent structure. A description of terms and documents based on the underlying latent semantic structure is used for representing and retrieving information. LSI starts with a matrix of terms by documents. Subsequently, it uses Singular Value Decomposition (SVD) to derive a particular latent semantic structure model from the term-by-document matrix. The result is a reduced model, the rank-k model with the best possible least square fit to the original matrix of terms by documents [Deerwester et al., 1990]. Subsequently, this model can be used to determine a similarity matrix.

Once all documents have been represented in the LSI subspace, the similarities between the documents can be computed. The cosine between their corresponding vector representations can be used for calculating this similarity metric. The metric has a value between $[0, 1]$ with a value of 1 indicating that two documents are (almost) identical. These measures can be used to cluster similar documents, or for identifying traceability links between the documents.

Finally, LSI does not rely on a predefined vocabulary or grammar for the documentation (or source code). This allows the method to be applied without large amounts of pre-processing (i.e., stemming) or manipulation of the input, and, therefore, it can reduce the costs of traceability link recovery considerably [Maletic et al., 2003; De Lucia et al., 2004].

Step 5: Selecting the Relevant Links

The possibly relevant links are selected automatically from the complete set of candidate links (from the LSI) using various link selection strategies. In our previous work, we proposed two link selection strategies, a *one* and a *two dimensional vector filter strategy* on the similarity matrix [Lormans and van Deursen, 2008]. These link selection strategies combine the already known strategies *constant threshold* (represented by the symbol c) and *variable threshold* (represented by a percentage q) discussed by De Lucia *et al.* [De Lucia et al., 2004]. The one-dimensional filter strategy considers every single column of the similarity matrix separately. Each column vector of the similarity matrix is taken as a new set of similarity measures, and it combines, for each column, the constant and the variable threshold approaches. The two-dimensional filter strategy extends the one-dimensional strategy by considering both dimensions of the similarity matrix. The benefits of these strategies are that they guarantee certain level of quality by using the constant threshold, and, yet, they take only the best $k\%$ of the links for a certain work product. Both strategies are described in detail in [Lormans and van Deursen, 2008], and they have shown improved results in terms of recall and precision. As with all information retrieval techniques, it is not guaranteed that all correct links are indeed found: both false negatives and false positives may arise.

Step 6: Generating Requirements Views

In this step, the requirements views are generated using the reconstructed traceability links. This step will be the focus for the rest of this chapter.

Step 7: Tackling Changes

Finally, the reconstructed traceability links and generated requirements views need to be able to tackle changes in the requirements. Therefore, the validated traceability matrix and the newly reconstructed traceability matrix need be compared after each run of the MAREV approach. Users can then validate the impact of a requirements change in the traceability matrix.

5.3.2 The REQANALYST Tool Suite

In order to support the MAREV approach, we developed the REQANALYST[1] [Lormans and van Deursen, 2008] tool. This tool can reconstruct

[1]REQANALYST is available from http://swerl.tudelft.nl/bin/view/Main/ReqAnalyst.

traceability information and generate requirements views using that reconstructed traceability information. In this section we summarize our earlier work on REQANALYST. In Section 5.5 we focus again on generating our requirements views using REQANALYST.

Extract-Query-View Approach

REQANALYST adopts the Extract-Query-View approach used in many reverse engineering tools van Deursen and Moonen [2006]. In this approach, first, the relevant data from the provided documents is extracted. This data, the work products, and, if available, the reference traceability matrices, are stored in a database. For reconstructing the traceability links, queries can be conducted on the database. The reconstructed information, combined with the data from the database, is used to generate the requirements views.

The reference traceability matrix is optional, and contains the correct links according to the experts in the project. It is only required to assess the outcomes of the tool, addressing the question as to which extent requirements views can be reconstructed automatically. Typical (reengineering) projects do not have such a matrix, to start with, and the ultimate goal is to generate this matrix automatically from the existing project documents, i.e., through using LSI.

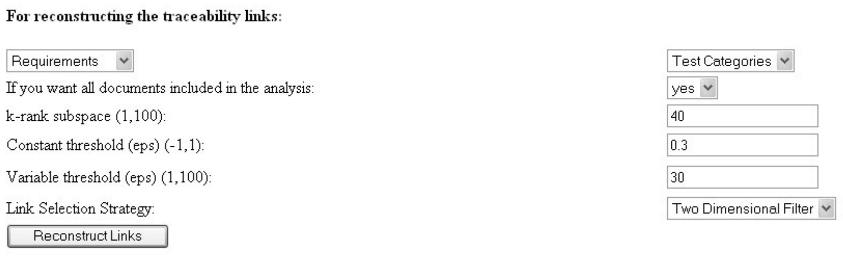
Implementation

REQANALYST is implemented using standard web-technology. For storing the data, a MySQL database is used. It is implemented as a a Java web application using Java Servlets and Java Server Pages (JSP). For the case study, the Apache Tomcat 5.5 web server was taken for deployment.

Functionality

A REQANALYST session starts by logging in. Each user of REQANALYST has specific rights to see certain projects. After authentication the user gets a list of projects. Once the user has chosen a project, REQANALYST shows the main menu. This main menu follows the steps from the Extract-Query-View approach van Deursen and Moonen [2006], including functionality for extracting the data from the complete set of provided documentation, and options for setting the parameters of the LSI reconstruction and the choice for a link selection strategy. Figure 5.1 shows a excerpt of the tool.

Once REQANALYST has executed a reconstruction, a menu appears showing the reconstructed traceability matrix and a number of options for generating various requirements views. This menu shows all the metrics rel-


For reconstructing the traceability links:

Requirements — Test Categories

If you want all documents included in the analysis: yes

k-rank subspace (1,100): 40

Constant threshold (eps) (-1,1): 0.3

Variable threshold (eps) (1,100): 30

Link Selection Strategy: Two Dimensional Filter

Figure 5.1: Input screen for Traceability Reconstruction

evant for assessing the reconstruction, such as recall, precision and the number of false positives and missing links in the traceability matrix. This menu is also used to generate the various requirements views.

Browsing in ReqAnalyst

An important feature of ReqAnalyst is the possibility to browse the reconstructed results. It allows engineers to inspect the reconstructed traceability matrix and browse through the traceability links, implemented as hyper links. When following the hyper link, all the information concerning the two entities involved becomes available and can be inspected. For example, the original text of both entities is shown in one view.

Furthermore, the reconstructed matrix can be compared with a reference matrix, if available. The reference matrix represents the traceability matrix as determined by the developers of a system and is only required for evaluation purposes. The correctly reconstructed links (correct positives) are indicated with an “X” and the cell is coloured green. The false positives are indicated as “fp” and are coloured yellow. Furthermore, the false negatives (missing links) are indicated through “fn” and are coloured red.

5.4 Which Views are Needed in Practice?

While Marev and ReqAnalyst provide a method and tool support for obtaining requirements views, it is less obvious which requirements views are actually needed in practice. To address this issue, we have set up a questionnaire and distributed it among various practitioners. Below, the questionnaire is described, and the three main types of views that emerged

Questions:	
1)	What is your role in the software development life-cycle?
2a) 2b)	What do you expect from a requirements view? What information would you like to see in a requirements view? (Examples: coverage, functionality, status)
3a) 3b)	What do you think persons in the roles below expect from a requirements view? - Project Manager - Requirements Engineer - System Architect - Programmer - Test Engineer - Quality Manager - Other? (please also define the role) What information do you think they would like to see? (Do not fill in your own role again)
4)	Do you think it is feasible to extract this information from the work products currently produced during development? (requirements specifications, design documents, etc.)

Table 5.1: Topics put forward in the Questionnaire

from our survey are discussed.

5.4.1 Requirements View Questionnaire

The goal of our questionnaire was to get an impression of which views would be helpful and which information these views should represent. The questions asked to the participants are shown in Table 5.1. The questionnaire was distributed among people of various roles within the software development life-cycle. The roles distinguished are: project manager, software process improvement / quality manager, product marketing manager, requirements engineer, system/software architect, programmer and test engineer, as well as more specific roles such as product owner and usability designer.

The questionnaire was spread among the industrial partners of the MERLIN project[1]. The MERLIN project is a European research project in the area of global software development in which various universities and companies participate. In total, the questionnaire was spread among all 7

[1]www.merlinproject.org

industrial partners. We got a response from 5 of the companies involved, all of which provided many replies according to their various roles. In total we had 12 fully filled in questionnaires containing around 100 descriptions of desirable views for different roles in the life-cycle.

It was also asked if these views could be extracted from the work products they currently produce during the development life-cycle. Most respondents think that this should be possible, because this information should generally be contained somewhere in the work products. However, the exact location of this information is not always known.

5.4.2 Main Outcomes

The outcome from the questionnaire is that requirements should be able to be traced into their associated subsequent work products. A challenge in that respect is that, in many cases, the readability of many of the work products leaves much to be desired, and that it is often hard to get an overview of the whole system. In addition to that, stakeholders can easily get lost when looking for information if there are too many possible links to follow. Our views should address this issue, and make it easier to deduce the right information needed for the view in question.

Another lesson learned from the questionnaire is that the following information is desirable in a requirements view:

- For each requirement, the source, description, motivation, importance, history, status and dependencies to other work products. This is actually an obligation of the new safety standard ISO/WD 26262 for systems in the automotive domain that is currently being developed [Findeis and Pabst, 2006].
- For each group of requirements, a list of all requirements, the status of their implementation and verification (not tested, test passed, test failed).
- Life-cycle paths; per requirement, the complete path it undergoes during the life-cycle. Two paths are of interest for the developers: the Requirements–Implementation path and the Requirements–Test path.
- For all the requirements, the coverage in a certain work product. These work products can, for example, be a lower level of requirements, the design or the test cases.

From the questionnaire it was concluded that various developers and managers are interested in specific information about a certain requirement (see first and third bullet) or a group of requirements, sometimes in relation to other work products (see last bullet).

From the answers to this questionnaire three types of views were distilled: *Coverage* views, *Life-cycle Path* views, and *Status* views. Below, these are discussed in detail.

5.4.3 Coverage Views

Requirements *coverage* views focus on the localization of the requirements in the rest of the system. These views show whether and where a certain requirement is associated with another artefact in the system. This can be coverage in the system architecture, in the detailed design, or in the test cases, to name only a few instances. The number of different types of coverage views depends on the meta-model defined for the development process. It prescribes which phases are defined and which work products are produced during these phases. This view is often used for tracing requirements changes into subsequent work products [von Knethen, 2001; Settimi et al., 2004], and it can, therefore, be used for impact analysis in system evolution [Bohner and Arnold, 1996].

According to Costello *et al.*, requirements coverage is defined as: *The number of requirements that trace consistently to the next level up or down* Costello and Liu [1995]. They originally defined this metric for requirement to requirement coverage. As this definition is very general, it is also suitable for the coverage of requirements to other work products.

Hull *et al.* also define three so called traceability metrics Hull et al. [2002]. One of them, *Traceability Breadth*, relates to coverage. It measures the extent to which requirements are covered by the adjacent layer above or below (within the defined meta-model).

We define requirements coverage as follows: If a link between a requirement and another work product, for example a test case, exists, and this link is correct, then is the requirement covered by that work product. The requirements coverage view shows which requirements are covered by work products, as well as the percentage of these requirements with respect to the total number of requirements. For example, the percentage of requirements (compared to all requirements) covered by a test case can be defined as follows:

$$coverage_{test} = \frac{|requirements_{test}|}{|requirements_{total}|},$$

where $coverage_{test}$ represents the coverage in the test case specification, $requirements_{test}$ the number of requirements traced consistently by test cases and $requirements_{total}$ the total number of requirements.

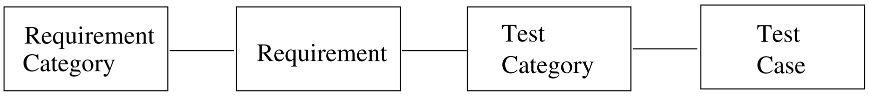


Figure 5.2: An example of a life-cycle path

This coverage metric is very general and fundamental, and can be used for requirements coverage in other life-cycle phases as well, such as the coverage of requirements in the design.

5.4.4 Life-cycle Path Views

Requirements *life cycle path* views deal with the transformations and decompositions that a requirement undergoes throughout the development process. The questionnaire showed that two life-cycle paths are important: the Requirements-Implementation path and the Requirements-Test path. When comparing this to the well-known V-model, it becomes apparent that these are the horizontal and vertical dimensions of this life-cycle model.

The length of a life-cycle path is captured by the second traceability metric of Hull *et al.*, called *Traceability Depth* Hull et al. [2002]. This metric relates to the number of layers along which the traceability extends, for example the layers along the left leg of the V-model for capturing software development. It can also be seen as the number of (model) transformations between the different types of work products.

As an example, Figure 5.2 shows a Requirements–Test-Path in a traceability meta-model. This example is taken from our case study which will be discussed in Section 5.6. It shows that the focus of interest lies in following the path of the requirements categories, via requirements and test categories, to test cases. The path extends along 4 layers according to Hull *et al.* Note, that a coverage view addresses only one layer.

In order to further characterize a life-cycle path view, another metric from Hull *et al* is relevant as well. This other metric, called *Traceability Growth*, measures how a requirement expands down through the layers of the meta-model (in our case the life-cycle path) Hull et al. [2002]. For example, a requirement can be covered by one test case or by multiple test cases. This is also a useful metric for impact analysis, which is why we will include it in our life-cycle path view.

5.4.5 Status Views

Requirements *Status* views concern the status of a (set of) work product(s) such as a (set of) requirement(s). The view shows a specific status of the work product in the life-cycle. In other words, if a link exists from a requirement to a source code document, it can be assumed that the status of the requirement is "implemented". In addition, this information can be used in order to obtain a coverage measure for the number of implemented requirements for project management purposes. For example, status views may be associated with a measure expressing that 60% of all requirements have the status "implemented". A project manager can use this information to monitor the progress of the project. Other management information can be obtained by computing percentages of requirements that have reached a certain status such as "tested successfully".

Traceability support is often not enough to generate complete status reports of requirements, for example, when a project manager needs to know whether all requirements have passed a test. Traceability can help identifying the requirements in the test document (the document that describes the test), and hopefully also in the test report document. The latter contains the information whether the implementation of a specific requirement has passed its test or not. This information needs to be extracted from the document and included in the status view as well.

In the case study, this extra status information was monitored in addition to the normal traceability data. We tried to retrieve "richer information" concerning the status of the requirements. For example, a status view for an individual requirement can show its relations to other work products (coverage) including its status such as "covered by test, but not tested yet", "covered by test, and failed the test" or "covered by design, but not covered by test".

5.5 Implementing the Views in ReqAnalyst

The three views presented should make it possible to obtain continuous feedback on the progress, in terms of requirements, of ongoing software development or maintenance projects. Furthermore, they facilitate communication between project stakeholders and different document owners. This section discusses how our ReqAnalyst tool as described in Section 5.3.2 has been extended to incorporate support for these three views.

Coverage Views

The "Coverage View" as implemented in ReqAnalyst shows the number of requirements that are covered (linked correctly) by some other work product, and the total number of requirements that are analyzed. It also shows the coverage percentage as defined in Section 5.4.3, i.e., percentage of the correctly reconstructed links between requirement and associated other work product. Furthermore, it lists the requirements with their description and the related artefacts of the other work product. Besides the coverage, it is also possible to see which requirements are not covered by the other work product. We call this view the "Orphans View". This view shows the same results as the coverage view, except for the related artefacts: as there are none, these cannot be shown. This view is important for developers as they need to inspect why the requirements in this view are not yet covered in the system.

Life-cycle path Views

The "Life Cycle Path View" as implemented in REQANALYST displays the stages involving a requirement. In particular, a tabular view is shown, illustrating the work products a requirement is related to, such as requirements categories or test cases. This table can also be used to obtain the values for the traceability growth metric at the various levels in the life cycle path. An example for our case study based on the traceability model in Figure 5.2 is shown at the end of the chapter in Figure 5.5.

Status Views

The "Status View" as implemented in our REQANALYST tool is based on the observation that every entity of a work product type can have multiple status attributes attached to it. So, besides extracting the relevant data for executing the automated reconstruction, it can also extract the additional status attributes from the provided documentation. These status attributes are saved separately in the database. When a user generates a view of a specific "requirement – test case" relation, for instance, it can also show the status attributes concerning this relation.

5.6 Case Study: Logica

The previous sections discussed the three most essential views considered by engineers, and we have proposed a method and a tool for reconstructing these views automatically from the available work products. This section presents the case study performed at Logica aimed at illustrating how the method and the tool work out in practice.

We begin with laying out the case study design, making use of the guidelines provided by Yin [Yin, 2003a,b]. Then, after discussing the nature of the project and the development process followed, we describe which requirements documents we used as input for the reconstruction effort. Furthermore, we explain the reconstruction approach and its specific parameter settings used, followed by a discussion of the traceability matrices obtained. Finally, we discuss how these matrices lead to the requirements views considered.

5.6.1 Case Study Design

The study aims at answering the following two essential research questions: (1) How and to which extent can requirements views be reconstructed from existing work products, and if this is the case, (2) can these requirements views help during development? Addressing question (1), we believe requirements views can be reconstructed, although, not up to the level desired. So, the question remains, whether the proposed techniques, although sub-optimal, may have a positive effect on the overall development process of a software project. The unit of analysis is a large and long-lasting development project carried out by Logica which is described in much more detail below. Question one is assessed by typical measures used in traceability link reconstruction, i.e. recall and precision. Additional measures are used to indicate the likely effort to assess the reconstructed views, i.e. validation percentage, and coverage. Addressing question (2) is a lot more difficult, because comparable data for a fully manual reconstruction approach are lacking. In that respect, we cannot come to definite objective conclusions on the performance of the automatic approach for the task under consideration.

5.6.2 Case Study Background

The project in our case study involves a traffic monitoring system (TMS), which is an important part of a traffic control and logistics system that is required to operate at its maximum capacity. The main purpose of TMS is to record the positions of vehicles in the traffic system. These recordings are used to adjust the schedules of running and planned vehicles as well as operating the necessary signalling. The TMS owners decided to outsource the development of TMS to Logica.

Initially, Logica used IBM Rational RequisitePro for managing the requirements and MIL-std-498 Department of Defence, USA [1994] for documenting their work products. The project consumed 21 man years in the past 3 years of development. In total, there are over 1200 requirements and over 700 test cases. All the traceability links between the work products were manually set. This manual effort, which is time-consuming and error-prone, is acceptable if it is done once. However, when existing requirements evolve or new requirements come in, the links can become inconsistent; old links may need to be dropped and new links may need to be added. These are examples for why tracing becomes inconsistent, and must be redone, eventually. Sometimes, the large number of changes made that the effort needed for updating the traceability links was comparable with completely resetting all the links. Having an automatic technique in place to reconstruct the inconsistent traceability links may, thus, save a lot of effort.

Furthermore, the customer was not willing, initially, to operate on the tagged documentation Logica provided along with the tool, since the customer wanted to keep control of their own documents. For managing the requirements in this particular case, Logica was forced to make separate requirements documents in which the traceability was manually set by the requirements engineers. Some of the mechanisms used for managing requirements evolution in this setting are described in Chapter 3 dealing with the same case study (see also [Lormans et al., 2004]).

This way of working had two important shortcomings. First, it made the information used for monitoring the progress of the requirements during the development process unreliable. This was mainly due to the difficulty of keeping the traceability links consistent during the evolution of the project. This increased the risks during the integration phase, such as requirements that are not implemented, or functionality that should not be implemented in the system. Second, the manual work for synchronizing the updates from the client introduced errors, and was time-consuming.

In a later stage of the project, the customer dropped the demand of ownership of all documents. Furthermore, Logica decided to reduce the number of links maintained to the most essential ones. In particular, test documentation and test descriptions were merged, thus simplifying the underlying meta-model. This reduction of possible traceability links also helped to reduce the risk of inconsistencies.

In addition to that, part of the traceability matrix was maintained within the documentation itself, instead of in a separate spreadsheet. Test documents include the unique identifiers of the requirements they cover. The documents are structured in such a way that the Doxygen[1] documentation generator can be used to produce a HTML representation of the full matrix.

In both, the initial, and the current way of working, traceability links are set manually. Our approach aims at offering partially automated tool support for this. The case study at hand offers an opportunity to investigate whether our proposed approach can be useful in practice, and whether it may reduce the effort needed for consistent traceability support. In the

[1]www.doxygen.org

Work product type	Number	Size in terms	Average terms per document
Requirements Categories	45	1168	183
Requirements	121	695	29
Test Categories	29	589	183
Test Cases	98	886	107

Table 5.2: TMS Case Study Statistics

case study, only the current way of working will be considered.

5.6.3 Available Data

In the TMS case study, we investigate the relation between requirements and test categories and between requirements and test cases. More specifically, we focus on the requirements-to-test-coverage and the requirements-test-path views.

Two main documents are provided: a System/Subsystem Specification (SSS), containing the requirements, and a Software Test Description (STD), containing the description of the test categories. Both are MS-Word documents and they are structured according to MIL-std-498 [Department of Defence, USA, 1994]. This means that traceability data is incorporated in these documents and that it is possible to obtain a reference traceability matrix from this data.

Besides the two MS-Word documents, an HTML document generated by Doxygen is available. This document is an addition to the STD, and it contains the description of the test cases. It also comprises the description of the test categories and, in some cases, also the descriptions of the requirements it refers to (see Section 5.6.2). Doxygen uses this additional information of the test categories, and, if available, the requirements to generate the HTML document. The HTML document is accompanied by an MS-Excel spreadsheet, which contains the traceability links between the requirements and the test cases. For our LSI analysis, we only extracted the test case descriptions without the additional data (as this data is sometimes missing).

Our meta-model for this case study is shown in Figure 5.3. It consists of the following work products. In the SSS, a hierarchy of requirements can be identified. The uniquely identifiable requirements are clustered according to a hierarchy, resulting in categories of requirements. Just like the individual requirements, these requirements categories have a unique numbering, so they were taken into account for analysis as well.

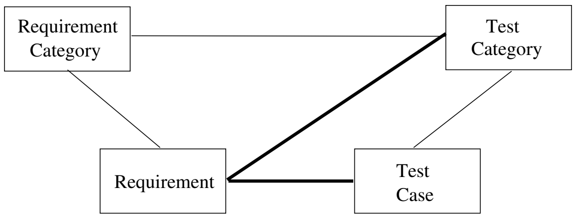


Figure 5.3: Traceability Meta-Model. The bold lines indicate the explicit traceability links available in the study.

Examples of requirements categories are general ones, such as goal and domain, as well as more specific ones, such as the use of computer resources, specific system interfaces, and safety. Each of these requirements categories has one or more uniquely identifiable requirements. The traceability between the requirements categories and requirements can be derived from the hierarchy. This traceability is not incorporated explicitly in the MS-Word documents.

For the test cases, the same hierarchy can be identified, resulting in the separate work products "test category" and "test case". Both are uniquely identifiable in the provided documentation. The main difference is that the two documents are not related directly, but only through the requirements. Thus, the individual test cases are not identifiable in the STD. In order to work out the hierarchical relations, the HTML files that include the test case descriptions and test scripts, have to be checked. They contain an identifier of a test category in the STD. However, the STD does contain the traceability links between the requirements and the test categories.

The progress of 121 requirements, distributed over 45 categories, was monitored. As these requirements are provided by MS-Word documents, some manual processing had to be done, in order to extract the relevant data from the SSS and store the processed tokens of text in the database. The requirements consist of a unique identifier and a description. Besides the requirements, the SSS document contains some context explaining certain domain knowledge for a group of requirements. This data was extracted as well and stored in the database, marking it as "context".

For the other work products, the requirements categories, test categories and test cases, the same approach for obtaining the relevant data was used, resulting in 45 requirements categories, 29 test categories and 98 test cases (see Table 5.2, above).

Logica presently maintains two types of links, as indicated by the bold lines in Figure 5.3. These links between requirements and test cases, and between requirements and test categories, are maintained in the SSS, STD,

Link source	Link target	# Reference links	# Candidate links
Requirements Categories	Requirements	121	5445
Requirements Categories	Test Categories	31	1305
Requirements	Test Categories	110	3509
Requirements	Test Cases	297	11858
Test Categories	Test Cases	122	2842

Table 5.3: Number of reference links and candidate links in the TMS case study

and spreadsheet documents. The remaining links in Figure 5.3 can either be derived from the maintained links, or from the hierarchical structure of the documents.

Table 5.3 displays, for each link type, the total number of candidate links that can be reconstructed as well as the total number of links in the reference traceability matrix. For example, there are 297 reference links derived for the "requirements – test case" link, whereas the total number of candidate links is $121 \times 98 = 11858$. The objective of our approach is to find this small number of correct reference links in the complete set of candidate links.

5.6.4 Reconstruction Approach

Reconstruction Input Parameters

The reconstruction of the traceability matrices for the different link types can be tuned in several ways. As we will see, the various link types call for slightly different parameter settings.

In all cases, we adopt a rank-k subspace of 40%. This is the size of the reduced semantic structure model produced by the singular value decomposition step of LSI. The new matrix is only 40% of the size of the original matrix, and, in LSI, it is important for filtering out unimportant details, while keeping the essential latent semantic structure intact. This step of LSI can be regarded as compressing the same information in a smaller subspace [Gross et al., 2007a], thereby generalizing the information contained.

The constant threshold is set to $c = 0.3$, i.e., two documents with a similarity below this value of c are never related. The variable threshold q is varied between 20% and 80%, indicating that the best q% of the interval between the minimum and the maximum of the similarity measures for a given document are used. The question here is which links are indeed relevant, or, in other words, where do we draw the line between interesting links and irrelevant links [Gross et al., 2007b]?

These parameters are chosen according to our experience in applying LSI (see Lormans and van Deursen [2008] for details on these parameters), and in the future, we anticipate that further “rules of thumb” for adjusting these parameters according to the problems at hand will have to be devised. In the presentation of the results in Tables 5.4–5.8 the first two columns indicate the values of c and q used.

Obtaining the Reference Matrix

The traceability data maintained manually by the software engineers at Logica were used as reference matrices in our case study. Maintaining such matrices and keeping them consistent by hand is hard and error-prone (see Section 5.6.2), so that the matrices were validated once more by Logica engineers. The existing matrix was compared with a matrix obtained automatically using our LSI-based approach. Assessing 100% of the links was considered too time-consuming. A rank-k subspace of 40%, $c = 0.3$, and $q = 20\%$ was used as inputs for this comparison. The engineers worked about 30 minutes to inspect the 29 false positives and 59 missing links issued by the tool (see Table 5.4). This resulted in resetting four missing links. Initially, they were indicated as link in the original matrix, but because REQANALYST did not reconstruct them, the engineers reassessed the links and decided to remove them from the reference traceability data. This improved the traceability matrix used as reference in our other reconstruction results.

Reconstruction Output Parameters

For each of the reconstructed matrices in Tables 5.4–5.8 seven results are shown that help to assess the usefulness of the reconstruction approach.

The set of *reconstructed links*, generated by REQANALYST, consists of *correct positives*, which are correctly reconstructed compared to the reference traceability matrix, and *false positives*, which are incorrectly reconstructed compared to the reference traceability matrix. Next, the *missing links* are shown (also known as false negatives), which are the links not reconstructed by REQANALYST, but identified as links according to the reference traceability matrix.

Finally, two commonly used metrics in the area of information retrieval are depicted; *recall* (correct positives / total reference links) and *precision* (correct positives / total reconstructed links) [Baeza-Yates and Ribeiro-Neto, 1999; Frakes and Baeza-Yates, 1992; Rijsbergen, 1979; Salton and McGill, 1986]. The ultimate goal would be to achieve a recall of 100% and a corresponding precision that is as high as possible, since in that case we only need to *eliminate* false positives. A recall below 100% which is often the case [Marcus and Maletic, 2003], inevitably means there are also false negatives (missing links). In the worst case, all candidate links need to be checked to identify these missing links, which takes much effort, but one of the goals of our approach was to reduce the manual effort needed to support consistent traceability (see Section 5.6.2).

Besides these metrics two other metrics were calculated, the *percentage of validation work* and the *coverage percentage*. For the application shown, the results of these last two columns are the most interesting.

The *percentage of validation work* refers to the effort needed to validate the reconstructed links manually compared to validating all possible candidate links manually (total reconstructed links / total candidate links). A validation percentage of 2% (see first row Table 5.4) means that the developers only need to validate 2% of all the candidate links manually.

The *coverage percentage* establishes a connection between the traceability matrix and the coverage views discussed in Section 5.4.3. The *coverage percentage* refers to the percentage of correctly covered work products compared to the total number of work products of that particular type, for example, the total number of correctly covered requirements compared to all the requirements.

5.6.5 Reconstructed Traceability Matrix Results

Given the traceability meta-model from Figure 5.3, five traceability link types are possible. First, we discuss the quality of reconstruction results for the link types Logica maintained, "requirements – test categories" and "requirements – test cases". Next, we discuss the link types we derived indirectly, "requirements categories – requirements", "test categories – test cases", and "requirements categories – test categories".

"Requirements – Test Categories"

Table 5.4 shows the results for the link reconstruction between the requirements and test categories. When we increase q we see the recall increasing and the precision decreasing as expected. The validation percentage also increases, meaning more links need to be validated. A low validation percentage is positive, as it indicates the effort needed to keep the traceability support consistent after a change, for example. In the case of $q = 20\%$, only 2% of the total candidate links need to be validated. In this example, 98% of the candidate links do not need to be validated.

However, in the case where the validation percentage is 2%, there are also 59 correct links missing compared to the reference traceability matrix. We would like to achieve a recall of 100%, so that only false positives

Link Type: Requirements – Test Categories								
c	q	Reconstructed Links		Missing Links	Recall	Precision	VP	CP
		Correct Positives	False Positives					
0.3	20%	51	29	59	0.46	0.64	2	43
0.3	40%	75	324	35	0.68	0.19	11	62
0.3	60%	82	722	28	0.75	0.10	23	68
0.3	80%	82	740	28	0.75	0.10	23	68
0.2	80%	95	1389	15	0.86	0.06	42	77
0.1	80%	107	2152	3	0.97	0.05	64	83

Table 5.4: Reconstruction results for links between requirements and test categories, where VP = Validation Percentage, and CP = Coverage Percentage.

need to be eliminated (see Section 5.6.4). Table 5.4 shows that with a constant threshold of c = 0.3, we never achieve a recall of 100%. Therefore, c was decreased to 0.2 and 0.1. With c = 0.1, a recall of almost 100% can be achieved. Unfortunately, the number of false positives increases, and, accordingly, the validation percentage. Yet, the total effort reduction is $100 - 64 = 36\%$. Antoniol *et al.* [Antoniol et al., 2002] used a similar effort estimation, which they called the Recovery Effort Index (REI). It is not clear, however, whether such measurements are realistic indicators of effort, because of lack of empirical data about a manual traceability recovery process. This will require more comparative studies in the future.

From these results it can be concluded that it is very hard to recover the last 10–15 missing links with the approach presented, and realize a recall of 100%. It is an open question whether there are textual revisions to the documents conceivable (such as an annotation mechanism, or more consistent wording of the requirements) that would enable automatic recovery.

The final column, the coverage percentage, increases as the recall increases. This is expected behaviour as it uses the correct positives as input and ignores the false positives. As the recall approaches 100%, the coverage percentage will get closer to the coverage that is obtained from the reference matrix. In the TMS case study, 85% of the requirements are covered by test categories. The missing links cause the coverage percentage to be 83% instead of 85%, as expected.

"Requirements – Test Cases"

Table 5.5 shows the results for the links between requirements and test cases. The results are of lower quality compared to the links between re-

Link Type: Requirements – Test Cases								
c	*q*	Reconstructed Links		Missing Links	Recall	Precision	VP	CP
		Correct Positives	False Positives					
0.3	20%	66	419	231	0.22	0.14	4	26
0.3	40%	141	2254	156	0.48	0.06	20	45
0.3	60%	186	3938	111	0.63	0.05	35	53
0.3	80%	186	3967	111	0.63	0.05	35	53
0.2	80%	223	6265	74	0.75	0.03	55	67
0.1	80%	260	8508	37	0.88	0.03	74	74
0.05	80%	265	8682	32	0.89	0.03	75	74
0.05	90%	276	10030	21	0.92	0.03	87	74

Table 5.5: Reconstruction results for links between requirements and test cases, where VP = Validation Percentage, and CP = Coverage Percentage.

quirements and test categories: For every value of the variable threshold q, the recall and precision are lower in this case. In order to get a reasonable recall, we need to decrease the constant threshold to $c = 0.05$. Even then, the recall is not 100%: again, we are not able to recover the final 20–30 missing links, which, are indicated as traceability links in the reference matrix.

This result has consequences for the applicability of this link relation. Looking at the validation percentage, it can be observed that 87% of all candidate links need to be validated. This means that many false positives have to be eliminated and almost all (87%) of the links must be checked manually. Somehow, there seems to be a mismatch between the requirements and the test cases.

The coverage of requirements in test cases also confirms this mismatch. The coverage percentage is 79% in the reference traceability matrix. Our result approaches that value, as expected. But, comparable to the previous case, some requirements seem to be hard to link to test cases as indicated by the difference between our value of 74% and the reference value of 79%.

A way to improve the results can be by incorporating the additional information of the test categories and requirements in the LSI analysis. This was not done, since this information is missing for some of the test cases (see Section 5.6.3). By adding this information, the identifiers of the test categories and requirements can be included in the LSI analysis, causing the similarity value to increase. The test categories did contain the unique identifiers of the requirements in their descriptions. This is probably one of the reasons why the results for the links between the requirements and test cases is lower. It also demonstrates the importance to include the iden-

Link Type: Requirements Categories – Requirements								
c	q	Reconstructed Links		Missing Links	Recall	Precision	VP	CP
		Correct Positives	False Positives					
0.3	20%	91	32	30	0.75	0.74	2	75
0.3	40%	113	313	8	0.93	0.27	8	93
0.3	50%	118	699	3	0.98	0.14	15	98
0.3	60%	119	1300	2	0.98	0.08	26	98
0.3	80%	119	1754	2	0.98	0.06	34	98

Table 5.6: Reconstruction results for links between requirements categories and requirements, where VP = Validation Percentage, and CP = Coverage Percentage.

tifiers in the LSI analysis.

"Requirements Categories – Requirements"

As discussed in Section 5.6.3, the System/Subsystem Specification (SSS) consists of a hierarchy of requirements. The higher level structure of requirements is called requirements categories. We investigated whether this containment relation can be identified using the link reconstruction approach presented.

Table 5.6 shows the results for the links between the requirements categories and requirements. These results are promising. Except for the three missing links, we already realize a recall of almost 100% with $q = 50\%$. None of the previous results has shown such high quality.

This result can be explained by the fact that a requirements category consists of one or more requirements *plus* some extra context. So, the requirements descriptions can literally be found in the description of the requirements category. The extra context is, in most cases, a general description of the requirements category. Our reconstruction approach benefits directly from the fact that a requirements category contains the individual requirement descriptions.

When doing a qualitative analysis on the three missing links we find a plausible explanation for the fact that they are not reconstructed. The two links we could not reconstruct are cancelled and they have no text describing the requirements except for the statement "cancelled".

As a consequence from the current configuration, the effort needed to validate the links is low. Besides that, the coverage is almost 100%. This means that all the requirements are covered by a requirements category. The reference matrix also shows that all the requirements are covered by

Link Type: Test Categories – Test Cases								
c	q	Reconstructed Links		Missing Links	Recall	Precision	VP	CP
		Correct Positives	False Positives					
0.3	20%	43	73	79	0.35	0.37	4	62
0.3	40%	62	324	60	0.51	0.16	14	62
0.3	60%	71	755	51	0.58	0.09	29	66
0.3	80%	71	867	51	0.58	0.08	33	66
0.2	80%	101	1512	21	0.83	0.06	57	83
0.1	80%	105	1682	17	0.86	0.06	63	83
0.05	80%	105	1682	17	0.86	0.06	63	83

Table 5.7: Reconstruction results for links between test categories and test cases, where VP = Validation Percentage, and CP = Coverage Percentage.

a requirements category. Our reconstruction results confirm this (see the last column of Table 5.6).

"Test Categories – Test Cases"

The same analysis that was done for the requirements categories and requirements was also carried out for the test categories and test cases. The major difference with the requirements hierarchy is that the test categories do not contain the test cases. The test categories are described in the Software Test Description (STD) and the test cases are described in the generated HTML document. There are no reference links maintained by Logica for this relation, so these links had to be derived via the links of the requirements.

Table 5.7 depicts the results of the link between the test categories and test cases. The results are comparable with the results of the reconstruction between the requirements and test cases. Again, it is difficult to realize a recall of 100%, so that the constant threshold must be decreased, which leads to almost 20 links not being recovered by the tool.

With a recall value of 86% already 63% of all candidate links need to be validated. If the aim is to achieve a recall of 100%, probably all candidate links need to be validated. This makes the effort reduction for this reconstruction minimal. In the future, we will have to find ways to increase recall without sacrificing precision.

The coverage of the reference matrix is 83%. A recall of 86%, realises a coverage of 83% which is equal to the coverage value of the reference matrix. This can be explained by the definition of the coverage metric. The coverage metric takes into account individual requirements. It checks if a test category is covered in the other work product, that is, the test cases. Thus, it only needs one consistent link to a test case to be set as covered. Still, a test category can have multiple links to multiple test cases. If one of these "extra" links is not reconstructed, this does not influence the coverage metric. Again, this metric only needs one consistent traceability link.

"Requirements Categories – Test Categories"

We expected the results of the link between the requirements categories and test categories to be comparable with, or even better than the relation between the requirements and the test categories. This has the following reasons. First, the level of granularity should match better. Earlier results show that if there is a mismatch in the level of granularity, the reconstruction results of LSI will decrease [Lormans and van Deursen, 2006, 2008]. Second, the requirements categories contain more text, so the vector representation of the requirements categories devised during latent semantic analysis is expected to contain more terms than the one for the requirements.

Table 5.8 shows the results of this link type. The results are indeed comparable with the results depicted in Table 5.4, but the results are not better. Thus, the clustering of the requirements into categories does not imply an improvement of the results. In other words, the "richer' vector representation of the requirements categories (because of the larger text size), does not influence the vector representation of the requirements in a positive way, compared to the vector representation of the test categories. The vector representations of the requirements and the requirements categories are comparable, causing the similarity measure to be comparable.

5.6.6 From Traceability Matrices to Requirements Views

The previous section presented the reconstruction results of the different traceability link types. The generated views were used to fill in the last two columns of the Tables 5.4, 5.5, 5.6, 5.7, and 5.8. The other metrics such as recall and precision are not relevant for the users of REQANALYST, and, thus, will not be depicted in a requirements view. Each view can be tailored to the needs of the users.

Figure 5.4 depicts an example of a coverage view. This view shows the number of requirements that are not covered in the test categories. In this case, 58 requirements are not covered and this results in a coverage of 52%. This view also lists each requirement that is not covered. The user can scroll this list and take the appropriate action.

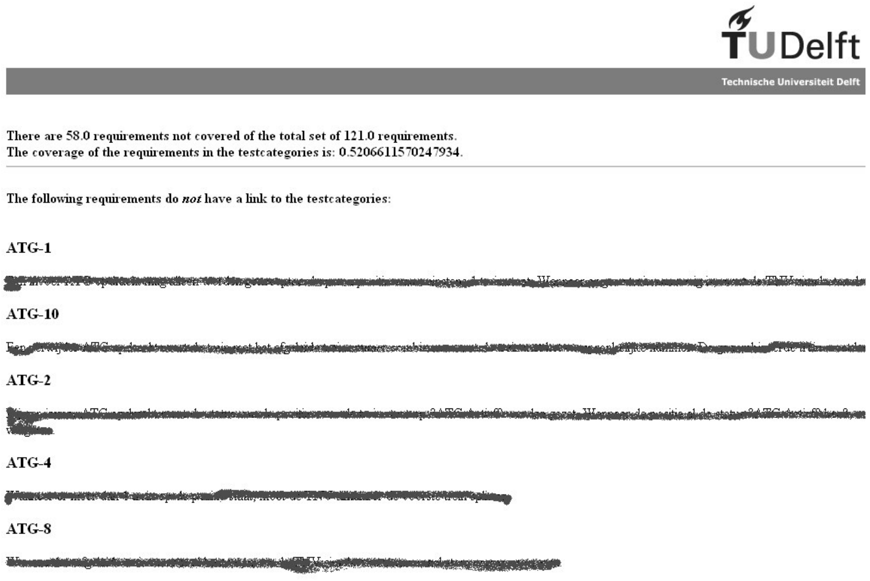
Link Type: Requirements Categories – Test Categories								
c	q	Reconstructed Links		Missing Links	Recall	Precision	VP	CP
		Correct Positives	False Positives					
0.3	20%	15	17	16	0.48	0.47	2	52
0.3	40%	17	85	14	0.55	0.17	8	59
0.3	60%	20	212	11	0.65	0.09	18	69
0.3	80%	21	224	10	0.68	0.09	19	72
0.2	80%	27	564	4	0.87	0.05	45	90
0.1	80%	31	795	0	1.0	0.04	63	90

Table 5.8: Reconstruction results for links between requirements categories and test categories, where VP = Validation Percentage, and CP = Coverage Percentage.

The views can use the automatically generated traceability links or the reference traceability matrices stored in the database. Finally, a validated matrix can be stored in the database as well. This validated matrix is then the preferred option for generating the views. To create a validated traceability matrix, all the reconstructed links are listed. The expert can review the complete list of reconstructed links and confirm or decline each candidate link. The links that are confirmed form the validated traceability matrix.

In order to create the Life-Cycle Path views, we can either use the reconstructed traceability data, or the reference traceability data. Figure 5.5 shows an example of a life-cycle path view, in which the requirements categories assume a leading role. We have made the identifier unreadable for confidentiality reasons. Figure 5.5 only shows a subset of 4 requirements categories. As can be seen, each requirements category results in 3 or more requirements. The last requirements category even results in 30 requirements. Next, the several requirements are again captured in one or more test categories. Note that in this case, the *traceability growth* is less than 1 (more artefacts on the lower level, than on the higher level). The first 3 requirements are captured in 1 test category, and the 30 requirements are captured by only 5 test categories. Finally, the traceability growth between the test categories and the test cases is greater than 1 (more artefacts on the lower level than on the higher level). The 5 test categories are covered by 27 test cases, and the 1 test category is covered by 3 test cases. The first two test categories do not have test cases related to them.

Finally, we are not able to show an example of a status view. In this case study, the status attributes are not provided in the documentation. So, we cannot show whether a requirement is approved, or whether, a test case is



There are 58.0 requirements not covered of the total set of 121.0 requirements.
The coverage of the requirements in the testcategories is: 0.5206611570247934.

The following requirements do *not* have a link to the testcategories:

ATG-1

[redacted]

ATG-10

[redacted]

ATG-2

[redacted]

ATG-4

[redacted]

ATG-8

[redacted]

Figure 5.4: Reconstructed coverage view. Company sensitive details have been made illegible on purpose.

executed and the system passed the test. Future case studies should provide this information. If status attributes are maintained, this additional information can easily be incorporated in a life-cycle path view.

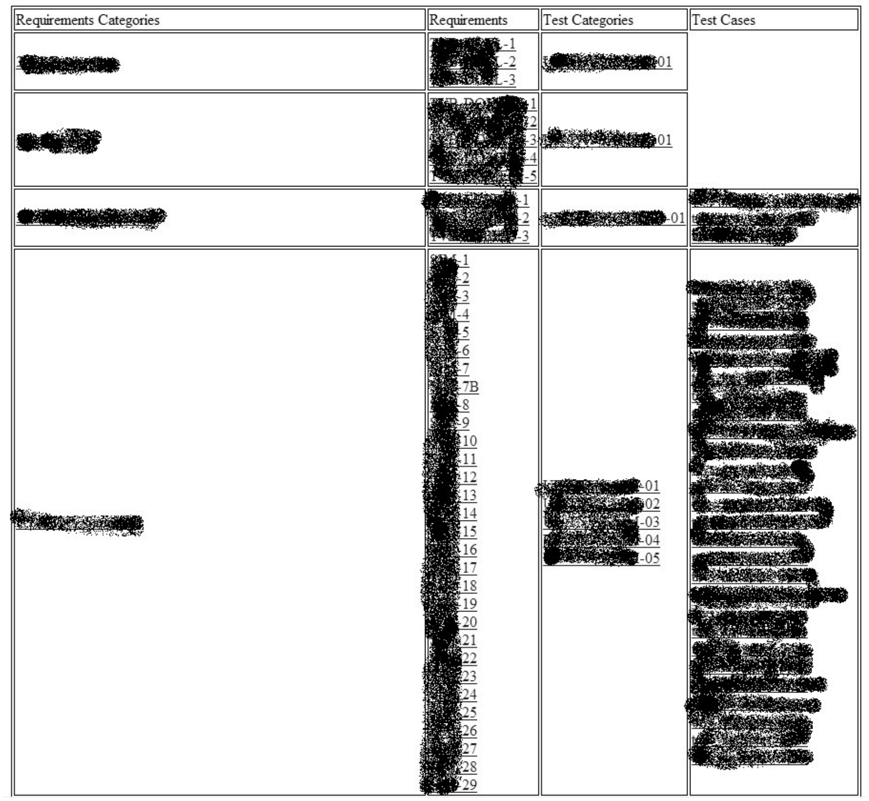
5.7 Discussion

Quality of the Reconstructed Links

The Logica case study, demonstrates that the results for the various link types differ:

- Linking requirements to test *categories* worked out reasonably well. This is an important link type, maintained manually by Logica.

- Linking requirements to individual test *cases* was harder: apparently the test case descriptions are too short and too specific to link them easily to requirements prose.



Requirements Categories	Requirements	Test Categories	Test Cases
[redacted]	[redacted] L-1 L-2 L-3	[redacted] 01	
[redacted]	[redacted] 1 2 3 4 -5	[redacted] 01	
[redacted]	[redacted] -1 -2 -3	[redacted] -01	[redacted]
[redacted]	[redacted] 1 2 3 4 5 6 7 7B 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29	[redacted] -01 02 -03 -04 -05	[redacted]

Figure 5.5: Reconstructed life-cycle path view. Company sensitive details have been made illegible on purpose.

- Linking requirements to their *requirements* category worked out very well, thanks to the fact that the requirements text was included in the category description.

Consistent Traceability Support

Our analysis identified several small inconsistencies. The traceability data incorporated in the SSS and the traceability data maintained in MS-Excel show different links compared to the content of the descriptions. For example, a requirement that was cancelled, was still included in the traceability data. The manual synchronization of these work products is, apparently, error-prone. REQANALYST can identify these inconsistencies, so that the developer can correct them. In this way, maintaining consistent traceability support becomes easier.

Requirements Views

Although more views can be defined in REQANALYST, the current views already got positive feedback from the developers at Logica. Our views increase developers' insights in the system and they improve the possibilities to review and validate the requirements systematically. Individual requirements can be inspected with respect to their coverage and their role within the system, using the life-cycle paths. Therefore, not all possible related documents need to be checked completely, reducing validation effort.

Currently, the number of views that can be generated using Doxygen is limited. The hyper links Doxygen is able to generate from its input files are bound to the information that is captured in those files. Our approach is more flexible. With the reconstructed traceability data we can generate the same and additional views compared to the Doxygen approach. So, our approach extends the current way of working at Logica.

An issue is the fact that our views greatly depend on REQANALYST's traceability support (as discussed above). Once the traceability is consistent, monitoring the progress of the requirements is improved by the requirements views proposed.

Effort Reduction

It is difficult to estimate whether and to which extent REQANALYST really reduces the effort needed for keeping the traceability support consistent. Is the 35% effort reduction reasonable? In our case, we did a first-time reconstruction and one increment (the validation session) [De Lucia et al., 2006b,a, 2007]. Following increments can take into account the validated reference traceability matrix. So, false positives that are already discarded from a previous reconstruction, and that do not relate to a change, are ignored. We expect that this will, again, reduce the effort for doing a next update. Only a small number of links, the links that are concerned with the changes, need to be validated. Initially, our reference traceability data was updated manually after the validation session, together with the expert. In order to come to final conclusions, in the future, we will have to pay more attention to how people are really constructing traceability links manually, and compare that to the performance of our automatic method.

Quality of the Documentation

Our validation session also improved the quality of the content of the work products. Normally, the specifications are reviewed by individual persons after a change. In our validation session, we inspected the false positives and missing links. Assessing the links, implied reviewing the descriptions of the related work products. This also led to more harmonized descriptions in the documentation. It is worth investigating what the documentation requirements are in order to enable full automated traceability with a 100% recall. If projects could improve their documentation, for example along the lines proposed in [De Lucia et al., 2006c], and that would enable fully automated traceability reconstruction, the benefits for practice would increase considerably.

Reconstruction Technology

The case study shows that in order to get a high recall, we have to live with a rather low precision – figures which are consistent with earlier studies [Lormans and van Deursen, 2006; De Lucia et al., 2006b]. This raises the question whether the information retrieval approach used, latent semantic indexing, can be further refined. Future work is needed to determine whether there are specific characteristics of the requirements specification domain that can help to obtain better results. For example, the hierarchical nature of requirements documents may offer further clues for reconstructing links.

In addition to that, the specific link *selection* approach could be further refined. Presently, we made use of our two-dimensional link selection strategy as described in our earlier work [Lormans and van Deursen, 2006, 2008], since in a set of separate case studies this strategy performed best. It may be worthwhile to investigate alternatives to this approach, possibly differentiating between various link types.

Generalizing the Findings

Naturally, many of the details in the case study are specific for the setting at Logica. Additional case studies are needed to determine to what extent our results can be truly generalized.

To that end, we have conducted initial experiments in a different industrial development project, this time in the electronics domain. In this case study, the meta-model is more complex, and the total set of documents is larger. Yet we can easily see the counterparts for the requirements and their categories, as well as the test cases and their categories. The initial results of these case studies yield traceability matrices and requirements views that are comparable in quality to the results from the Logica case.

Threats to Validity

We conclude our discussion with a brief analysis of potential threats to validity of the case study findings, conforming with [Yin, 2003a,b].

A first concern to discuss is *construct validity*, which deals with the question whether the type of observations made can actually help in answering the case study's questions. The risks of subjective observations has been eliminated by the use of the REQANALYST tool suite, which automatically produces the data in the tables as discussed in Section 5.6. Obtaining an accurate reference matrix is perhaps the most subjective element of the case study, since the tool findings resulted in discussion on the correctness of the reference matrices produced. The process to carefully obtain this matrix was described in Section 5.6. A final issue related to construct validity is whether "validation percentage" is a reasonable measure for effort (reduction) – this question was discussed earlier in this section as well. In all cases, we actively involved various people from Logica in the case study, in order to minimize the risk of bias and subjective findings.

Since our case study is exploratory in nature, there are no threats to *internal validity*. With respect to external validity, we refer to the observations made above in the discussion on generalizing our findings.

Last but not least, *repeatability* ("reliability" in terms of Yin [2003a]) is affected by the closed nature of an industrial case study like ours. Thus, while all data have been carefully collected and are indeed available, full repeatability is only possible within Logica. This is, in fact, important for Logica as well, since they are interested in conducting more studies like this one.

Revisiting the Case Study Questions

Before performing the case study, we had a few anticipations and expectations towards the likely outcome of a project like the one described here. One question was not initially related to the actual case study, and more of a general nature: which requirements views are needed in practice? According to the answers obtained from industrial partners, we concentrated on coverage views, life-cycle views, and status views. In particular, the first group, coverage views, is gaining importance, in many software domains, simply through the fact, that engineers want to assess the likely effect of a change in requirements on all other work products. For some domains, the automotive domain for example, it will be compulsory in the future to provide such traceability views for certification.

Other questions, more fundamental to the case study performed, dealt with the how and the extent to which the traceability views in a system can be reverse-engineered from the existing work products. We have demonstrated the "how" sufficiently through application of LSI in our proposed MAREV method and its associated tool REQANALYST. LSI is capable to generate traceability links between documents that share the same inherent semantic concepts. It is quite robust with respect to the type and structure of the documents provided. Our case study is, therefore, successful in demonstrating the application of our method and tool in such a reconstruction context.

The question of the extent to which REQANALYST can reconstruct links correctly cannot be answered sufficiently in a single case study. We have seen that LSI can reconstruct, sometimes more, sometimes less traceability links for the required views. This depends on the parameters used, leading to high recall and low precision, or low recall, with high precision. The actual question to be answered here is whether and to which extent missed links or many false positives are acceptable, and that depends on the quality of the reference matrix provided. The reference matrix is typically provided by the developers of a system as a result of some tedious manual process, and in other projects, we have observed that, sometimes, developers cannot agree on the right links, or they simply forgot to define links. The extent to which requirements views are reconstructed correctly is therefore still an open question that must be answered empirically through a number of similar case studies with thorough verification of the reference matrix, and this leads us the next questions asked earlier in this chapter: can the approach be used to reconstruct traceability views, and can the reconstructed views help in real software development. The first question we answer with a definitive yes, but engineers have to decide whether low quality of the outcome is a serious hindrance for its application. Industry is often quite pragmatic in the application of automated tools: any little tool support is better than nothing, and only looking at and assessing automatically generated views might be a lot easier than creating them from scratch. At least the tool is capable of generating the most obvious links for views that are easy to establish. However, an aftertaste remains. That is the number of missed links. At the moment, there is no way to identify missed links without visiting all links, i.e. if there is no reference matrix. Therefore, we cannot claim our method is useful for software engineers as it is, nor can we say how much, effort it can save, if any. We do not have conclusive data on the effort of reconstructing views manually. What we can foresee as future research, however, is an extended iterative reconstruction method in which the recall is increased gradually, generating many false positives, which can be filtered through a comparison with false positive links dismissed earlier. This may lead to a more precise reconstruction for which only a few new links would have to be assessed per iteration. That way, the method could bootstrap its own reference matrix and extend that on the way.

Another improvement could come from including a feedback mechanism similar to the one described in [De Lucia et al., 2006a; Huffman Hayes et al., 2006].

5.8 Contributions and Future Work

In this chapter, we have studied the reverse engineering of requirements views from software development work products, in the context of an industrial outsourcing project. We consider the following as our key contributions:

- The identification, through a questionnaire among practitioners, of three relevant requirements views: coverage views, life-cycle path views, and status views.
- An approach to reconstruct these requirements views from software development work products, supported by our REQANALYST tool suite;
- The application of our approach to an ongoing project at Logica, illustrating
  1. how the software development process steers the reconstruction process and determines the meta-model used;
  2. how the quality of the reconstructed traceability matrix can vary per link type;
  3. how the traceability matrices can be used to obtain requirements views.

Our future work will concern the following issues. First, we would like to tune our approach and come to more specific guidelines to reduce the effort needed to get a validated reference traceability matrix. Furthermore, we would like to expand the number of requirements views for more complex environments with more sophisticated meta-models. Last but not least, as mentioned in the previous section, we are presently working on an industrial case in the area of consumer electronics. This case concerns a globally distributed software development environment and a product-line, making it a very complex environment to apply our method.

5.9 Epilogue

In this chapter, we made an inventory of the views that should be relevant for industrial practitioners. Besides that, we applied our MAREV method and its newly defined views in an industrial case study.

In the next chapter, we widen our scope and project our problem to the domain of global distributed software development. We investigate the impact of requirements evolution in that domain by monitoring a case study, which implemented a tool, called SOFTFAB, for distributed software development. From that case study we derive a number of desirable features for successful software development. One of them is traceability support. We map our MAREV approach to that situation and investigate if MAREV is suitable to be applied in a global distributed software development environment.

Chapter 6

Managing Software Evolution in Distributed Software Engineering[1]

Developing a software system in collaboration with other partners, and on different geographical locations is a big challenge for organizations. In this article we first discuss a system that automates build and test processes: SOFTFAB. *This system has been successfully applied in practice in the context of multi-site projects. Then, we discuss a case where it was applied to a more challenging type of collaboration: a multi-partner development environment. Furthermore, we investigate the underlying concepts of* SOFTFAB *and use them to define a list of features for systems that support distributed software engineering.*

6.1 Introduction

Many forces make software development more and more an activity that is distributed over multiple geographical locations. Examples of such forces are acquisitions, outsourcing, mergers, time-to-market (round-the-clock development), and the (un)availability of a trained workforce [Carmel, 1999; Herbsleb and Moitra, 2001]. Additionally, software is more and more developed in collaboration with partners located at different geographical locations. For example, within Philips an internal prediction was made that within the next five years, more than 90% of its software development is done in some form of collaboration. This does not mean that all software

[1]This chapter was originally published as: Hans Spanjers, Maarten ter Huurne, Bas Graaf, Marco Lormans, Dan Bendas, and Rini van Solingen. Tool support for distributed software engineering. In *Proceedings of the IEEE International Conference on Global Software Engineering (ICGSE'06)*, pages 187–198, Florianopolis, Brazil, 2006. IEEE Computer Society

development is outsourced or done by suppliers, but that less than 10% of Philips' software will be completely developed internally.

Especially for software the trend towards engineering on different sites and with different partners is of interest, because software, compared to hardware, has negligible reproduction and transportation cost. Copying software code, reusing it, and sending it around the globe can be done in a split second and free of charge in a multi-site and multi-partner development environment. But as simple as it sounds, so difficult it is to apply this idea in a world where cultural and time differences, intellectual property interests, complex development environments, confidentiality issues, and so on, make collaboration difficult, leading to decreased development performance [Herbsleb and Mockus, 2003].

Grinter *et al.* describe a number of problems involved in multi-site development [Grinter et al., 1999]. It appears that software engineering largely builds upon informal communication. This informal communication is essential for creating understanding among developers of what is going on in their software development processes, also referred to as awareness [Chisan and Damian, 2004]. For multi-site development, a lack of such awareness leads to unexpected results from other sites, resulting in, for example, misalignment and rework [Grinter et al., 1999]. Another problem for multi-site projects is finding the right experts when they are needed. Such communication problems not only exist for (remote) multi-site development, they already exist when developers are apart as little as 30 meters [Allen, 1977].

Beside communication, also technical issues play a role. The use of different tools and data formats, for instance, makes it difficult to easily exchange information and development artefacts [Gao et al., 2002].

Different types of solutions exist for specific distributed software engineering (DSE) problems. Typically improvements can be realized in the processes, technologies or organization of software engineering [Humphrey, 1989]. Some of the difficulties related to DSE can be addressed by the use of technical infrastructures that explicitly support DSE. Such DSE support systems should provide a means to connect the software development environments of different development organizations in a way that is both *acceptable* and *convenient* for the collaborating partners. This not only involves access to the created software development products, but also access to technical software development resources, such as tools and test equipment. At the same time the different development organizations should be able to stay in control of the work products and resources located at their site.

At Philips a system that was originally developed to automate build and test processes, called SoftFab, is now applied to support multi-site development as well. Its web interface makes it particularly suited for such projects. Already 40 projects at Philips have used SOFTFAB in a multi-site setup and the results are promising. Measurements show, for instance, that projects using SOFTFAB can reduce the budget required for testing by 30-35%. Therefore, we decided to take a closer look at this DSE support system. We asked ourselves the question: "If projects are so enthusiastic about this DSE tooling, what are the underlying concepts that make it successful?" We investigate the applicability of SOFTFAB to other types of collaboration that involve multiple partners, and take a closer look at it to find out the reasons for its success, with the intention to formulate them as features a DSE system should provide.

The remainder of this chapter is organized as follows. In Section 6.2 we discuss related work. Then, in Section 3 we provide an overview of SOFTFAB, discussing the main parts of the SOFTFAB infrastructure and its features that improve distributed software development. In Section 4 we present a case study describing experiences with SOFTFAB and show the benefits and shortcomings of SOFTFAB in a multi-partner environment. An extension to SOFTFAB, called SKYFAB, is introduced in Section 5 and in Section 6 we discuss the features a multi-partner DSE support system, such as SKYFAB, should have. We end this chapter with a discussion in Section 7 and some concluding remarks in Section 8.

6.2 Background and Related Work

Research in the area of global and distributed software development mainly addresses the lack of informal communication in such settings. Proposed solutions basically follow two strategies: 1) reduce the need for informal communication, or 2) ease, stimulate, and support informal communication, often by the use of Internet technologies.

Grinter *et al.* follow the first strategy by proposing an organizational solution [Grinter et al., 1999]. They define several coordination models for dividing the work across the different sites. These models use different dimensions along which to divide the work. The idea is to co-locate work by the dimensions for which coordination is most difficult, thus facilitating informal communication for the coordination along that dimension. Other types of coordination mechanisms are then required to deal with coordination along the other dimensions. Such mechanisms can be either technical or procedural (processes). Interface definitions are an example of such a mechanism. The dimensions they propose are: functional area of expertise (co-locate experts), product structure (organization follows software architecture, c.f. Conway's Law [Conway, 1968]), and process steps (every site is responsible for a (series) of development activities).

In practice typically multiple dimensions are important leading to hybrid models. In the customization model, for example, one site develops the core product and other sites add features specific for a certain customer base. This model divides work along the process steps as well as the product structure dimensions.

The distribution of work across different sites inevitably introduces the need to transfer work products from one site to the other. Hand-off points define how and when sites perform these transfers and also specify the requirements for the involved work products [Grinter et al., 1999]. Many technical solutions can be found to support the handling of these hand-off points.

Typically, industrial companies develop their own supporting infrastructure for DSE. Fujitsu, for example, has developed such a system [Gao et al., 1999, 2002]. This solution uses Internet technology to allow for remote access to software development products and resources, within their company. Their work is focused on the technical challenges and does not specifically address multi-partner DSE.

Other research work on tool support for distributed software engineering often uses standard Internet technologies or groupware technologies, such as peer-to-peer [Lanubile, 2003]. Lanubile *et al.* present a web-based support system for distributed software inspection that supports both synchronous and asynchronous communication between the inspectors [Lanubile et al., 2003].

The tool we present not only allows communication by sharing of information, but in a sense also alleviates the need for informal communication by improving awareness in an alternative way [Chisan and Damian, 2004]. Next to an organizational and technological perspective, DSE problems can also be addressed from a process-based perspective, i.e., by deploying software engineering processes that explicitly support DSE. The CMMI, for instance, addresses some DSE problems in the process areas: Supplier Agreement Management and Integrated Supplier Management [Chrissis et al., 2003].

6.3 SOFTFAB and SKYFAB

6.3.1 SOFTFAB: A DSE Infrastructure for Automated Building and Testing

At Philips a software infrastructure is used to automate testing and building. This infrastructure, which is called SOFTFAB, enables projects to automate the build and test process, and control them remotely. SOFTFAB has been applied already in more than 40 projects. Figure 6.1 shows the SOFTFAB architecture. We call a SOFTFAB installation a "factory"; each factory consists of the following major parts:

1. A Control Center (CC) for managing and controlling the factory
2. One or more Factory PC's (FPC's) capable of performing one or more tasks
3. A network for connecting the Control Center and all the Factory PC's

The Control Center is a central server that manages all the tasks involved in the test and build process. A task can be anything as long as it is finite and produces a result that can be interpreted by the Control Center. Tasks typically compile source code, execute tests, or transfer files. All defined tasks are stored in a database. SOFTFAB users can interact with SOFTFAB via a web interface. This interface allows users to define the properties of tasks (e.g., location of input files), cluster tasks that are often used together and schedule jobs for execution. Jobs execute a single task or a group of tasks. The web interface makes it possible to control these processes remotely in multi-site environments. The results of build and test tasks are also available via the web interface, giving all sites access to reports and log files. Role-based access rights ensure that selected users can manage or operate the SOFTFAB, while others are only allowed to track the status of jobs.

Figure 6.2 shows the execution queue in the main view of the SOFTFAB web interface. It contains a list of recent jobs that are either waiting, currently in execution, or finalized. Jobs are composed from individual tasks, which are simple, atomic activities, performed on Factory PC's. The progress of a job can be tracked by its tasks in the "status" column of the execution queue, where individual tasks are represented as vertical bars. A colouring scheme is used to indicate the status of a job. As tasks are completed, their colour changes from white (waiting), via blue (executing), to green (complete success), orange (success with warnings) or red (failure). Failure of a task can prevent other tasks from executing, for example if a build fails then there is nothing to test.

Each executed job is stored together with all its configuration parameters, task results and reports, which can be inspected later. Figure 6.3 shows this job view for a specific job. It shows involved tasks, inputs, and links to the generated reports. This view represents a single line (job) from the job list in Figure 6.2 and can be opened by simply clicking that line.

A Factory PC is capable of performing one or more tasks. As such, it can be seen as a software development resource offering a specific set of capabilities. The Factory PC's in a factory are connected via a network to the Control Center for exchanging information. Factory PC's are connected

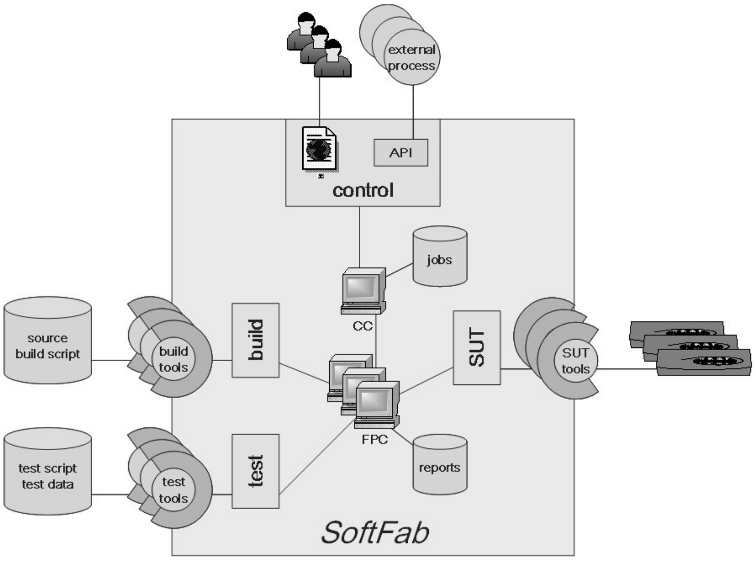
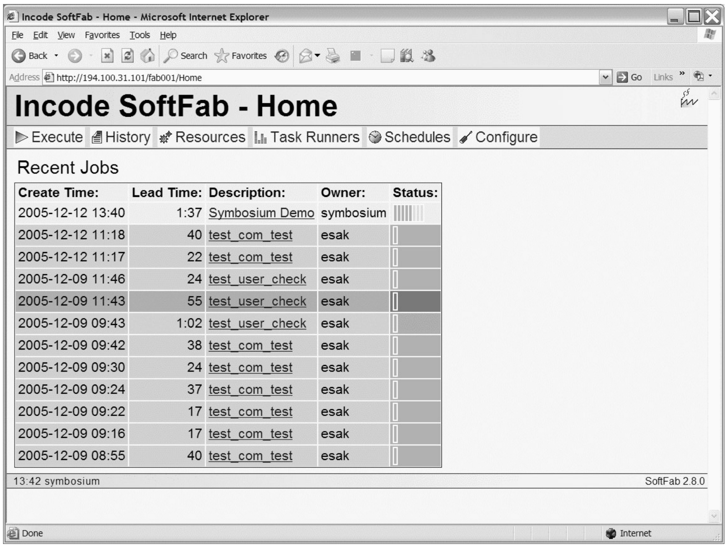


Figure 6.1: SOFTFAB Architecture


Incode SoftFab - Home - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://194.100.31.101/fab001/Home
Go Links
Execute History Resources Task Runners Schedules Configure
Done
Internet

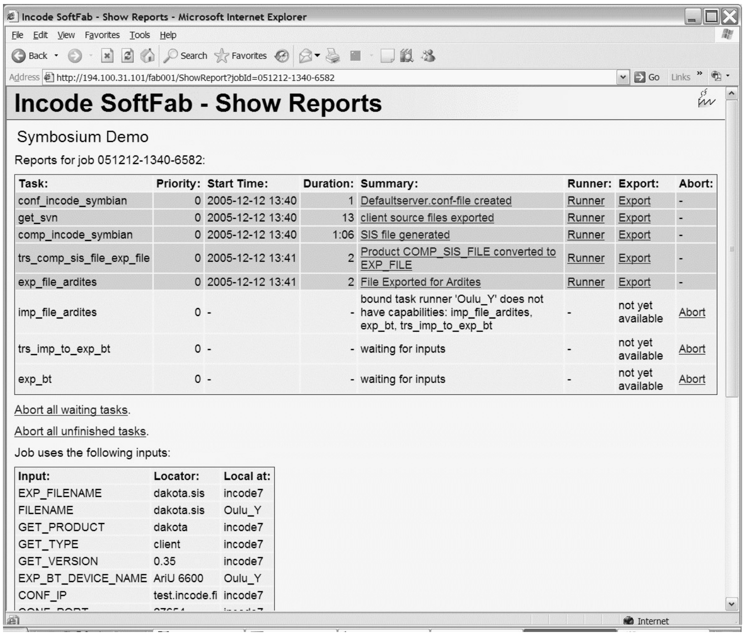
Incode SoftFab - Home

Recent Jobs

Create Time:	Lead Time:	Description:	Owner:	Status:
2005-12-12 13:40	1:37	Symbosium Demo	symbosium	
2005-12-12 11:18	40	test_com_test	esak	
2005-12-12 11:17	22	test_com_test	esak	
2005-12-09 11:46	24	test_user_check	esak	
2005-12-09 11:43	55	test_user_check	esak	
2005-12-09 09:43	1:02	test_user_check	esak	
2005-12-09 09:42	38	test_com_test	esak	
2005-12-09 09:30	24	test_com_test	esak	
2005-12-09 09:24	37	test_com_test	esak	
2005-12-09 09:22	17	test_com_test	esak	
2005-12-09 09:16	17	test_com_test	esak	
2005-12-09 08:55	40	test_com_test	esak	

13:42 symbosium SoftFab 2.8.0

Figure 6.2: Main View of SOFTFAB



Incode SoftFab - Show Reports - Microsoft Internet Explorer

http://194.100.31.101/fab001/ShowReport?jobId=051212-1340-6582

Incode SoftFab - Show Reports

Symbosium Demo

Reports for job 051212-1340-6582:

Task:	Priority:	Start Time:	Duration:	Summary:	Runner:	Export:	Abort:
conf_incode_symbian	0	2005-12-12 13:40	1	Defaultserver.conf-file created	Runner	Export	-
get_svn	0	2005-12-12 13:40	13	client source files exported	Runner	Export	-
comp_incode_symbian	0	2005-12-12 13:40	1:06	SIS file generated	Runner	Export	-
trs_comp_sis_file_exp_file	0	2005-12-12 13:41	2	Product COMP_SIS_FILE converted to EXP_FILE	Runner	Export	-
exp_file_ardites	0	2005-12-12 13:41	2	File Exported for Ardites	Runner	Export	-
imp_file_ardites	0	-	-	bound task runner 'Oulu_Y' does not have capabilities: imp_file_ardites, exp_bt, trs_imp_to_exp_bt	-	not yet available	Abort
trs_imp_to_exp_bt	0	-	-	waiting for inputs	-	not yet available	Abort
exp_bt	0	-	-	waiting for inputs	-	not yet available	Abort

Abort all waiting tasks.

Abort all unfinished tasks.

Job uses the following inputs:

Input:	Locator:	Local at:
EXP_FILENAME	dakota.sis	incode7
FILENAME	dakota.sis	Oulu_Y
GET_PRODUCT	dakota	incode7
GET_TYPE	client	incode7
GET_VERSION	0.35	incode7
EXP_BT_DEVICE_NAME	AriU 6600	Oulu_Y
CONF_IP	test.incode.fi	incode7

Figure 6.3: Job View of SOFTFAB

via standard network protocols, making it irrelevant where they are located and straightforward to set up a distributed factory.

Via Factory PC's, the SOFTFAB infrastructure interacts with the actual test and build tools. These tools are used as plug-ins in the SOFTFAB architecture. For the integration of specific tools and test equipment (SUT) it is necessary to develop glue-ware (wrappers) to allow SOFTFAB to interact with them. Besides tools, also test equipment can be made available via Factory PC's. As such, the capabilities a Factory PC offers are determined by what software is installed on it and what hardware is connected to it. On the other hand, a task defined on the Control Center requires a certain set of capabilities. The Control Center uses these capabilities to assign tasks to appropriate Factory PC's.

Besides the web interface, a Control Center also offers a programmable interface (API) that makes it possible to automate the control of a SOFTFAB. This makes it possible to integrate SOFTFAB in existing automated processes.

The Control Center is implemented in Python and the software that runs on the Factory PC is implemented in Java. It can be deployed on various operating systems, even multiple operating systems within a single factory. Furthermore, it is based on an open architecture: wrappers allow any tool that has a programmable interface (command line, COM, SOAP, etc.) to be integrated in SOFTFAB, enabling usage of both off-the-shelf development tools and in-house developed tools. Many wrappers are already available for mainstream development tools including Make, Doxygen, and JUnit. New ones can be developed by users, making SOFTFAB an effective backbone for tool interoperability.

Features of SOFTFAB that improve distributed software engineering include:

- The execution view displays the current activities at other locations, informing about the status of work and thereby increasing awareness.

- Engineers at different sites have access to the same reports and log files through their web browsers, facilitating communication.

- Work products and resources can be accessed, regardless of location, local time, language or availability of human resources.

- The Control Center coordinates tasks involving distributed resources, such as tasks involving building and testing executed on different computers, possibly on different sites.

- An overview of build and test procedures (tasks) is available on the Control Center. All details of the procedures are contained in the wrappers. This makes implicit knowledge explicit, facilitating new employees' training and allowing procedures to be easily transferred to other project teams.

- In-depth knowledge of a task is only required for initial implementation and for maintenance of associated wrappers, not for their execution. Therefore, tasks can be run by any user, without depending on the availability of an expert.

6.3.2 SoftFab Experiences

SoftFab's possibilities for DSE support can best be illustrated by a real-life example. In Finland we conducted a case study to investigate the applicability of SoftFab in collaborations that involve multiple partners. Due to, for example, intellectual property interests and confidentiality issues such collaborations pose extra requirements to DSE support systems compared to single-company, multi-site projects. We use this case study to explain the problems that are encountered in such a collaboration, and how SoftFab is set up and used. The case study involved three partners, each on a different development site:

- The system integrator. Develops environment aware applications for smart phones.
- The COTS supplier. Sells a database management system for mobile and embedded applications.
- The testing subcontractor. A research group at a university, which is specialized in software testing.

In our case study the system integrator replaces a certain layer of one of its mobile products with the data management solution developed by the COTS supplier. The testing subcontractor provides services related to the validation of the integrated product by executing tests and measuring performance for different software configurations. This migration project uses SoftFab as infrastructure.

One SoftFab factory is distributed over the COTS supplier and system integrator to share the releases of the COTS components. Another SoftFab factory is distributed over the integrator and the testing subcontractor. The first collaboration focuses primarily on the sharing of work products. The latter also allows the integrator to use the resources of the subcontractor. In the remainder we will focus on the latter as this application is more complex.

For deploying the SoftFab infrastructure a couple of activities need to be done at every site:

- Training of employees and analyzing the existing building and testing procedures.
- Installation and configuration of the SOFTFAB software on Control Center and Factory PC's.
- Development of scripts to automate several tasks previously preformed manually (e.g., retrieving configurations, or deploying installation packages to target devices).
- Development of wrappers linking existing automated tasks to SOFTFAB.

For the collaboration between the testing subcontractor and the integrator a testing facility is built at the subcontractor's site, which is connected to the integrator's site using SOFTFAB. The SOFTFAB Control Center is situated at the integrator's site, in a demilitarized zone (DMZ) that is accessible from outside of the company's intranet, and also acts as a bridge for transferring files over the firewall. One Factory PC is available at the same location, having access to company specific assets and tools. A computer at the subcontractor is connected as a second Factory PC. Its capabilities are limited to importing files from other locations and deploying and executing applications on target devices.

Via SOFTFAB the testing subcontractor can use Factory PC's at the integrator's site to build components from arbitrary versions of the source code, which can be used to build and test any version of the system. At the other site, the integrator can remotely execute tests at the subcontractor's site, using different versions of target hardware. Without SOFTFAB this would have required the integrator to request a test from the subcontractor. Then, the subcontractor has to wait for the right version of the application to be sent, after which it can be tested and the test results returned. This illustrates that SOFTFAB can speed up the integration process significantly.

The mobile application is built at the integrator site from source code stored in their configuration management system, using their tools, licenses and scripts. The parameters for this task, such as product versions, library versions, and hard-coded constant values can be specified. After the application is built and packaged as a binary installation file, it is encrypted and exported to the subcontractor's Factory PC. In the final phase the install package is deployed on a target device via a Bluetooth link, after which the application can be executed for testing. This complete scenario can be controlled from a single (remote) location using SOFTFAB's web interface.

The testing subcontractor and the integrator used different types of tools that were easily plugged into the local SOFTFAB infrastructure by writing wrappers using their command line interface.

The benefits of this setup were especially visible during one specific experience at the subcontractor during a complete build-deploy-run cycle. This operation failed at the point where the binary should have been transferred outside from the integrator's intranet to the test equipment at the subcontractor's site. SOFTFAB's job view (see Figure 6.3) revealed that one of the engineers at the integrator's site was already re-executing some of the tasks involved in this job. So, somebody already noticed the problem and was busy fixing it. No telephone or email was required to understand the situation, to see latest progress, or to know that a solution was on the way.

This example shows that SOFTFAB increases developers' awareness, without the need for informal communication. Partners are able to understand the situation based on the information shown by SOFTFAB and take action if needed.

We have used SOFTFAB in this case study in a multi-partner collaboration. This is a different type of collaboration than for which SOFTFAB was developed until now. Obviously, the partners involved also identified some shortcomings:

- All partners have the same level of access to all resources. Some partners wanted to have more control over their own Factory PC's. This is especially important when a project has to deliver to multiple customers.

- All partners have access to all work products. In general, however, partners require more control on sharing of work products, for instance, the ability to allow sharing between Factory PC's within one partner, but to limit sharing between different partners.

- Implementation details of processes of one partner are visible for all partners. It might take several tasks to produce a particular work product. This is not relevant for other partners; only the final product is. The processes of each partner need to be encapsulated, hiding its inner workings. Each partner should be able to control which of its tasks and work products are visible from the outside for reasons of sensitivity (Intellectual Property) and clarity (abstracting from implementation details).

6.3.3 SKYFAB: A Support System for Multi-Partner DSE

It turned out that the application of SOFTFAB in a collaborative environment with multiple companies was more difficult than collaboration within

Scope	Terminology	Sharing and Collaboration
Multi-site	SoftFab	- “standard SoftFab” benefits - test equipment, tools, test cases and test data
Multi-project	Inter Factory Resource Sharing	- software licenses - equipment
Multi-partner	SkyFab	- software development processes and procedures - test equipment, tools, test cases and test data - protection of IP - respecting firewalls

Table 6.1: Collaboration scope

a single company. This suggests that there are different types of collaboration possible that might require specific support from a DSE infrastructure. Table 6.1 presents three collaboration levels and summarizes what is shared at each level. The levels are ordered according to their scope of sharing: the larger the sharing scope, the more difficult the collaboration.

A standard SoftFab setup, as discussed earlier, easily shares several resources, data, and procedures (e.g., via test scripts). Typically, every project implements its own factory. At Philips, SoftFab has also been used to share resources between projects. In such a case, each project owns a factory and the Factory PC controlling the resource is listening to the Control Centers of both projects, sharing the unique resource transparently for the end-user. The most challenging level of sharing, the collaboration between multiple partners (companies, universities, and so on), requires some additional features. A Multi-Partner DSE (MP-DSE) support system providing those features is currently being implemented as an extended version of SoftFab, and is called SkyFab.

The involvement of multiple partners makes software development more complex, for instance, when partners have their own policies on security and protection of intellectual property. To address this SkyFab will allow each partner to implement its own local factory behind a corporate firewall. A globally shared SkyFab Control Center is placed in a DMZ and is connected to the Control Centers of the local factories, thus forming a hierarchy of multiple SoftFab installations. Now one can run a job of which the tasks are executed in different partners' factories using the local available resources without breaking the partner's security rules. Figure 6.4

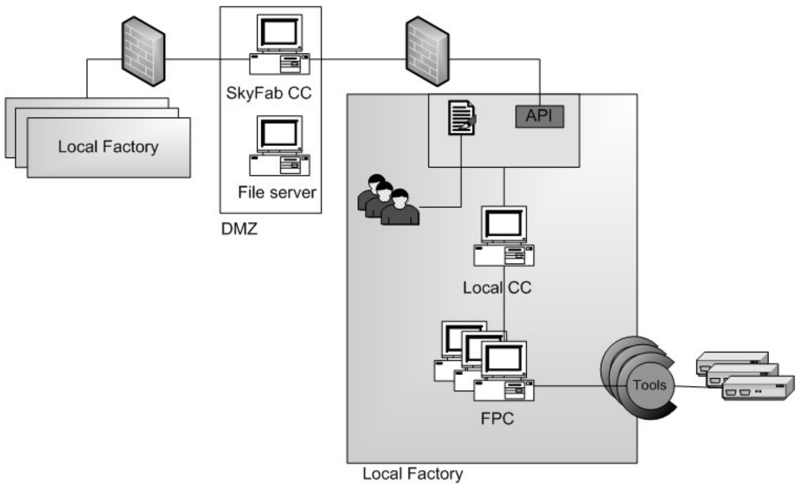


Figure 6.4: SKYFAB Architecture

shows the architecture of a SKYFAB factory. The setup of our case study can be seen as an intermediate step towards this architecture.

6.4 Features of a MP-DSE Support System

Below we list a set of desirable features for MP-DSE support systems. These features not only follow from the case study discussed above, but also from the experience of using SOFTFAB in about 40 different projects at Philips. Some of these features are supported by SOFTFAB; others are not supported by SOFTFAB, but appeared to be desirable in practice. These will be implemented in SKYFAB.

6.4.1 Work Product Sharing

When software is developed collaboratively, work products (designs, documents, test results, code, executables, etc.) need to be shared among the different development sites. In our case study, for instance, the testing subcontractor can only test the application when the executable binaries, compiled at the integrator's site, are provided. This does not mean, that everything needs to be shared, it means that sharing of work products needs to be decided upon and organized, i.e., handoff points [4] need to be defined. Sharing can be downloading or uploading of work products.

In SOFTFAB sharing of work products is already arranged both explicitly and transparently (and as extension to SOFTFAB, SKYFAB has this ability as well). On the one hand, it is possible to explicitly define tasks that have the purpose of retrieving information from, e.g., a configuration management system and deliver the result via the web interface. On the other hand, tasks can be configured to require input from the site of a partner. During task execution this input is then retrieved transparently for the user.

6.4.2 Development Resource Sharing

Each partner typically has its own development tools and equipment, which are often even different between sites of one partner. Especially for embedded software development it can be difficult and expensive to replicate test environments on different sites, which, in turn, makes it difficult to reproduce test results on the different locations. In our case study, for instance, a testing facility was only available at the testing subcontractor's site. DSE support systems must be able to deal with these different technical environments without requiring technical expansions on other sites. In a collaborative environment, especially in the case of a multi-partner collaboration, one cannot expect other partners to rigorously change or expand their technical development environment.

Therefore, in some cases, collaborating partners should be able to start and control tasks such as compiling, building, testing, code generation, static analysis, etc. at other sites. This does not mean that every task must be fully remotely accessible or controllable, but it does mean that a dedicated selection of these tasks should be externally executable. As such, the testing subcontractor was able to compile a specific version of the application to test using the integrator's build environment remotely. The configuration of the tasks and their required tooling is the responsibility of the site where a task runs. Executing a task typically produces a work product, which again can be shared.

In a multi-site SOFTFAB setup, this is supported via the access to and control over the Factory PC's in other sites. Each Factory PC declares its capabilities explicitly: it declares which tasks it is able to perform and which output it is able to produce based on which inputs. In SKYFAB a hierarchy of SOFTFAB factories will be introduced, enabling one partner to execute tasks in the local factory of another partner. This allows a partner to share its development resources, without releasing all control over them (see also Section 6.3).

6.4.3 Product and Resource Access Control

Although it is necessary to share work products and resources between partners, not everything needs to be shared. A specific partner will need to allow access to certain work products and resources. However, due to confidentiality issues, every partner needs to be in control of the accessibility of their work products and resources. Especially, when considering distributed development with different partners this is an important issue, as intellectual property or business interests, for instance, need to be protected. In our case study, for example, the source code of the product under test was not disclosed to the testing partner.

The role-based access rights mechanism in SOFTFAB currently defines three fixed roles. This is not sufficient for multi-partner collaborations, where a specific partner wants to control access to his local factory for each partner separately. In SKYFAB access rights will be more fine-grained and will be managed per individual user. SKYFAB uses a local SOFTFAB factory per partner. Each local Control Center serves as a kind of gatekeeper to the working environment at a specific site. Certain processes and work products can be shared with the other partners, while others remain private. It allows the other partners to execute operations on certain data and retrieve the results, without the need to access the data directly.

6.4.4 Heterogeneous Environment Support

Each partner has its own software development infrastructure. Some of the differences originate in the different role each partner plays in the collaboration, requiring different technical solutions. Homogenization of systems is not a solution: it would disrupt established ways of working and invalidate past investments, in addition to ignoring the fact that the diverse skills of the different partners are key to making a collaboration perform better than a single partner could. Thus, DSE tools should be based on an open architecture, allowing interaction with different systems from multiple vendors running on different platforms. This is especially important for MP-DSE.

SOFTFAB supports this by storing results in the native format of a tool. For example, test results are stored in the test report format of the specific test tool, typically a plain text, HTML or PDF. The resulting document is then accessible to other sites, for instance, via a web server. Other sites do not need to have licenses for these other tools but still are able to create and access the results. Furthermore, SOFTFAB enables a partner to execute test processes that use tools and equipment at a remote partners' site. One partner, for example, can test software in the test environment of other partners, without having the licenses for the involved tools.

6.4.5 Real-time Status Updating

Collaboration between sites and especially between partners requires a continuous insight in the status of work and work products. In a non-distributed setting this awareness is created by informal communication via email, telephone and in face-to-face meetings. DSE support systems should compensate for the lack of informal communication between remote locations and enable transparency in work carried out and the work products being produced, in order to increase developers' awareness. An example of tool-mediated awareness was presented in Section 6.4.

SOFTFAB supports this by providing on-line insight into the work carried out at other sites, showing status and result overviews of tasks carried out (see again Figure 6.2 and Figure 6.3). Depending on users' access rights (see also Section 6.3), they are able to access these overviews to find out what has been done and what the result are of the tasks executed. As developers can see for themselves what is going on, there is no need to consult other sites just to know the current status. They do not need to ask; SOFTFAB simply shows it.

Naturally, for more complex tasks, such as analyzing a difficult bug, direct communication between partners is still required. In such cases, SOFTFAB helps by providing a clearly labeled status overview of the tasks that were executed on all sites, and by providing access to all reports and log files to all developers involved. This avoids misunderstandings ("which version are we talking about?") and allows engineers to focus on the problem only.

6.4.6 Consistency and Timeliness Management

In our case study, the testing subcontractor needed to run its regression tests against the latest release of the product as well as against previous versions. As such, the subcontractor needs the correct versions of the application to be easily available.

If operations on software development data can be remotely executed, the need to physically distribute that data disappears. Sites are able to acquire work products when needed. Version checks will not be necessary, because the tested product is directly and remotely built from the configuration management system at the development site. Differences and delays due to development at different continents and time zones are overcome when a DSE support system is able to manage consistency and timeliness of work products automatically.

In SOFTFAB consistency is supported by ensuring that the actual developer or maintainer of a work product has it physically on its own site and allows sharing it with others. In case of sharing of resources, consistency and timeliness is managed by ensuring that the owning site also develops and maintains its resources. Additionally, the automation of tasks makes developers independent of the presence of people at other sites.

6.4.7 Knowledge Transfer

The knowledge and experience of developers is incorporated in the work products they produce. The same applies to automated scripts used in a DSE support system. As such, the tasks that are incorporated in a DSE support system should hide complexity from their users. When this is done correctly, the transferability of work processes and sometimes of complete projects increases. When tasks are not automated, not archived and not documented in a standardized way, transferability of work products is only feasible face-to-face by teaching and handing over. By putting the knowledge in a standardized way into scripts, users can (remotely) control and execute these work processes without the need to understand their content. As such, complexity is hidden, increasing the transferability of work.

SOFTFAB supports this by standardized scripts (wrappers) that can be executed remotely, but are maintained locally.

6.4.8 Traceability Support

In a MP-DSE environment many work products are being developed by teams that are working at various locations. In many cases only one system needs to be produced. Therefore in a MP-DSE environment, integration is especially important. To realize successful integration of the system, you need to be able to trace the work products that are being developed by the various teams. Besides that, project progress and coverage can be monitored during the project. The total control of a MP-DSE project and the quality of a MP-DSE project improves when implementing traceability support.

SOFTFAB does not explicitly implement traceability support. However, implicitly some traceability support is implemented in SOFTFAB via its hierarchy of tasks and jobs. All tasks can be traced to their corresponding job.

Besides that, SOFTFAB has implemented interfacing to known version control tooling. This way changes to the code can be managed and monitored. Finally, build settings are managed in SOFTFAB. Using these settings, each release of the software can be rebuilt using the corresponding source code and test cases.

6.5 SOFTFAB and Monitoring Requirements Evolution

In our experiment we tried to align our MAREV approach and REQANALYST tool suite with SOFTFAB. Our experience shows that both tools can work together easily. The reason for this is that both tools are able to import and export data in a structure way. In other words, they implement one of the identified processes in the RES framework (see Section 3.6.3), namely the process of interaction with stakeholders annex importing and exporting data in a structured way.

In Section 6.4.4, we have already explained that SOFTFAB supports the sharing of documentation in formats that are accessible by multiple platforms (plain text, HTML, and PDF). REQANALYST can parse these documents and import the data. Then, REQANALYST can process this data and produce its predefined views using this data. If necessary, REQANALYST can export this data to SOFTFAB.

At the time this research was carried out, a plug-in that completely integrated both tools was not available. We started to build a complete tool chain using Eclipse[1] that contains both SOFTFAB and REQANALYST, but at the time of writing this, it has not been completed yet and remains future work.

6.6 Discussion

A set of features for MP-DSE has been introduced in Section 6.4 of this chapter. The need for each individual feature differs for each application of MP-DSE. Naturally, the ability to share information is a necessary condition for any collaboration. However, many solutions offer that feature: a simple FTP-server would suffice. The other features are more specifically aimed at support for distributed software engineering.

For instance, the ability to share technical software development resources - together with work product sharing the most important feature of SOFTFAB- solves a number of practical problems often encountered in multi-site development project, such as the difficulty of replicating build and test environments on multiple sites. The ability to share both work products and resources, sets SOFTFAB apart from many other systems that can be used to support distributed software development, such as groupware systems.

Support for heterogeneous development environments is obviously only necessary in situations where the technical environments used at the different sites in a collaboration are actually different. A similar argument

[1]www.eclipse.org

holds for fine-grained control of access to work products and resources, which is primarily relevant in cases were intellectual property is an issue.

Real-time status updates and consistent and timely access to work products are features that are not absolutely necessary, but they are very useful to increase the awareness of developers at different sites.

Finally, support for transfer of work processes makes it easier to handover a project to another site or to a customer, for instance, when a product has been delivered and enters the maintenance phase of the software life-cycle. Thus, indirectly such a feature improves its maintainability.

The need for the above features potentially exists in every phase of the software life-cycle, not only for building and testing. Currently, SOFTFAB mainly supports the build and test phases of a software project. Support for other phases is constrained by one limitation: if the phase cannot be controlled remotely (and hence at least partly automated), SOFTFAB cannot deal with it. Therefore, the applicability of SOFTFAB to other development phases strongly depends on the usage of automated tools in those phases. Previous research shows that in the early phases of the life-cycle, i.e., requirements engineering and architecture development, tool support is still limited (See [Graaf et al., 2003] and Chapter 2 of this thesis). During later phases usage of automated tools is more common. As SOFTFAB builds upon remote control and access, it largely builds upon automated tooling. Therefore, it may seem that the applicability and benefit of SOFTFAB lies in later phases of the life-cycle.

However, more tools are expected to be used in earlier phases as well. For instance, if we consider the trend of model-driven development, and OMG's model-driven architecture (MDA) [OMG, 2007] in particular, we see that tool vendors develop more and more tools to be applied during architecture development. These tools automate development tasks, such as model transformation and code generation.

Also in the area of requirements engineering some software engineering activities are amenable to support by SOFTFAB. One of them is coverage analysis: determining the extent to which requirements are covered by other (downstream) work products. Tools for doing this automatically are being developed, as discussed in Chapter 4 and 5 of this thesis. By connecting such tools, a DSE support system could help to provide up-to-date status views of the requirements coverage of a software system under development. The underlying (potentially confidential) information does not have to be shown in a shared report, but it can be used for generating the coverage view. This way the actual progress of a project in terms of addressed requirements can constantly be monitored during the life-cycle.

SOFTFAB and SKYFAB support such future developments by sharing the control and results of automated software engineering tasks. However, it should be remembered that even when the features discussed in this chapter are all fully supported by a DSE support system, successful collaboration cannot be guaranteed. The success lies in the way the DSE support system is used, which is largely determined by the willingness of companies to collaborate, and their openness towards and confidence in each other.

6.7 Contributions and Future Work

DSE support is of large interest for today's industry. Software has an intrinsic power due to its relatively cheap reproduction and transportation cost. To fully benefit from this strength, DSE support systems need to be deployed, even across company borders. The SOFTFAB infrastructure discussed in this chapter is based on industrial multi-site practice. It has been applied many times to full satisfaction of the involved partners. Furthermore, we discussed SKYFAB based on the application of SOFTFAB as a MP-DSE support system.

Based on Philips' experience, using a system such as SOFTFAB to connect the development environments of partners that develop software in collaboration, while they maintain self-control, has great potentials to speed up software development. As such, we conclude that the road towards profitable, faster and more reliable software development lies in collaboration. DSE support systems are needed for that, preferably developed and built upon industrial best-practices.

We consider the following to be the key contributions of this chapter:

- We discussed SOFTFAB, an infrastructure for automating the build and test process.
- We illustrated the possibilities of SOFTFAB in a case study, revealing the strengths and the weakness of SOFTFAB in a multi-partner setting.
- We proposed an extended version of SOFTFAB, SKYFAB, that addresses the issues revealed in the multi-partner case study, and identified three levels of collaboration that are relevant for MP-DSE support systems
- We introduced and analyzed seven features of a MP-DSE support system, giving a good overview of the requirements for developing or selecting such a support system.

In future research we will expand and implement the SKYFAB concept. Currently, we are further evolving SOFTFAB into SKYFAB. In this industrial case study we implement the features we identified to overcome the difficulties we encountered when applying SoftFab as-is in a multi-partner environment. After that we intend to investigate how to extend SkyFab to support other software development phases, as discussed in Section 6.4. We are already developing an application for analyzing and monitoring requirements in a distributed software development setting.

6.8 Epilogue

This chapter has given an overview of features that are essential for successful tool support in MP-DSE projects. We mapped our Marev and its accompanying tool ReqAnalyst to this domain and learned that our Marev approach can easily be combined without spending much effort on integrating the both tools. A prerequisite is that the import and export features of both tools are mature, so data exchange is possible. Then the real integration depends on the structure of the data itself as defined in the traceability model.

In the next chapter, we finalize our research work and reconsider the research questions defined in Chapter 1.

Chapter 7

Conclusions

This chapter concludes our research project concentrating on monitoring requirements evolution and improving requirements management. We recall our research objectives and reflect on the results presented in each chapter in relation to our research objectives. Finally, we provide recommendations and discuss future work.

7.1 Recalling the Research Objectives

In this dissertation we addressed the problem of monitoring requirements evolution during the software engineering life-cycle. We defined four themes in Chapter 1: exploring, structuring, showing, and expanding the world. These four themes cover our research sub-questions also defined in Chapter 1:

RQ1 How is requirements management currently done in practice and what is the impact of requirements evolution?

RQ2 Which views do we need to support the process of monitoring requirements evolution?

RQ3 How can we automate the reconstruction of requirements traceability to support requirements evolution?

RQ4 What is the impact of global software development on the management of requirements evolution?

We investigated these themes in order to answer our main research question:

RQ0 How can we develop an automated requirements management environment to improve the management of requirements evolution?

7.2 Contributions

We consider the following to be our key contributions:

- We gave an overview of software engineering technologies used in practice for the development of embedded software (see Chapter 2).

- We listed a number of features a requirements managements system should consider, investigated how these requirements management features are tackled in an industrial case study, and identified a number of problems currently adopted requirements management methods and tools have when applied in practice (see Chapter 3).

- We proposed a framework that addresses the concerns with respect to monitoring requirements evolution such as the interaction with stakeholders, consistent processing of changes, and presentation of information using requirements views(see Chapter 3).

- We provided MAREV, a methodology for reconstructing requirements traceability and generating requirements views (see Chapter 4).

- We defined a new filter strategy for selecting traceability links from a LSI similarity matrix (see Chapter 4).

- We implemented MAREV in our REQANALYST tool suite (see Chapter 4).

- We applied our approach in several academic and industrial case studies (see Chapters 4 and 5).

- We defined a number of views, which are useful for monitoring requirements evolution(see Chapter 5).

- We studied the global software engineering domain and defined features tool support for multi-partner distributed software engineering should consider (see Chapter 6).

7.3 Discussion and Evaluation

Now that we have revisited our research objectives and listed our contributions, we will discuss each of them in a separate section, using our research objectives to guide the discussion.

7.3.1 Exploring the World (R1)

Our first research question concerning requirements management in practice, is mainly answered in Chapters 2 and 3. Our main observation from Chapter 2 is the gap between the state of the practice and the state of the art. In the embedded software domain, industry is cautious with using new technology. The main reason for this appears to be the fact that these organisations are avoiding risks. To do so, they only introduce small changes in their organisation and all technology needs to have a certain maturity; in other words, they have proven to be reliable.

In our research, we took this observation into account by minimizing the use of new technologies. In our solution we tried to use existing technologies, such as Latent Semantic Indexing (LSI), an established information retrieval method.

7.3.2 Structuring the World (R2)

For structuring the process of requirements evolution, in Chapter 3 we defined our RES framework. This framework captures all relevant processes for managing and controlling requirements evolution. Both managing and controlling the structure of requirements evolution are captured in two RES processes: Exchanging Data, and Processing Changes to that data. Both processes will be discussed separately.

The Exchange of Data This process concerns the interaction with the stakeholders. We need to exchange data and as this data is changing continuously, we need to control this process. Mainly, this is a configuration issue; a meta-model needs to be defined and mapped to the current way of working. A key problem is that each project has a different configuration. The document structure is different for most projects, complicating any form of automation. In practice this means that for most projects we need to manually configure the data that needs to be exchanged (the work products defined in the meta-model).

The result of the first step, the meta-model, should be communicated with and agreed upon by all stakeholders. Next, the current project setting needs to be transformed to the input needed for an automated analysis approach. The concepts in the meta-model need to be identified in the current document structure. Once that is done a pre-processor can automatically process the documents. Thus, the first two steps of our MAREV method are hard to automate and will, in most cases, remain a manual activity.

The identification of the concepts that need to be traced can be automated by using project-specific scripts. For running projects the configuration has already been established and these scripts need to be developed from scratch. This typically is a one-time investment that can be used throughout a project's life-time. Projects that start from scratch can take into account these MAREV requirements when defining their configuration. When doing this, the project can benefit from the MAREV possibilities during the rest of its life-time.

Processing Changes This process concerns automating traceability support when changes occur. We have learned that it is difficult to keep a project configuration consistent, especially regarding traceability. Our MAREV approach aims at providing a solution for this problem, by generating candidate traceability links automatically. With respect to this link reconstruction two conclusions can be drawn.

The first conclusion relates to link types. The performance of LSI for the different link types differs systematically. The links between requirements and test cases, in general, performed better than the links between requirements and design. One of the reasons LSI performs worse for "requirement – design" relations is the fact that for designs many diagrams are used for documentation (see Chapter 4). These diagrams are often hardly described using text. Information retrieval techniques are less suitable for traceability reconstruction in this case.

The second conclusion relates to the link selection strategies. With respect to the link selection strategies it is not always easy to determine when a strategy performs better. This very much depends on the application objectives. In the case studies we showed the results of applying LSI using different parameter values and compared the available link selection strategies. We have also seen that the link selection strategies cannot always avoid generating more links than desirable, for example, generating false positives.

One lesson learned is the fact that people make mistakes when updating information. In all case studies, we found inconsistencies during our analysis. The traceability data incorporated in the documents and the traceability data maintained separately (if maintained at all) show differences compared to the content of the descriptions. The manual synchronization is apparently error-prone. REQANALYST can identify these inconsistencies, after which the developer can correct them. This way, maintaining consistent traceability support becomes easier.

7.3.3 Showing the World (R3)

Our final process defined in our RES framework, was presenting the data. This process is directly related to our third research question. If we want to manage and control requirements evolution we need a way to monitor it.

In our approach we defined a number of views to realize this.

Although the number of views in REQANALYST can be extended, the views incorporated already increase developers' insights in the system. Our stakeholders (developers, project managers, etc...) are satisfied with the views that can be generated. They show improvements with respect to other requirement management tools. Our views improve the possibilities to systematically review and validate the requirements; individual requirements can be inspected with respect to their coverage and their role within the system.

An issue is the fact that our views greatly depend on REQANALYST's traceability support (as discussed above). Once the traceability is consistent, the progress of requirements can easily be monitored with the defined requirements views. Our incremental approach in MAREV helps managing the evolution of the reconstructed and validated traceability links.

7.3.4 Expanding the World (R4)

In our last research sub-question we considered global software engineering and investigated the impact for monitoring requirements evolution. In Chapter 6 we showed that global software development requires some additional features for tooling or requires more attention to specific features compared to "normal" software development.

One of the features, which had our special interest, was traceability support. In SOFTFAB this feature was not implemented. However, file transfer is implemented in SOFTFAB and so the files could easily be shared and imported in REQANALYST. Therefore, our MAREV approach can be used in a global software development project. In such a project SOFTFAB and REQANALYST can be combined and complement each other without spending much effort on integrating the both tools.

7.4 Recommendations and Future Work

In this thesis we discussed the monitoring of requirements evolution. However, not all discussed issues have been solved. In each chapter, we raised a number of questions that need to be investigated further. In this section we recall these open issues and discuss the possibilities for future research.

The first issue we need to discuss is the use of LSI and preparing the data for LSI analysis. The pre-processing of the provided documents requires quite some manual work. The first step involves mapping and transforming the documents to the traceability meta-model defined for the project.

In an ideal situation, the document structure is already in the format needed for pre-processing and executing the LSI analysis. If not, the interaction process as described in the RES framework should be followed. Then the manual activities only need to be executed once. In the current situation, changes to the documents are done in the original document format. Next, they still need to be process in transformed documents. This whole process of preparing the documentation for LSI analysis needs to be optimized.

A second issue is that the current processing of large data sets via LSI requires quite some resources (time and computing power). For (near) real-time analysis of the requirements it is not acceptable that the analysis takes more than 10 minutes. In addition to that, in our current implementation of REQANALYST the traceability reconstruction is done during a web session. To make the tool more user-friendly and scalable, this processing needs to be done earlier or faster. Future research should provide these technological improvements.

In Chapter 6, we discussed a tool called SOFTFAB, which currently does not support the requirements monitoring as discussed in Chapters 3, 4, and 5. In Chapter 6, we defined the features needed for distributed software engineering and included traceability support. REQANALYST provides this feature, so in future research SOFTFAB and REQANALYST should be combined to provide distant requirements monitoring.

Finally, in this research we have discussed a number of case studies. These were academic case studies as well as industrial case studies each varying is size and complexity. Still more case studies need to be done to tune the MAREV approach and to make it more user-friendly, practical, and scalable.

Bibliography

Rasha Abbas, Philip Dart, Ed Kazmierczak, and Fergus O'Brien. Outsourcing software applications development: issues, implications, and impact. Technical Report 97/27, University of Melbourne, December 1997. URL http://www.cs.mu.oz.au/publications/tr_db/mu_97_27.ps.gz. 20 pages.

Amer Al-Rawas and Steve Easterbrook. Communication problems in requirements engineering: a field study. In *Proc. of the 1st Westminster Conf. on Professional Awareness in Softare Engineering*, London, February 1996.

Ian Alexander. Towards automatic traceability in industrial practice. In *Proc. of the 1st Int. Workshop on Traceability*, pages 26–31, Edinburgh, UK, 2002.

Thomas J. Allen. *Managing the Flow of Technology: Technology Transfer and the Dissemination of Technological Information within the R&D Organization*. MIT Press, 1977.

Giuliano Antoniol, Gerardo Canfora, Gerardo Casazza, Andrea De Lucia, and Ettore Merlo. Recovering traceability links between code and documentation. *IEEE Trans. Softw. Eng.*, 28(10):970–983, 2002. ISSN 0098-5589. doi: http://dx.doi.org/10.1109/TSE.2002.1041053.

Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. ISBN 020139829X.

M. Berry, Z. Drmac, and E. R. Jessup. Matrices, vector spaces, and information retrieval. *SIAM Review*, 41(2):335–362, 1999.

R. Binder. *Testing Object-Oriented Systems: Models, Patterns, and Tools.* Addison-Wesley, 2000.

Shawn Bohner and Robert Arnold, editors. *Software Change Impact Analysis*. IEEE Computer Society, 1996.

Sjaak Brinkkemper. Requirements engineering research the industry is and is not waiting for. In *Proc of the 10th Int. Workshop on Requirements engineering: Foundation for Software Quality*, 2004.

Erran Carmel. *Global Software Teams: Collaborating Across Borders and Time Zones*. Prentice Hall, Upper Saddle River, NJ, 1999.

James Chisan and Daniela Damian. Towards a model of awareness support of software development in gsd. In *Proc. of the 3th Int. Workshop on Global Software Development (GSD'04)*, pages 28–33, 2004.

Mary Beth Chrissis, Bart Broekman, Sandy Shrum, and Mike Konrad. *CMMI: Guidelines for Process Integration and Product Improvement*. Addison-Wesley, 2003.

James Clark and Steve DeRose. XML path language (XPath) version 1.0. Technical report, W3C, November 1999.

Jane Cleland-Huang, Carl K. Chang, and Jefrey C. Wise. Automating performance-related impact analysis through event based traceability. *Requirements Engineering*, 8(3):171–182, August 2003.

Jane Cleland-Huang, Raffaella Settimi, Chuan Duan, and Xuchang Zou. Utilizing supporting evidence to improve dynamic requirements traceability. In *Proc. of the 13th IEEE Int. Conf. on Requirements Engineering*, pages 135–144, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2425-7. doi: http://dx.doi.org/10.1109/RE.2005.78.

Paul Clements, David Garlan, Len Bass, Judith Stafford, Robert Nord, James Ivers, and Reed Little. *Documenting Software Architectures: Views and Beyond*. Pearson Education, 2002. ISBN 0201703726.

Paul Clements, David Garlan, Reed Little, Robert Nord, and Judith Stafford. Documenting software architectures: views and beyond. In *Proceedings of the 25th International Conference on Software Engineering*, pages 740–741, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1877-X.

Melvin E. Conway. How do committees invent? *Datamation*, 14(4):28–31, 1968.

Rita J. Costello and Dar-Biau Liu. Metrics for requirements engineering. *Journal of Sys. and Softw.*, 29:39–63, 1995.

Daniela Damian, James Chisan, Polly Allen, and Brian Corrie. Awareness meets requirements management: awareness needs in global software development. In *Proc. of the ICSE Int. Workshop on Global Software Development*. University of Victoria, 2003.

Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. Enhancing an artefact management system with traceability recovery features. In *Proc. of the 20th IEEE Int. Conf. on Software Maintenance*, pages 306 – 315. IEEE Computer Society, 2004.

Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. Adams re-trace: A traceability recovery tool. In *Proc. of the 9th European Conf. on Software Maintenance and Reengineering*, pages 32–41. IEEE Computer Society, March 2005.

Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, and Francesco Zurolo. COCONUT: COde COmprehension Nurturant Using Traceability. In *Proceedings of the 22nd IEEE International Confernece on Software Maintenance (ICSM'06)*, pages 274–275, Washington, DC, USA, 2006a. IEEE Computer Society. ISBN 0-7695-2354-4. doi: http://dx.doi.org/10.1109/ICSM.2006.19.

Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. Can information retrieval techniques effectively support traceability link recovery? In *Proc. of the 10th Int. Workshop on Prog. Compr.*, Athens, Greece, 2006b. IEEE Computer Society.

Andrea De Lucia, Rocco Oliveto, Francesco Zurolo, and Massimiliano Di Penta. Improving comprehensibility of source code via traceability information: a controlled experiment. In *Proceedings of the 14th IEEE International Conference on Program Comprehension (ICPC'06)*, pages 317–326, Washington, DC, USA, 2006c. IEEE Computer Society. ISBN 0-7695-2601-2. doi: http://dx.doi.org/10.1109/ICPC.2006.28.

Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. Recovering traceability links in software artifact management systems using information retrieval methods. *ACM Transactions on Software Engineering and Methodolology*, 16(4):Art. nr. 13, 2007.

S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

Department of Defence, USA. Military standard on software development and documentation (MIL-STD-498), 1994.

M. Di Penta, S. Gradara, and G. Antoniol. Traceability recovery in rad software systems. In *Proc. of the 10th Int. Workshop on Program Comprehension*, page 207, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1495-2.

Ralf Dömges and Klaus Pohl. Adapting traceability environments to project-specific needs. *Com. ACM*, 41(12):54–62, 1998. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/290133.290149.

S. Easterbrook and B. Nuseibeh. Managing inconsistencies in an evolving specification. In *Proc. of the 2th IEEE Int. Symposium on Requirements Engineering*, pages 48–55. IEEE Computer Society, 1995. ISBN 0-8186-7017-7.

A. H. Eden and R. Kazman. Architecture, design, implementation. In *Proceedings of International Conference on Software Engineering*, Portland ORA, 2003. ICSE 2003.

Ludwig D.J. Eggermont, ed.k. Embedded systems roadmap 2002. Technical report, STW Technology Foundation/PROGRESS, Utrecht, March 2002. http://www.stw.nl/progress/ESroadmap/ESRversion1.pdf.

European Space Agency. ESA software engineering standards (ESA PSS-05-0 Issue 2). Technical report, ESA Board for Software Standardization and Control (BSSC), 1991.

Matthias Findeis and Ilona Pabst. Functional safety in the automotive industry, process and methods. VDA (Verband der Automobilindustrie) Winter meeting 2006, 2006.

A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints: A framework for integrating multiple perspectives in system development. *Int. Journal of Softw. Eng. and Know. Eng.*, 2(1):31–58, March 1992.

Anthony Finkelstein. Unsolved problems in requirements engineering. Requirenautics Quarterly, BCS RESG, September 2004.

William B. Frakes and Ricardo Baeza-Yates, editors. *Information retrieval: data structures and algorithms*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992. ISBN 0-13-463837-9.

Jerry Z. Gao, Cris Chen, and David K. Leung. Engineeing on the internet for global software production. *IEEE Computer*, 32(5):38–47, May 1999.

Jerry Z. Gao, Fukao Itaru, and Y. Toyoshima. Managing problems for global software production – experience and lessons. *Information Technology and Management*, 3(1–2):85–112, January 2002.

O. Gotel and A. Finkelstein. An analysis of the requirements traceability problem. In *Proc. of the 1st IEEE Int. Conf. on Requirements Engineering*, pages 94–101, Colorado springs, April 1994.

Bas Graaf, Marco Lormans, and Hans Toetenel. Software technologies for embedded systems: An industry inventory. In *Product Focused Software Process Improvement, 4th International Conference, PROFES2002*, volume 2559 of *Lecture Notes in Computer Science*, pages 453–465. Springer-Verlag, 2002.

Bas Graaf, Marco Lormans, and Hans Toetenel. Embedded software engineering: state of the practice. *IEEE Software*, 20(6):61–69, November – December 2003.

Rebecca Grinter, James D. Herbsleb, and Dewayne E. Perry. The geography of coordination: Dealing with distance in R&D work. In *Proc. of the Int. Conf. on Supporting Group Work (GROUP'99)*, pages 306–315, New York, NY, USA, 1999. ACM Press.

H.-G. Gross, M. Lormans, and J. Zhou. Reformulating component identification as document analysis problem. In B. Shishkov J. Filipe, M. Helfert, editor, *2nd Intl Conference on Software and Data Technologies*, pages 111–116. Insticc Press, July, 22–24 2007a. ISBN 978-989-8111-10-8.

H.-G. Gross, M. Lormans, and J. Zhou. Towards software component procurement automation with latent semantic analysis. *Electronic Notes in Theoretical Computer Science*, 189:51–68, July 2007b. ISSN 1571-0661.

Derek J. Hatley and Imtiaz A. Pirbhai. *Strategies for Real-Time System Specification*. Dorset House Publishing, 1987.

David C. Hay. *Requirements Analysis: From Business Views to Architecture*. Prentice Hall PTR, 2003.

James D. Herbsleb and Audris Mockus. An empirical study of speed and communication in globally distributed software development. *IEEE Transactions on Software Engineering*, 29(6):481–494, June 2003.

James D. Herbsleb and Deependra Moitra. Global software development. *IEEE Software*, 18(2):16–20, March 2001.

C. Hofmeister, R. Nord, and D. Soni. *Applied Software Architecture*. Addison-Wesley, 1999.

Jane Huffman Hayes, Alex Dekhtyar, and James Osborne. Improving requirements tracing via information retrieval. In *Proc. of the 11th IEEE Int. Conf. on Requirements Engineering*, page 138, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1980-6.

Jane Huffman Hayes, Alex Dekhtyar, and Senthil Karthikeyan Sundaram. Improving ater-the-fact tracing and mapping: Supporting software quality predictions. *IEEE Software*, 22(6):30–37, 2005.

Jane Huffman Hayes, Alex Dekhtyar, and Senthil Sundaram. Advancing candidate link generation for requirements tracing: The study of methods. *IEEE Trans. on Softw. Eng.*, 32(1):4–19, January 2006.

M.E.C. Hull, K. Jackson, and A.J.J. Dick. *Requirements Engineering*. Springer, 2002.

W.S. Humphrey. *Managing the Software Process*. Addison-Wesley, 1989.

IEEE. IEEE recommended practice for architectural description of software intensive systems (IEEE-std-1471), 2000.

IEEE. IEEE guide for developing system requirements specifications (IEEE-std-1233), 1998a.

IEEE. IEEE recommended practice for software requirements specifications (IEEE-std-830), 1998b.

H. B. M. Jonkers. Ispec: Towards practical and sound interface specifications. In *Proc. of the 2nd Int. Conf. on Integrated Formal Methods*, pages 116–135, London, UK, 2000. Springer-Verlag. ISBN 3-540-41196-8.

Gerald Kontonya and Ian Sommerville. *Requirements Engineering: Processes and Techniques*. John Wiley and Sons, 1998.

Philippe Kruchten. The 4+1 view model of architecture. *IEEE Softw.*, 12(6): 42–50, 1995. ISSN 0740-7459. doi: http://dx.doi.org/10.1109/52.469759.

Patricia Lago and Hans van Vliet. Explicit assumptions enrich architectural models. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 206–214, 2005. ISBN 1-59593-963-2.

Filippo Lanubile. A P2P toolset for distributed requirements elicitation. In *Proc. of the 2nd Int. Workshop Global Software Development (GSD'03)*, pages 12–15, 2003.

Filippo Lanubile, Teresa Mallardo, and Fabio Calefato. Tool support for geographically dispersed inspection teams. *Software Process Improvement and Practice*, 8(4):217–231, October 2003.

M. M. Lehman Laszlo A. Belady. A model of large program development. *IBM Systems Journal*, 15(3):225–252, 1976.

Dean Leffingwell and Don Widrig. *Managing Software Requirements: A Unified Approach*. Addison Wesley, 2000.

M. M. Lehman. Software's future: Managing evolution. *IEEE Software*, 15 (1):40–44, January – February 1998.

Jun Lin and *et al.* Poirot: A distributed tool supporting enterprise-wide traceability. In *Tool Demo at the IEEE Int. Conf. on Requirements Engineering*. IEEE Computer Society, September 2006.

M. Lindvall and K. Sandahl. Practical implications of traceability. *Softw. Pract. Exper.*, 26(10):1161–1180, 1996. ISSN 0038-0644.

C.L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.

Marco Lormans. Monitoring requirements evolution using views. In *Proceedings of the 11th European Conference on Software Maintenance and Reengineering*, Amsterdam, The Netherlands, March 2007.

Marco Lormans and Arie van Deursen. Reconstructing requirements coverage views from design and test using traceability recovery via LSI. In *Proc. of the Int. Workshop on Traceability in Emerging Forms of Software Engineering*, pages 37–42, Long Beach, CA, USA, November 2005.

Marco Lormans and Arie van Deursen. Can LSI help reconstructing requirements traceability in design and test? In *Proc. of the 10th European Conf. on Software Maintenance and Reengineering*, pages 47–56, Bari, Italy, March 2006. IEEE Computer Society.

Marco Lormans and Arie van Deursen. Reconstructing requirements traceability in design and test using latent semantic indexing. *Journal of Software Maintenance and Evolution: Research and Practice*, 2008. Submitted for publication.

Marco Lormans, Hylke van Dijk, Arie van Deursen, Eric Nöcker, and Aart de Zeeuw. Managing evolving requirements in an outsoucring context: An industrial experience report. In *Proc. of the Int. Workshop on Principles of Software Evolution (IWPSE'04)*, Kyoto, Japan, 2004. IEEE Computer Society.

Marco Lormans, Hans-Gerhard Gross, Arie van Deursen, Rini van Solingen, and Andre Stéhouwer. Monitoring requirements coverage using reconstructed views: An industrial case study. In *Proc. of the 13th Working Conf. on Reverse Engineering*, pages 275–284, Benevento, Italy, October 2006.

Marco Lormans, Arie van Deursen, and Hans-Gerhard Gross. An industrial case study in reconstructing requirements views. *Journal of Empirical Software Engineering*, 2008.

Jonathan I. Maletic, Ethan V. Munson, Andrian Marcus, and Tien N. Nguyen. Using a hypertext model for traceability link conformance analysis. In *Proc. of the Int. Workshop on Traceability in Emerging Forms of Software Engineering*, pages 47–54, Montreal, Canada, 2003.

A. Marcus, J.I. Maletic, and A. Sergeyev. Recovery of traceability links between software documentation and source code. *Int. Journal of Softw. Eng. and Know. Eng.*, 15(5):811–836, October 2005a.

Andrian Marcus and Jonathan I. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *Proc. of the 25th Int. Conf. on Software Engineering*, pages 125–135, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1877-X.

Andrian Marcus, Xinrong Xie, and Denys Poshyvanyk. When and how to visualize traceability links. In J.I. Maletic, J. Cleland-Huang, J. Huffman Hayes, and G. Antoniol, editors, *3rd Intl Workshop on Traceability in Emerging Forms of Software Engineering*, pages 56–61, Long Beach, California, November 2005b.

MERLIN. Embedded systems engineering in collaboration, 2005. URL http://www.merlinproject.org.

MOOSE. software engineering MethOdOlogieS for Embedded systems, 2004. URL http://www.mooseproject.org.

Hans W. Nissen, Manfred A. Jeusfeld, Matthias Jarke, Georg V. Zemanek, and Harald Huber. Managing multiple requirements perspectives with metamodels. *IEEE Softw.*, 13(2):37–48, 1996. ISSN 0740-7459. doi: http://dx.doi.org/10.1109/52.506461.

Bashar Nuseibeh, Jeff Kramer, and Anthony Finkelstein. A framework for expressing the relationships between multiple views in requirements specification. *IEEE Trans. Softw. Eng.*, 20(10):760–773, 1994. ISSN 0098-5589. doi: http://dx.doi.org/10.1109/32.328995.

Johan Natt och Dag, Vincenzo Gervasi, Sjaak Brinkkemper, and Bjorn Regnell. Speeding up requirements management in a product software company: Linking customer wishes to product requirements through linguistic engineering. In *Proc. of the 12th Int. Conf. on Requirements Engineering Conference*, pages 283–294, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2174-6. doi: http://dx.doi.org/10.1109/RE.2004.47.

Johan Natt och Dag, Vincenzo Gervasi, Sjaak Brinkkemper, and Bjorn Regnell. A linguistic-engineering approach to large-scale requirements management. *IEEE Softw.*, 22(1):32–39, 2005. ISSN 0740-7459. doi: http://dx.doi.org/10.1109/MS.2005.1.

OMG. MDA. http://www.omg.org/mda, 2007.

S. Park, H. Kim, Y. Ko, and J Seo. Implementation of an efficient requirements-analysis supporting system using similarity measure techniques. *Information and Software Technology*, 42(6):429–438, 2000.

Päivi Parviainen, Maarit Tihinen, Marco Lormans, and Rini van Solingen. Requirements engineering: Dealing with the complexity of sociotechnical systems development. In José Luis Maté and Andrés Silva, editors, *Requirements Engineering for Sociotechnical Systems*. IdeaGroup Inc., 2004. Chapter 1.

Roger S. Pressman. *Software Engineering, A Practitioner's Approach, 6th edition*. McGraw-Hill, 2005.

Rafael Prikladnicki, Jorge Audy, and Roberto Evaristo. Requirements management in global software development: Preliminary findings from a case study in a sw-cmm context. In *Proc. of the ICSE Int. Workshop on Global Software Development*. University of Victoria, 2003.

B. Ramesh and M. Jarke. Toward reference models for requirements traceability. *IEEE Trans. Softw. Eng.*, 27(1):58–93, 2001. ISSN 0098-5589. doi: http://dx.doi.org/10.1109/32.895989.

B. Ramesh, T. Powers, C. Stubbs, and M. Edwards. Implementing requirements traceability: a case study. In *Proc. of the 2nd IEEE Int. Symp. on Requirements Engineering*, page 89, Washington, DC, USA, 1995. IEEE Computer Society. ISBN 0-8186-7017-7.

C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 1979. ISBN 0408709294.

James Robertson and Suzanne Robertson. Volere requirements specification template. Technical report, Atlantic Systems Guild, 2000.

Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986. ISBN 0070544840.

Raffaella Settimi, Jane Cleland-Huang, Oussama Ben Khadra, Jigar Mody, Wiktor Lukasik, and Chris DePalma. Supporting software evolution through dynamically retrieving traces to UML artifacts. In *Proc of the 7th Int. Workshop on Principles of Software Evolution*, pages 49–54, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2211-4. doi: http://dx.doi.org/10.1109/IWPSE.2004.21.

Ian Sommerville. *Software Engineering, 6th edition*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001. ISBN 0201175681.

Hans Spanjers, Maarten ter Huurne, Bas Graaf, Marco Lormans, Dan Bendas, and Rini van Solingen. Tool support for distributed software engineering. In *Proceedings of the IEEE International Conference on Global Software Engineering (ICGSE'06)*, pages 187–198, Florianopolis, Brazil, 2006. IEEE Computer Society.

Marco Toranzo and Jaelson Castro. A comprehensive traceability model to support the design of interactive systems. In *Proc. of the Workshop on Object-Oriented Technology*, pages 283–284, London, UK, 1999. Springer-Verlag. ISBN 3-540-66954-X.

A. van Deursen, C. Hofmeister, R. Koschke, L. Moonen, and C. Riva. Symphony: View-driven software architecture reconstruction. In *Proceedings Working IEEE/IFIP Conference on Software Architecture (WICSA'04)*, pages 122–134. IEEE Computer Society Press, 2004.

Arie van Deursen and Leon Moonen. Documenting software systems using types. *Science of Computer Programming*, 60(2):205–220, April 2006. URL http://www.st.ewi.tudelft.nl/~arie/papers/types/scp2001.pdf.

Rob van Ommering, Frank van der Linden, Jeff Kramer, and Jeff Magee. The Koala component model for consumer electronics software. *IEEE Computer*, 33(3):78–85, March 2000.

Hans van Vliet. *Software Engineering: Principles and Practice, 3rd Edition*. Wiley, 2008.

Antje von Knethen. A trace model for system requirements changes on embedded systems. In *Proc. of the 4th Int. Workshop on Principles of*

Software Evolution, pages 17–26, New York, NY, USA, 2001. ACM Press. ISBN 1-58113-508-4. doi: http://doi.acm.org/10.1145/602461.602465.

Antje von Knethen, Barbara Paech, Friedemann Kiedaisch, and Frank Houdek. Systematic requirements recycling through abstraction and traceability. In *Proc. of the Int. Conf. on Requirements Engineering*, pages 273–281, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1465-0.

Yingxu Wang, Graham King, Hakan Wickberg, and Alec Dorling. What the software industry says about the practices modelled in current software process models? In *Proceedings of the 25th EUROMICRO Conference*, volume 2, pages 162–168. IEEE Computer Society, 1999.

Karl E. Wiegers. Automating requirements management. *Software Development*, 7(7), July 1999.

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

Robert K. Yin. *Applications of Case Study Research*. Sage Publications, 2003b.

John A. Zachman. A framework for information systems architecture. *IBM Syst. J.*, 26(3):276–292, 1987. ISSN 0018-8670.

A. Zisman, G. Spanoudakis, E. Perez-Mi nana, and P.Krause. Tracing software requirements artifacts. In *Proc. of Int. Conf. on Software Engineering Research and Practice*, pages 448–455, Las Vegas, Nevada, USA, 2003.

Xuchang Zou, Raffaella Settimi, Jane Cleland-Huang, and Chuan Duan. Thresholding strategy in requirements trace retrieval. In *CTI Research Symposium*, pages 100–103, 2004.

Samenvatting

Moderne software-systemen worden steeds complexer. Ook de omgeving waarin deze systemen opereren wordt steeds dynamischer. Het aantal belanghebbenden neemt tevens toe en ook de wensen en eisen (requirements) van de belanghebbenden veranderen constant als gevolg van de als maar veranderende omgeving. Een consequentie van deze trend is dat het aantal eisen aan een software-systeem toeneemt. Daarnaast veranderen de eisen van een belanghebbende continue.

Binnen het software-ontwikkelproces is requirements-engineering het process dat er voor moet zorgen dat de wensen en eisen aan het software-systeem van alle belanghebbenden worden gespecificeerd. Tevens behoort het managen van deze wensen en eisen gedurende het totale software-ontwikkelproces tot de activiteiten van dit proces. In tegenstelling tot de klassieke manier van software-ontwikkeling, waar het proces requirements-engineering enkel in de beginfase van het totale software-ontwikkelproces is gepositioneerd, zijn wij van mening dat deze activiteit het totale software-ontwikkelproces omvat, van begin tot eind.

Het requirements-engineeringproces moet in de moderne aanpak van software-ontwikkeling om kunnen gaan met de eerste twee wetten van software-evolutie gedefineerd door Lehman: *Continue verandering* en *Toenemende complexiteit* [Lehman, 1998; Laszlo A. Belady, 1976]. Deze twee wetten zeggen dat 1) software-systemen continue aangepast moeten worden aan de nieuwe wensen en eisen van de belanghebbenden of de veranderingen in hun omgeving om niet nutteloos te worden en 2) software-systemen die evolueren worden steeds complexer door de nieuwe of veranderende wensen en eisen van belanghebbenden.

Probleem

In dit onderzoeksproject hebben we ons geconcentreerd op het managen van het requirements-evolutieproces. De hoofdvraag die we willen beantwoorden is: *Hoe kunnen we een requirements-managementomgeving ontwikkelen die het managen van requirements-evolutie verbeterd?*

In dit onderzoeksproject zijn we begonnen met het identificeren van de problemen die organisaties typisch ondervinden met betrekking tot de activiteiten gerelateerd aan requirements-engineering en meer specifiek, tijdens het managen van requirements. Hiervoor hebben we een vragenlijst opgesteld en verschillende interviews gehouden bij de industriële partners van de MOOSE- en MERLIN-projecten. We hebben ons binnen het onderzoek specifiek geconcentreerd op het uitbesteden van software-ontwikkeling en het ontwikkelen van software op verschillende locaties (beter bekend onder de naam Global Software Development). De problemen op het gebied van het managen van wensen en eisen aan een software-systeem zijn anders voor deze type projecten. We hebben bijvoorbeeld een project bij Logica ter hande genomen waarbij de software-ontwikkeling was uitbesteed aan een derde partij. Daarnaast hebben we een software-ontwikkeltool (gereedschap), genaamd SOFTFAB en ontwikkelt door Philips, bestudeerd dat er speciaal voor bedoeld is om gedistribueerde software-ontwikkeling te ondersteunen.

De praktijkstudies bevestigden dat het managen van wensen en eisen een groot probleem is in de praktijk. De wensen en eisen aan een software-systeem veranderen continue waardoor de traceerbaarheid van wensen en eisen moeilijk is. Dit leidt er toe dat het managen van deze eisen onbetrouwbaar wordt. Dit leidt er vervolgens weer toe dat het onmogelijk wordt om de voortgang te monitoren van de eisen binnen het software-ontwikkelproces: zijn de eisen en wensen al in het ontwerp meegenomen en worden ze al afgedekt in de testgevallen? Uiteindelijk kan dit er toe leiden dat het opgeleverde systeem niet voldoet aan de vooraf gestelde eisen van de belanghebbenden.

Resultaten

Als oplossing hebben we een raamwerk gedefinieerd, genaamd RES (Requirements Engineering System). RES definieert drie hoofdprocessen die relevant zijn voor het managen van requirements-evolutie: 1) de interactie met belanghebbenden, 2) het consistent verwerken van veranderingen en 3) het presenteren van de informatie die betrekking heeft op de wensen en eisen van de belanghebbenden, daarbij gebruikmakend van zogenaamde “requirements views”.

Daarnaast hebben we een methodologie ontwikkelt, genaamd MAREV (Methodology for Automating Requirements Evolution using Views), die het proces van traceerbaarheid van wensen en eisen automatiseert. De methodologie dekt alle drie de processen die gedefinieerd zijn in het RES-raamwerk af en bestaat uit een totaal van 7 stappen.

We hebben de MAREV-methodologie geïmplementeerd in een software-produkt (tool suite), genaamd REQANALYST, voor het ondersteunen van software-ontwikkeling. REQANALYST gebruikt “Latent Semantic Indexing” (LSI), een “Information Retrieval”-techniek voor het reconstrueren van traceerbaarheidsrelaties tussen de wensen en eisen van belanghebbenden naar andere werkproducten, die tijdens het ontwikkelproces geproduceerd worden, zoals ontwerpen en testgevallen. Deze traceerbaarheidsinformatie wordt vervolgens gebruikt bij het genereren van “requirements views”. Deze views helpen bij het monitoren en managen van de wensen en eisen waaraan het software-systeem moet voldoen.

Tot slot, hebben we onze MAREV-methodologie toegepast in verschillende academische en industriële praktijkstudies welke hebben geresulteerd in tal van leer- en verbeterpunten.

Curriculum Vitae

Marco Lormans was born in Heemskerk on the 23th of April 1978. He received his VWO diploma in 1996 at the Berlingh College in Beverwijk. Next, he started his study in computer science at Delft University of Technology. In 2002, he received his master degree in computer science from the section Design of Information Systems. The topic of his master thesis concerned component-based development and he was supervised by Prof. dr. ir. J.L.G. Dietz and Dr. P.G. Kluit.

In 2002 he started his Ph.D. research in the Software Engineering group of the department of Software Technology at Delft University of Technology. His supervisor was Prof. Dr. A. van Deursen head of the section Software Engineering and SWERL lab[1]. Marco investigated the evolution of requirements, and focussed on how to manage this process of requirements evolution.

Currently, Marco is working as consultant Systems Development at Logica[2]. He consults organisations on how to built software systems and control the development process. This includes, amongst other things, consulting on how to implement requirements management and tackle requirements evolution.

[1]http://swerl.tudelft.nl
[2]http://www.logica.com

Titles in the IPA Dissertation Series since 2002

M.C. van Wezel. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects*. Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. *Techniques for Understanding Legacy Software Systems*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

S.P. Luttik. *Choice Quantification in Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

R.J. Willemen. *School Timetable Construction: Algorithms and Complexity*. Faculty of Mathematics and Computer Science, TU/e. 2002-05

M.I.A. Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-06

N. van Vugt. *Models of Molecular Computing*. Faculty of Mathematics and Natural Sciences, UL. 2002-07

A. Fehnker. *Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-08

R. van Stee. *On-line Scheduling and Bin Packing*. Faculty of Mathematics and Natural Sciences, UL. 2002-09

D. Tauritz. *Adaptive Information Filtering: Concepts and Algorithms*. Faculty of Mathematics and Natural Sciences, UL. 2002-10

M.B. van der Zwaag. *Models and Logics for Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-11

J.I. den Hartog. *Probabilistic Extensions of Semantical Models*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2002-12

L. Moonen. *Exploring Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-13

J.I. van Hemert. *Applying Evolutionary Computation to Constraint Satisfaction and Data Mining*. Faculty of Mathematics and Natural Sciences, UL. 2002-14

S. Andova. *Probabilistic Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2002-15

Y.S. Usenko. *Linearization in μCRL*. Faculty of Mathematics and Computer Science, TU/e. 2002-16

Division of Mathematics and Computer Science, VUA. 2004-06

L. Cruz-Filipe. *Constructive Real Analysis: a Type-Theoretical Formalization and Applications*. Faculty of Science, Mathematics and Computer Science, KUN. 2004-07

E.H. Gerding. *Autonomous Agents in Bargaining Games: An Evolutionary Investigation of Fundamentals, Strategies, and Business Applications*. Faculty of Technology Management, TU/e. 2004-08

N. Goga. *Control and Selection Techniques for the Automated Testing of Reactive Systems*. Faculty of Mathematics and Computer Science, TU/e. 2004-09

M. Niqui. *Formalising Exact Arithmetic: Representations, Algorithms and Proofs*. Faculty of Science, Mathematics and Computer Science, RU. 2004-10

A. Löh. *Exploring Generic Haskell*. Faculty of Mathematics and Computer Science, UU. 2004-11

I.C.M. Flinsenberg. *Route Planning Algorithms for Car Navigation*. Faculty of Mathematics and Computer Science, TU/e. 2004-12

R.J. Bril. *Real-time Scheduling for Media Processing Using Conditionally Guaranteed Budgets*. Faculty of Mathematics and Computer Science, TU/e. 2004-13

J. Pang. *Formal Verification of Distributed Systems*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-14

F. Alkemade. *Evolutionary Agent-Based Economics*. Faculty of Technology Management, TU/e. 2004-15

E.O. Dijk. *Indoor Ultrasonic Position Estimation Using a Single Base Station*. Faculty of Mathematics and Computer Science, TU/e. 2004-16

S.M. Orzan. *On Distributed Verification and Verified Distribution*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-17

M.M. Schrage. *Proxima - A Presentation-oriented Editor for Structured Documents*. Faculty of Mathematics and Computer Science, UU. 2004-18

E. Eskenazi and A. Fyukov. *Quantitative Prediction of Quality Attributes for Component-Based Software Architectures*. Faculty of Mathematics and Computer Science, TU/e. 2004-19

P.J.L. Cuijpers. *Hybrid Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2004-20

N.J.M. van den Nieuwelaar. *Supervisory Machine Control by Predictive-Reactive Scheduling*. Faculty of Mechanical Engineering, TU/e. 2004-21

E. Ábrahám. *An Assertional Proof System for Multithreaded Java - Theory and Tool Support-* . Faculty of Mathematics and Natural Sciences, UL. 2005-01

R. Ruimerman. *Modeling and Remodeling in Bone Tissue*. Faculty of Biomedical Engineering, TU/e. 2005-02

C.N. Chong. *Experiments in Rights Control - Expression and Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-03

H. Gao. *Design and Verification of Lock-free Parallel Algorithms*. Faculty of Mathematics and Computing Sciences, RUG. 2005-04

H.M.A. van Beek. *Specification and Analysis of Internet Applications*. Faculty of Mathematics and Computer Science, TU/e. 2005-05

M.T. Ionita. *Scenario-Based System Architecting - A Systematic Approach to Developing Future-Proof System Architectures*. Faculty of Mathematics and Computing Sciences, TU/e. 2005-06

G. Lenzini. *Integration of Analysis Techniques in Security and Fault-Tolerance*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-07

I. Kurtev. *Adaptability of Model Transformations*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-08

T. Wolle. *Computational Aspects of Treewidth - Lower Bounds and Network Reliability*. Faculty of Science, UU. 2005-09

O. Tveretina. *Decision Procedures for Equality Logic with Uninterpreted Functions*. Faculty of Mathematics and Computer Science, TU/e. 2005-10

A.M.L. Liekens. *Evolution of Finite Populations in Dynamic Environments*. Faculty of Biomedical Engineering, TU/e. 2005-11

J. Eggermont. *Data Mining using Genetic Programming: Classification and Symbolic Regression*. Faculty of Mathematics and Natural Sciences, UL. 2005-12

B.J. Heeren. *Top Quality Type Error Messages*. Faculty of Science, UU. 2005-13

G.F. Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations*. Faculty of Science, Mathematics and Computer Science, RU. 2005-14

M.R. Mousavi. *Structuring Structural Operational Semantics*. Faculty of Mathematics and Computer Science, TU/e. 2005-15

A. Sokolova. *Coalgebraic Analysis of Probabilistic Systems*. Faculty of Mathematics and Computer Science, TU/e. 2005-16

T. Gelsema. *Effective Models for the Structure of pi-Calculus Processes with Replication*. Faculty of Mathematics and Natural Sciences, UL. 2005-17

P. Zoeteweij. *Composing Constraint Solvers*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-18

J.J. Vinju. *Analysis and Transformation of Source Code by Parsing and Rewriting*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-19

M.Valero Espada. *Modal Abstraction and Replication of Processes with Data*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2005-20

A. Dijkstra. *Stepping through Haskell*. Faculty of Science, UU. 2005-21

Y.W. Law. *Key management and link-layer security of wireless sensor networks: energy-efficient attack and defense*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-22

E. Dolstra. *The Purely Functional Software Deployment Model*. Faculty of Science, UU. 2006-01

R.J. Corin. *Analysis Models for Security Protocols*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-02

P.R.A. Verbaan. *The Computational Complexity of Evolving Systems*. Faculty of Science, UU. 2006-03

K.L. Man and R.R.H. Schiffelers. *Formal Specification and Analysis of Hybrid Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2006-04

M. Kyas. *Verifying OCL Specifications of UML Models: Tool Support and Compositionality*. Faculty of Mathematics and Natural Sciences, UL. 2006-05

M. Hendriks. *Model Checking Timed Automata - Techniques and Applications*. Faculty of Science, Mathematics and Computer Science, RU. 2006-06

J. Ketema. *Böhm-Like Trees for Rewriting*. Faculty of Sciences, VUA. 2006-07

C.-B. Breunesse. *On JML: topics in tool-assisted verification of JML programs*. Faculty of Science, Mathematics and Computer Science, RU. 2006-08

B. Markvoort. *Towards Hybrid Molecular Simulations*. Faculty of Biomedical Engineering, TU/e. 2006-09

S.G.R. Nijssen. *Mining Structured Data*. Faculty of Mathematics and Natural Sciences, UL. 2006-10

G. Russello. *Separation and Adaptation of Concerns in a Shared Data Space*. Faculty of Mathematics and Computer Science, TU/e. 2006-11

L. Cheung. *Reconciling Nondeterministic and Probabilistic Choices*. Faculty of Science, Mathematics and Computer Science, RU. 2006-12

B. Badban. *Verification techniques for Extensions of Equality Logic*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2006-13

A.J. Mooij. *Constructive formal methods and protocol standardization*. Faculty of Mathematics and Computer Science, TU/e. 2006-14

T. Krilavicius. *Hybrid Techniques for Hybrid Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-15

M.E. Warnier. *Language Based Security for Java and JML*. Faculty of Science, Mathematics and Computer Science, RU. 2006-16

V. Sundramoorthy. *At Home In Service Discovery*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-17

B. Gebremichael. *Expressivity of Timed Automata Models*. Faculty of Science, Mathematics and Computer Science, RU. 2006-18

L.C.M. van Gool. *Formalising Interface Specifications*. Faculty of Mathematics and Computer Science, TU/e. 2006-19

C.J.F. Cremers. *Scyther - Semantics and Verification of Security Protocols*. Faculty of Mathematics and Computer Science, TU/e. 2006-20

J.V. Guillen Scholten. *Mobile Channels for Exogenous Coordination of Distributed Systems: Semantics, Implementation and Composition*. Faculty of Mathematics and Natural Sciences, UL. 2006-21

H.A. de Jong. *Flexible Heterogeneous Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-01

N.K. Kavaldjiev. *A run-time reconfigurable Network-on-Chip for streaming DSP applications*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-02

M. van Veelen. *Considerations on Modeling for Early Detection of Abnormalities in Locally Autonomous Distributed Systems*. Faculty of Mathematics and Computing Sciences, RUG. 2007-03

T.D. Vu. *Semantics and Applications of Process and Program Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-04

L. Brandán Briones. *Theories for Model-based Testing: Real-time and Coverage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-05

I. Loeb. *Natural Deduction: Sharing by Presentation*. Faculty of Science, Mathematics and Computer Science, RU. 2007-06

M.W.A. Streppel. *Multifunctional Geometric Data Structures*. Faculty of Mathematics and Computer Science, TU/e. 2007-07

N. Trčka. *Silent Steps in Transition Systems and Markov Chains*. Faculty of Mathematics and Computer Science, TU/e. 2007-08

R. Brinkman. *Searching in encrypted data*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-09