Managing Natural Language Requirements in Large-Scale Software Development

Johan Natt och Dag

LUND UNIVERSITY

Department of Communication Systems
Lund Institute of Technology

ISSN 1101-3931
ISRN LUTEDX/TETS–1070–SE+222P

Printed in Sweden
E-kop
Lund 2005

"The Machines are only tools after all, which can help humanity progress faster by taking some of the burdens of calculations and interpretations off its back. The task of the human brain remains what it has always been; that of discovering new data to be analyzed, and of devising new concepts to be tested."

Hiram Mackenzie
Vice-Co-ordinator, The Northern Region

in 'I, robot' by Isaac Asmiov, 1950

This thesis is submitted to Research Board FIME - Physics, Informatics, Mathematics and Electrical Engineering - at Lund Institute of Technology, Lund University in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Engineering.

Contact information:

Johan Natt och Dag
Department of Communication Systems
Lund University
PO Box 118
SE-221 00 LUND
Sweden

Phone: +46 46 222 08 83
Fax: +46 46 14 58 23
e-mail: johan.nattochdag@telecom.lth.se

Abstract

An increasing number of market- and technology-driven software development companies face the challenge of managing several thousands of requirements written in natural language. The large number of requirements causes bottlenecks in the requirements management process and calls for increased efficiency in requirements engineering.

This thesis presents results from empirical investigations of using linguistic engineering techniques to alleviate three requirements management activities in large-scale software development: identification of duplicate requirements, linkage of related requirements, and consolidation of different sets of requirements. The activities rely on one common activity: finding requirements that are semantically similar, i.e., refer to the same underlying functionality.

Three case studies are presented, in which three different companies, comprising three different requirements management challenges, are investigated. Simulation is used to explore process bottlenecks and two different sets of industrial requirements are used for evaluating suggested solutions. A controlled experiment is also presented, evaluating a new open source support tool for semi-automatic identification of similar requirements.

The results show that, for the investigated activities, lexical similarity between requirements may be a sufficient approximation of their semantic similarity. It is also shown that automatic calculation of this similarity may support the activities and give valuable time-savings. The results from the presented research point in one direction: that simple, robust, and cost-efficient linguistic engineering techniques can give effective support to requirements management activities.

Acknowledgements

First of all, I want to thank Dr. Björn Regnell for supporting, encouraging, guiding, and restraining me throughout my work. Your competence has been so valuable. Also, special thanks to Prof. Per Runeson for invaluable support and discussions in my research and in my studies.

I also want to thank Prof. Claes Wohlin for attracting my attention to PhD studies in software engineering. My PhD studies have been a worthwhile experience and have provided me with better tools and experience to draw reasonable conclusions on phenomena in software engineering.

Many thanks to all co-authors and each and every one who have contributed to the research presented in this thesis. Special thanks to my most recent and closest collaborators, Dr. Vincenzo Gervasi and Prof. Sjaak Brinkkemper, whose collaboration I have enjoyed tremendously. My thoughts also goes to all the researchers and industrial representatives that I have met, discussed with, and been inspired by at conferences and meetings all over the world.

Thanks to all my colleges at the department with whom I have struggled with a number of research issues, assignments, and teaching issues, as well as a collection of more trivial but highly appreciated and amusing questions.

Last, but undoubtedly not least, thanks to friends and family who have listened and cared. Invaluable and always remembered.

Johan Natt och Dag
Lund, January 2005

Table of Contents

List of papers

The following papers are included in this thesis:

[I] **Exploring Bottlenecks in Market-Driven Requirements Management Processes with Discrete Event Simulation**
Martin Höst, Björn Regnell, Johan Natt och Dag, Josef Nedstam, & Christian Nyberg
Journal of Systems and Software, 59, pp. 323–332, 2001.

[II] **A Feasibility Study of Automated Support for Similarity Analysis of Natural Language Requirements in Market-Driven Development**
Johan Natt och Dag, Björn Regnell, Pär Carlshamre, Michael Andersson, & Joachim Karlsson
Requirements Engineering, 7(1), pp. 20-33, 2002.

[III] **Speeding up Requirements Management in a Product Software Company: Linking Customer Wishes to Product Requirements through Linguistic Engineering**
Johan Natt och Dag, Vincenzo Gervasi, Sjaak Brinkkemper, & Björn Regnell
Proceedings of the 12th International Requirements Engineering Conference, RE2004, pp. 283–294, Kyoto, Japan, September 2004. IEEE CS.

[IV] **A Linguistic Engineering Approach to Large-Scale Requirements Management**
Johan Natt och Dag, Vincenzo Gervasi, Sjaak Brinkkemper, & Björn Regnell
IEEE Software, 22(1), pp. 32–39, 2005

[V] **An Experiment on Linguistic Tool Support for Consolidation of Requirements from Multiple Sources in Market-Driven Product Development**
Johan Natt och Dag, Thomas Thelin, & Björn Regnell. Submitted to *Empirical Software Engineering*. 2005.

[A] **Appendix: ReqSimile Technical Architecture**
Johan Natt och Dag
Technical Report, CODEN:LUTEDX(TETS-7206)/1-28/(2005)&local1, Department of Communication Systems, Lund University, Sweden, 2000.

Related publications

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

[VI] **Exploring Bottlenecks in Market-Driven Requirements Management Processes with Discrete Event Simulation**
Martin Höst, Björn Regnell, Johan Natt och Dag, Josef Nedstam, & Christian Nyberg
Paper presented at the *Software Process Simulation Modeling Workshop*, London, UK, July, 2000.
This paper is an earlier version of Paper I. It was selected and extended for a special issue of *Journal of Systems and Software*.

[VII] **An industrial case study of usability evaluation**
Johan Natt och Dag & Ofelia S Madsen.
Master Thesis, Report No.
CODEN:LUTEDX(TETS-5390)/1–190/(2000)&local8. Department of Communication Systems, Lund Unviersity, Sweden, 2000.
This thesis contains the basis for the research presented in Paper VIII. It contains more elaborate background information and more research data.

[VIII] **An industrial case study of usability engineering in market-driven packaged software development**
Johan Natt och Dag, Björn Regnell, Ofelia S Madsen, & Aybüke Aurum.
M. J. Smith, G. Salvendy, D. Harris & R. J. Koubek (Eds.), *Proceedings of HCI International: Vol 1. Usability Evaluation and Interface Design: Cognitive Engineering, Intelligent Agents and Virtual Reality*, pp. 425–429, Mahwah, NJ: Erlbaum, 2001.
This paper presents the results and experiences from conducting two known usability evaluation, using a questionnaire and a heuristic evaluation, at a large software development company.

[IX] **Visualization of agreement and satisfaction in distributed prioritization of market requirements**
Björn Regnell, Martin Höst, Johan Natt och Dag, Per Beremark, & Thomas Hjelm
A. L. Opdahl, K. Pohl & M. Rossi (Eds.), *Proceedings of the Sixth International Workshop on Requirements Engineering: Foundation for Software Quality*, pp. 125–136. Essen, Germany: Essener Informatik Beiträge, 2000.

This paper is an earlier version of Paper IX. It was selected and extended for publication in a special issue of *Requirements Engineering*.

[X] **An industrial case study on distributed prioritisation in market-driven requirements engineering for packaged software**
Björn Regnell, Martin Höst, Johan Natt och Dag, Per Beremark, & Thomas Hjelm
Requirements Engineering, 6(1), pp. 51–62, 2001
This paper presents an industrial case study where a distributed prioritization process is proposed, observed, and evaluated. The paper also presents an approach to visualize the priority distribution among stakeholders, together with measures on disagreement and satisfaction.

[XI] **Evaluating automated support for requirements similarity analysis in market-driven development**
Johan Natt och Dag, Björn Regnell, Pär Carlshamre, Michael Andersson, & Joachim Karlsson
C. Saliensi, A. L. Opdahl, K. Pohl, & M. Rossi (Eds.), *Proceedings of the Seventh International Workshop on Requirements Engineering: Foundations for Software Quality*, pp. 190-201. Essen, Germany: Essen Informatik Beiträge, 2001.
This paper is an earlier version of Paper II. It was selected and extended for publication in a special issue of *Requirements Engineering*.

[XII] **An industrial survey of requirements interdependencies in software release planning**
Pär Carlshamre, Kristian Sandahl, Mikael Lindvall, Björn Regnell, & Johan Natt och Dag. *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*, pp. 84–91. Los Alamitos, CA: IEEE Computer Society Press, 2001.

This contains an automated analysis of interrelationships between requirements. The techniques used are those in Paper II.

[XIII] **Requirements mean decisions! - Research issues for understanding and supporting decision-making in requirements engineering**
Björn Regnell, Barbara Paech, Aybüke Aurum, Claes Wohlin, Allen Dutoit, & Johan Natt och Dag.
Proceedings of the First Swedish Conference on Software Engineering Research and Practise, pp. 49–52) (Report No. 2001:10). Ronneby, Sweden: Blekinge Institute of Technology, Department of Software Engineering and Computer Science, 2001.
This paper presents research issues with focus on requirements engineering as a decision-making process.

[XIV] **Market-driven requirements engineering challenges: an industrial case study of a process performance declination**
Robert Booth, Björn Regnell, Aybüke Aurum, Ross Jeffery, & Johan Natt och Dag.
A. Aurum & R. Jeffery (Eds.), *Proceeding of the Sixth Australian Workshop on Requirements Engineering*, pp. 41–47. Sydney, Australia: University of New South Wales, The Centre for Advanced Software Engineering Research, 2001.
This paper presents a second study of the requirements engineering process at Telelogic AB.

[XV] **Challenges in market-driven requirements engineering - an industrial interview study**
Lena Karlsson, Åsa G. Dahlstedt, Johan Natt och Dag, Björn Regnell, & Anne Persson
C. Saliensi, K. Pohl, & B. Regnell (Eds.), *Proceedings of the Eighth International Workshop on Requirements Engineering: Foundation for Software Quality*, pp. 37–49. Essen, Germany: Essener Informatik Beiträge, 2002.
This paper presents preliminary results from an interview study of five market-driven software development companies in Sweden.

[XVI] **Market-Driven Requirements Engineering Processes for Software Products - A Report on Current Practices**
Åsa G. Dahlstedt, Lena Karlsson, Johan Natt och Dag, Björn Regnell, & Anne Persson
Proceedings of the First International Workshop on COTS and Product Software, Monterey Bay, CA, 2003.
This paper presents a further analysis of the interview study in Paper XV comprising a comparison to the characteristics of market-driven development reported in the literature.

Introduction

Software development keeps increasing in scope and pace. As software systems encompass increased functionality, new application areas, and new markets and customers, competition intensifies, expectations rise, and development complexity increases. In this seemingly never-ending upward spiral, market- and technology-driven companies are facing a challenge of dealing with information flows that may overwhelm their management and analysis capabilities.

Of all the artifacts produced and gathered within software development, requirements are particularly difficult to manage effectively due to their unstructured nature. Requirements also have a potential to grow to such volumes and arrive at such rates that specific information and knowledge management challenges emerge: an increasing difficulty to identify and maintain requirements inter-relationships and deterioration of the requirements repository in terms of duplicate, outdated, and improperly updated requirements.

Requirements management processes may be very different in design and implementation (Kotonya & Sommerville, 1997; Robertson & Robertson, 1999). Nevertheless, companies that acknowledge both customer involvement and their own innovative potential as rewarding means for discovering successful product services and functionality are faced with a common challenge: analyzing and evaluating every incoming requirement, customer wish and technical suggestion as soon and as thoroughly as possible.

A major contributing reason for these difficulties is that requirements are communicated in natural language, which induces several problems in software development, like imprecision, ambiguity, incompleteness, conflict, and inconsistency – all of which take time to resolve. As there often are insuffi-

cient resources available to resolve these requirement quality issues, supportive requirements management techniques that are more resilient to inferior linguistic quality would be desirable. As proper requirements engineering and management is regarded as an important success factor (Hofmann & Lehner, 2001), additional support would be of value to industry.

This thesis presents the results from research conducted during a period of 4 years. The major results are presented in separate articles, which are included in their original form. In the first paper, results from an empirical investigation into a market-driven requirements management process are presented. The investigation shows that there is a need for better support in the continuous management process, as it easily becomes overloaded. The need for better support gave rise to the idea of investigating if linguistic engineering techniques could support activities in the management of several thousands of requirements written in natural language. Empirical investigations are presented that address the following specific requirements management activities:

Duplicate identification With a steady stream of incoming requirements, emerging from different stakeholders and stored in a database, there is a high probability that there are several instances of requirements that address the same functionality. To avoid requirements deterioration, duplicate requirements are preferably removed or grouped together. As it is a problematic and time-consuming task to find the duplicates, automated assistance would be of great value (Paper II).

Linking similar requirements Requirements that emerge from the market and from within the organization are, for different reasons, preferably kept separated. However, assistance in finding and linking requirements that address the same functionality, is of great value in order to align business goals with customer satisfaction. (Paper III; Paper IV).

Requirements consolidation Huge requirements documents that arrive from key customers at several different occasions should be consolidated with the requirements currently stored in the database. New requirements should be added and analyzed, but in order to avoid reanalyzing arriving requirements that are already stored in the database repository (with accompanying additional information), assistance in finding the overlap is of great value (Paper V).

The first part of this thesis presents the background to and summarizes the research work. The second part comprise the five papers that constitute the main contribution of the thesis.

1 Background

The research and the associated results presented in this thesis apply to the field of software engineering, in which methods, techniques, and tools are utilized to overcome the challenges in development and maintenance of complex software systems (Sommerville, 2001). About 15 years ago, a sub-discipline within software engineering emerged due to specific challenges in handling customers' wishes and needs (Sommerville & Sawyer, 1997). The sub-discipline, termed requirements engineering (RE), involves the activities in software development where customers' wishes and needs, i.e. the requirements, are elicited, specified, analyzed and selected before proceeding with software design, implementation, verification and validation.

The last years, a new approach to developing software, *market-driven development* or packaged software development (Sawyer et al., 1999), has gained increasing importance as software development companies turn to new and larger markets. The significance of market-driven development is also indicated by the growing interest in commercial off the shelf (COTS) development in the RE research community (Wieringa & Ebert, 2004). In market-driven development, software is developed to satisfy the needs of a range of different customers and end-users, as opposed to in contractual development where there is only a single customer. The approach affects requirements engineering in several ways. Competition in the market must be defeated, and one way to do this is by secretly developing successful solutions and presenting them to the market before any competitor. The competition puts a schedule constraint where short time-to-market is crucial (Sawyer, 2000). Furthermore, to keep the competitive advantage, negotiation with customers and end-users is very limited. Instead, many requirements are invented within the developing company (Potts, 1995). Constantly striving to be ahead of competitors, the market-driven development company therefore frequently delivers new and improved releases of a software system, in order to keep old customers satisfied and to win new ones (Potts, 1995; Carlshamre & Regnell, 2000).

The characteristic differences between contractual, or bespoke, software development and market-driven software development have been summarized by Carlshamre (2002). This summary is found in Table 1.1 (derived from Kamsties et al. 1998; Keil & Carmel 1995; Lubars et al. 1993; Novorita & Grube 1996; Potts 1995; Yeh 1992 with minor additions from Lubars et al. 1993; Robertson & Robertson 1999). In particular, fundamental organizational issues, such as the primary goal, the success measurements and the

Table 1.1: Comparison of bespoke software development and market-driven software development characteristics (from Carlshamre 2002 with additions from Lubars et al. 1993; Robertson & Robertson 1999)

Characteristic	**Bespoke development**	**Market-driven development**
Primary goal	Compliance to requirements specification.	Time-to-market. Requirements are jettisoned rather than allowing delay of release.
Measure of success	Satisfaction, acceptance.	Sales, market share, product reviews.
Life cycle	One release, then maintenance.	Several releases, as long as there is a market for the product.
Requirements conception	Elicited, analyzed, validated.	Invented. Either the market (marketing department) permits a feature, or technology does.
Requirements specification	Used as a contract between customer and supplier.	Rarely exists in orthodox RE terms, if so, they are much less formal. Requirements are communicated verbally.
Users/end-users[1]	Known or easily identifiable.	Difficult to identify or initially unknown.
Customers[1]	Software orderer. Contract negotiator.	Agents for different markets. Key customers may get tailored software.
Physical distance to users	Usually small.	Usually large.
Main stakeholder	Customer organization.	Developing organization.
Specific RE issues	Elicitation, modeling, validation, conflict resolution.	Managing a steady stream of requirements. Prioritizing, cost-estimating, release planning.
Developer's association with the software	Short-term (until end of project).	Long-term, promoting e.g. investment in maintainability.
Validation	Ongoing process.	Very late, e.g. at trade affairs.
Use of RE standards and explicit methods	More common.	Rare.
Use of iterative development	Less common.	More common.
Domain expertise available on the development team.	More common	Less common (product development often breaks new ground).

[1] The terms user and customer are here further elaborated compared to Carlshamre (2002).

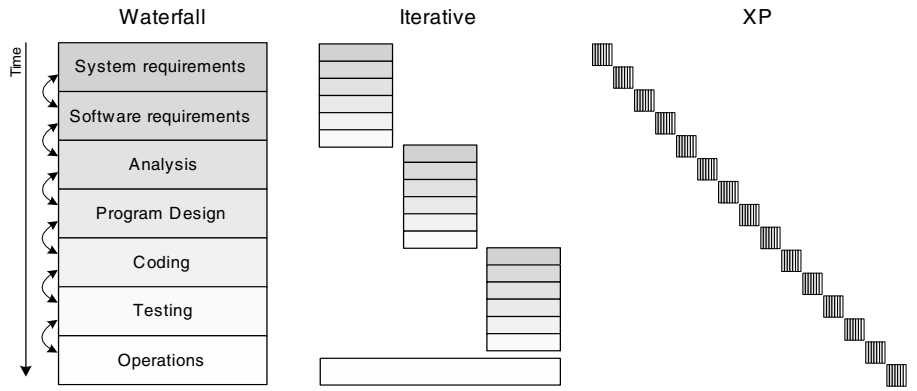


Figure 1.1: Software development process models (Royce, 1970; Beck, 1999)

product life cycle, are very unlike. The differences are so pervading that many traditional requirements engineering practices are unusable for the market-driven company. In the next section requirements engineering is further elaborated, including the specific characteristics of requirements engineering in market-driven software development.

2 Requirements Engineering

The main objective of requirements engineering is to correctly understand the needs of the system's customers or users. During the requirements engineering process, these needs are transformed into a coherent formal specification, which describes what the resulting software system should accomplish. The discipline traditionally stipulates to write an as flawless specification as possible, which essentially means to fulfill the quality attributes listed in Table 1.2 (Davis, 1993).

By accurately describing what the system should do, the requirements specification can act as an agreement, and even as a formal contract, between the customer and the software development organization. Solutions are banned from the specification and, traditionally, it has to be finalized before any successive work in the development process is initiated. This waterfall development approach, illustrated by the leftmost process model in Figure 1.1, was a first solution to the chaotic development in the late 60's and was strongly advocated.

At that time, software was very simple compared to the complex systems developed today. It was possible to specify, once and for all, what the system

Table 1.2: Quality attributes for the software requirements specification (derived from Davis, 1993)

Attribute	Description
Correct	Every requirement represents something required by the system.
Unambiguous	Every requirement has only one interpretation.
Complete	Everything the software is supposed to do is included.
Verifiable	There exists a cost-effective process with which a person or machine can check that the actual as-built software product meets every requirement.
Consistent	No requirements in a given subset within the specification conflict with each other.
Understandable by customers	Requirements should be negotiated in a form that suits the customer or user who usually do not understand formal methods.
Modifiable	Structure and style are such that any necessary changes to the requirements can be easily, completely, and consistently executed.
Traced	The origin of each requirement is clear.
Traceable	The specification is written to facilitate referencing of each requirement.
Design independent	The specification does not imply a specific software architecture or algorithm.
Annotated	The necessity of each requirements is denoted essential, desirable or optional. Volatility is indicated by a textual annotation.
Concise	Given two specifications of the same system, each exhibiting identical levels of all the above qualities, the shorter specification is the better one.
Organized	Requirements are easy to locate.

should do, then design the system, code it, test it and finally deliver it. However, as software complexity has increased, so has the complexity and difficulty of requirements engineering. The goal of accurately specifying requirements and verifying and validating these before actual development starts, requires software developers to be very rigorous.

Unfortunately, the rigorousity is insufficient. Customers and user's are not able to express all their needs at the beginning of the development project. Due to the complexity of software systems, contradictions and changed minds emerge throughout development. As changes to the specification in the later

phases of the waterfall model is extremely costly, the waterfall model has gone through several refinements. For example, shorter development cycles have been reached through an iterative development approach, where the waterfall development approach is repeated in smaller increments. This approach is depicted in the middle of Figure 1.1.

Recently, agile software development (Martin, 2002) has gained a lot of interest in industry. The most known agile development approach, extreme programming (XP), turns the waterfall process sideways (i.e. all activities are performed simultaneously) and repeats it in extremely small increments, as shown to the right in Figure 1.1 (Beck, 1999). The benefits of the waterfall model are utilized, such as its straight-forwardness, while some of the drawbacks are avoided, such as the need for heavy documentation and the lack of support for parallel activities, user involvement, and quick results.

As a consequence of the new approaches to software development, the activities in requirements engineering has changed. One notable difference is that the concept of *the* requirements specification for a particular system is not always meaningful. Requirements are increasingly often stored in a central repository, such as a requirements database, which comprise all requirements ever captured and which functions as a source for decisions on which requirements to implement. In particular, regarding the incremental and evolutionary development models, it has been realized that completeness is sometimes impossible to achieve (Siddiqi & Shekaran, 1996; Goguen, 1996). In extreme programming, which advocates even smaller increments and continuous requirements analysis together with a customer or a representative, the traditional requirements specification has been abandoned to make way for the new concept of user stories, initially specified and documented on plain paper cards.

2.1 Requirements engineering in market-driven software development

Due to its specific characteristics, market-driven development makes it particularly difficult and sometimes unfeasible to comply to all of the requirements specification quality attributes in Table 1.2. There are three main, interlinked reasons.

Continuous elicitation Requirements arrive continuously, throughout development from several different sources, such as the marketing department, usability architects, support, developers, etc. Requirements

may arrive in bursts of thousands at several different occasions or arrive at a rather even pace throughout development, averaging to 3-5 requirements a day (Paper I; Regnell et al. 1998). As a consequence, the requirements repository eventually comprises several thousand requirements (see Paper II and Paper III). Many of these requirements are never selected for implementation. It is therefore not motivated to only speak about *the* requirements specification. At best, there is a coherent repository of requirements, rather than a diverse set of separate requirements specifications. Ideally then, the repository of requirements could be regarded as a set of requirements that should be quality assured. But, as a large amount of requirements are elicited and invented that are never selected for implementation, it is not feasible to pursue such an endeavor for the whole requirement set.

Time-to-market constraints The time-to-market constraint forces the development organization to implement requirements before all quality attributes have been properly checked. Several requirements may at an early stage have been found to bring competitive advantage and are therefore selected for implementation. When development of a new release is initiated, there is no complete requirements specification. Deliberately, implementation is started before every requirement has been selected. This situation makes a quality attribute such as 'completeness' impossible to assure before the release of a new version of the software product. In fact, it is virtually impossible to know when analysis is complete. It is always possible to do more analysis and improve the requirements. In market-driven development the completion criterion is, unfortunately to a large extent, the date.

Market competition Competition in the market place forces the developing company to accept changes to the selected requirements throughout the development cycle. As opposed to contractual development, the risk is on behalf of the developing company. Decisions on which requirements to include in the next release must be made early in the development, based on the information available. Later on in development, these decision may be found incorrect or outdated. Competitor analysis and market strategies may call for changes to the requirements that were initially selected. Constraints put by the hardware, which often must be decided upon early in the development process, may also require changes to the requirements. For the companies to stay ahead

of competitors, new versions of the software have to be released as soon as there is a major improvement available. Often it is required to release new versions more frequently than it is, with acceptable quality, possible to develop.

Market-driven requirements engineering is thus a continuous process. After an initial version of a product has been released, there is a need for a dynamic process of elicitation, analysis, and prioritization (Paper I; Regnell et al. 1998). The consequence is a constant change and evolution of requirements, which is a general particular challenge in software development (Jones, 1996). The origins to changing requirements have been identified by Parker et al. (1993):

Environmental turbulence Markets change, materials and means of production offer new opportunities, government policies and legislation may be changed, and different structures and management practices are employed. The goals are not likely to be the same at the time development starts and at the time of delivery.

Stakeholder engagement in requirements elicitation At the start of development, stakeholders find it very difficult to produce a complete set of requirements. It takes time to formally identify stakeholders' goals, understand how technical opportunities may serve the goals, make stakeholders agree, resolve priorities and conflicts, etc.

System use and user development As a system is delivered and put in use it may stimulate new ways of working. This may generate new and changing requirements. Pilot systems, prototypes, demonstrations, and scenarios may stimulate the stakeholders during development. The origin of change would in that case be stakeholder engagement in requirements elicitation.

Situated action and task variation The performance of a system is matched to the specific task conditions, existing at the time. Therefore, systems must be made sufficiently flexible to permit customization and personalization. A certain set of requirements are requirements for change to be an inherent and on-going capability of the delivered system.

Constraints of planned organizational development Both organizational process and existing systems put constraints on the requirements. As the

processes change and existing systems are replaced, affected requirements must be systematically enhanced or replaced.

To manage changes to requirements there is a distinguishable process within requirements engineering, *requirements management*, that aims at understanding and controlling change. In addition to managing change, requirements management comprise concerns for requirements storage and traceability issues (Sommerville, 2001). A European survey of 4,000 companies has shown that management of requirements was one of the major problem areas in software development (Kotonya & Sommerville, 1997).

The challenge of managing change is generally difficult because requirements most often are written and communicated in natural language (Mich et al., 2004). In market-driven requirements, the huge amount of requirements makes it particularly difficult. After years of rewarding research that has helped us understand and improve the way requirements may be specified and formulated, the state of the practice is generally that such guidelines are rarely applied. There is a large gap between the formal models advocated by many researchers and the informality that dominates in industry. Several reasons can be identified to why requirements are initially specified in natural language and in many cases kept in that form throughout the development process:

- Natural language is our primary communication language that is shared by all stakeholders and participants in the development process. Formal languages require specific training, which is unrealistic to expect from every stakeholder in particular customers or end users.

- Requirements engineering is a social and evolutionary process where requirements are elicited and specified at different levels of abstraction, at different points in the development process.

- Natural language is universal, meaning that it can be used to talk about arbitrary domains and at arbitrary levels of abstraction. Few formal languages, if any, have this strength.

- As mentioned above, in market-driven development there are comparatively few of the proposed requirements that are actually selected for implementation. Since not all requirements are expected to be implemented, there is little motivation for spending time formalizing them. In particular, experience tells us that companies, which value close interaction with their customers and rapid reaction to changing market

conditions, do not find it cost-beneficial to translate all requirements into formal specifications.

- Many formal methods do not offer any support for the management and analysis of erroneous, incomplete, or partially-specified requirements. In contrast, natural language techniques adapt naturally to such situations, that in practice make up a large part of a requirement life cycle.

- While formal languages can improve our ability to check internal consistency and completeness of requirements (a process often referred to as verification), they can not capture external properties of the requirements, e.g. correspondence between the requirements and the actual user intentions. It requires good communication and interaction with the stakeholders to verify such properties (validation) – and to this end, natural language is a more suited language.

So, despite its recognized and infamous deficiencies there are few incentives to avoid natural language. We should therefore expect that natural language use can not be escaped. This is also noted by Jackson, stating that requirements engineering is where the informal meets the formal (Jackson, 1995). Hence, the gap between the users' needs and a new release of the software system must be bridged using methods and techniques that acknowledge, in some form, communication in natural language.

The challenges of managing enormous amounts of requirements, written in natural langauge, that continuously must be analyzed, reanalyzed and consolidated is still generally left untouched. This is partly reflected by current requirements management tools. The tools do provide the functionality to store huge volumes of requirements, define different views of requirements, and assign links between requirements. However, they do not give appropriate assistance in the actual matching of thousands of incoming requirements with those already analyzed. Requirements management tools could do better than providing simple keyword search facilities to alleviate the manual burden of consolidating large amounts of requirements.

Companies facing these challenges may arrive at a cross-road where the choice is to reduce the flow of incoming requirements or to assign more resources to handle them (Paper I). However, seen from a business perspective, neither of these approaches is particularly rewarding (and in many situations impossible). Choking the elicitation and invention of new requirements may increase the risk of missing potential business opportunities (Kristensson et

al., 2002), and adding more people to do the job has been shown to be too costly and, at times, counter-productive (Paper I; Brooks 1975/1995.

In essence, these companies face an information overload problem and there is a strong need for more supportive information management tools, aimed at the management of requirements written in natural language. The approach suggested in this thesis is to apply linguistic engineering techniques.

3 Linguistic engineering

Language processing techniques emerged during the Second World War when computers were utilized to break message codes (Jurafsky & Marting, 2000). Since then, a number of overlapping fields has emerged: computational linguistics, natural language processing (NLP), text and information retrieval, speech recognition, artificial intelligence, natural language understanding, and computational psycholinguistics. The common goal for all these fields is to enable computer systems to perform tasks involving language processing.

The goal has partially been fulfilled. There are several examples of where computers are successfully used to process information, whether it is written or spoken. Typically, the systems either do shallowing processing across a broad range of data (e.g. search engines of various kinds) or they do detailed processing across a narrow range of data (e.g. specialized automated information services over the phone).

Still, computer analysis of speech and text remains complicated and there is a long way to go before a computer system can mimic human processing in such a way that it is indistinguishable from that of a human. The Loebner Price, set up in 1991, is to be awarded to the first author of a computer program to pass an unrestricted Turing test (Turing, 1950). The principle of the test is that if a computer's responses are indistinguishable from those of a human, the computer could be said to be thinking. The prize has not yet been awarded.

In 1992 the term Linguistic Engineering was described in the Technical Background Document for the Linguistic Research and Engineering (LRE) Programme:

> *"Linguistic engineering (LE) is an engineering endeavour, which is to combine scientific and technological knowledge in a number of relevant domains (descriptive and computational linguistics, lexicology and terminology, formal languages, computer science, software engineering techniques, etc.). LE can be seen as a rather pragmatic*

> *approach to computerised language processing, given the current inadequacies of theoretical Computational Linguistics."*
>
> (from Garigliano, 1995)

The LE approach is elaborated by Garigliano (1995), who points out a range of criteria for applied systems dealing with natural language, such as usability, robustness, flexibility, and efficiency. The criteria expose the possible variation points for the usefulness of an NLP-based system. In essence, it is a matter of systematic cost-benefit analysis. The long-term objective is to produce automatic systems that are of valuable use in specific environments.

Irrespective of the approach taken to develop systems that process natural language, there are a set of traditional linguistic concepts that need to be defined in order to understand the different levels of analysis required by an automatic system:

Morphology The components of words. It involves the understanding of inflections, derivations, and the formation of compound words.

Syntax The structural relationships between words. It involves the understanding of how different words (i.e. nouns, adjectives, verbs, etc.) are combined into clauses, which, in turn, are combined into sentences.

Semantics The meaning of words, phrases, sentences, and texts. It is often contrasted to syntax, as it is possible to break the rules of language by producing a grammatically correct sentence, which is semantically anomalous (e.g. "Colorless green ideas sleep furiously" (example from Chomsky, 2002)). However, the form of a statement in natural language cannot be analyzed separately without reference to its meaning. Jackson & Moulinier (2002) illustrates this point by comparing the two sentences "She boarded the airplane with two suitcases" and "She boarded the airplane with two engines". The suitcases belong to the woman, while the engines belong to the airplane. These kinds of ambiguities typically cause problems to computer programs.

Pragmatics How language is used to accomplish a goal. This involves the understanding of how the context influences the interpretation. One example, often used to illustrate how men and women communicate, can also function as an example of the distinction between the meaning of the sentence and the meaning of the speaker. A man and a woman are traveling across the country. The man is driving and the woman is

watching the scenery. After hours driving, the woman notices a coffee house by the road and utters "Oh, what a nice coffee house". The man politely responses "Yes, it certainly is" and keeps on driving. But perhaps the woman did not only assert the fact that the coffee house was nice, but also communicated a request for taking a break by the coffee house.

The above aspects of language present challenges for any computerized processing of natural language text (as far as requirements are concerned, pragmatics and semantics even presents challenges to human beings). In general, there are two approaches available to handle the challenges. One approach has its roots in linguistic analysis and typically consists of rules for manipulating the input text, e.g. grammar rules. The approach is therefore often characterized as *symbolic* (Jackson & Moulinier, 2002). The other approach, gaining increased interest in the 1990s, has its roots in statistical analysis. It involves a quantitative approach to automated language processing, including such fields as probabilistic modeling, information theory, and linear algebra (Manning & Schütze, 2002). It is often characterized as *empirical* (i.e. based on experience and experimentation) as language data is derived from large text corpora, such as news feeds and web pages.

Most characteristically, the two approaches handle phenomena such as ambiguity differently. The symbolic approach resolves ambiguity by adding another rule (which somehow must be formalized). This approach relies on human experts that are able to identify and describe the language constructs. The empirical approach uses mathematical models and statistical methods to associate probabilities to and decide among different alternative analyzes.

For most processing systems, the approaches are combined in order to reach the best result. So is also the case for the research and tool presented in this thesis. Symbolic approaches are utilized for morphological analysis, while statistical approaches are used for dealing with semantics of requirements. In section 5 the approach taken and the techniques used in this thesis are further elaborated.

4 Research in applying linguistic engineering to RE

As pointed out in the well-referenced paper by Ryan, there have been many unrealistic expectations on natural language processing techniques given the desire for a system that could support the currently expensive activities within RE (Ryan, 1997). These expectations are typically based on misconceptions

about what the communication problem in industrial RE really is and to what extent the requirements on a system are available in textual form (see for example Sutton (2000) for a discussion on linguistic problems with requirements elicitation).

The criticism taken into account, Ryan concludes that RE is a social process and that linguistic techniques can succeed only in a supporting role to this process – not by trying to replace it. Today, there are likely few researchers in RE that would argue against this statement.

To relate this work to the current body of knowledge, presented here is a survey of research aimed at supporting RE activities using linguistic engineering techniques, grouped by three major RE process activities addressed:

Domain and requirements understanding, which is a fundamental success factor in all systems and software development.

Requirements verification and validation, which are carried out to ensure that a specification is internally consistent and to certify that the requirements are a correct representation of the users' intentions (Boehm, 1984).

Requirements management, dealing with storage, change management and traceability issues. This is within the scope of this thesis.

In many cases the industrial applicability and scalability is yet to be determined through larger case studies with real data. Also, although most approaches acknowledge ambiguity and inconsistencies, it is seldom reported how any other pollution in the data is treated (e.g. misspellings and non-information carrying characters). A combination of different techniques would likely be the most rewarding and the research surveyed provides a basis for this acquisition.

4.1 Domain and requirements understanding

Comprehension of the application domain is a fundamental success factor in all systems and software development. Comprehension of the domain is also an essential pre-requisite for understanding the requirements. Naturally, domain understanding and requirements understanding complement each other. Through requirements elicitation domain understanding improves, which in turn matures the understanding of all the requirements. Therefore,

domain and requirements understanding are key activities in the general requirements analysis process. One outcome of these activities is the improved ability to understand the users' needs and requirements to pursue.

A central task in domain and requirements understanding is to identify and understand domain concepts, also called domain abstractions. Domain abstractions are general concepts that are formed to represent common features of specific instances in the domain. Although detailed information is left out, abstractions make communication more efficient within the domain. Developers of a software system that shall give support within a domain must however take into account not only the general concept, but also the specific instances, in order to fully understand the abstractions. Therefore, it is of great value to identify and investigate the domain abstractions further as they may support domain and requirements understanding (Goldin & Berry, 1997).

Domain abstractions are typically represented in natural language through sets of terms (often nouns and noun phrases). Researchers have therefore investigated linguistic engineering techniques to extract these terms, representing the abstractions, from the discourse generated from interview transcripts and customer wishes expressed in natural language. Following is a survey of the major research efforts addressing abstractions.

Goldin & Berry (1997) present an original approach and a prototype tool for suggesting requirement abstractions to the human elicitor. Their method compares sentences using a sliding window approach on a character-by-character basis and extracts matching fragments that are above a certain threshold in length. The approach can properly handle arbitrary lengths, gaps and permutations and avoids some specific weaknesses in confidence and precision when using only parsers or counting isolated words.

Rayson et al. (2000) report experiences from one of the rare projects where probabilistic NLP techniques have been used. The authors present two experiments using tools they have developed (part-of-speech and semantic taggers integrated into an end-user tool). The experiments suggest that the tools are effective in helping to identify and analyze domain abstractions. This is further supported by a later study by Sawyer & Cosh (2004) where ontology charts of key entities are produced using collocation analysis.

A different approach to requirements understanding is taken by Gervasi (2000) who uses lexical features of the requirements to cluster them according to specific criteria, thus obtaining several versions of a requirements document. The sectional structure of these documents and the ordering of re-

quirements inside each section, are optimized to facilitate understanding for specific purposes.

4.2 Requirements verification and validation

While requirements verification is carried out to ensure that a specification is internally consistent, validation is performed to certify that the requirements are a correct representation of the users' intentions (Boehm, 1984). Naturally, validation and verification of requirements are key activities in requirements analysis. The ability to succeed in these activities also builds upon the domain understanding discussed earlier and it is generally acknowledged that spending more time in these stages and finding errors early is more rewarding than proceeding too soon to coding (Boehm, 1976; Daly, 1977; Davis et al., 1997). Therefore, considerable research effort has been put applying natural language processing to requirements verification and validation. The typical approaches taken are:

- to create models from the textual representation, or
- to develop indicators of violation of specific rules.

The two approaches generally involve some kind of parsing, where the input text, very often manually accommodated, is transformed into a representation which enables modeling and identification of semantic and syntactical incorrectness.

Requirements verification and validation are not carried out separately. Checking a set of requirements for consistency may reveal internal inconsistencies that may as well be external. These inconsistencies must also be resolved with a stakeholder. Therefore, these two activities are here addressed together.

Gervasi & Nuseibeh (2002) treat the validation as a decision problem on whether a given software model, synthesized by parsing the requirements text, satisfies certain properties. They present an experiment with the use of lightweight formal methods. Through an eight-step process, models are built and the violated properties are reported. The steps in the set-up phase are manual but expected to be reusable (defining requirements style, structure and language; selecting properties; defining models; building domain glossary). The steps in the production phase are mainly automatic (preprocessing the requirements text; parsing; modeling; validation). The experiment shows that even subtle errors, not discovered by human inspection, may be identified.

Cybulski & Reed (1998, 1999) describe an elicitation method and a supporting management tool that help in analyzing and refining requirements by using a parser, semantic networks, a domain-mapping thesaurus, and faceted classification schemes to allow requirements formalization. The natural language components are used to force the requirements engineer to rephrase requirements in order to unify the terminology. Their method puts an emphasis on reuse of requirements specifications to further aid the refinement.

Another way of generally improving the quality of written requirements is suggested by The Goddard Space Flight Center's Software Assurance Technology Center (SATC) (Wilson et al., 1996). They have derived seven quality indicators used for measuring the quality of requirements specifications. A list of generally accepted quality attributes, found in the literature, was related to objective and quantitative measures, in order to derive primitive indicators of the specification quality. The list of indicators was also improved by investigating the term usage in 46 requirements specifications from different NASA projects. They have developed a tool that reports the quality of requirements specifications and which is also used by NASA to improve their requirements specifications. They conclude that automatic processing can give insight into the specification quality and improve the effectiveness of expressing requirements specifications in natural language.

Fabbrini et al. (1998) also propose a quality model for natural language requirements. They focus on the linguistic properties of requirements documents and match factors from a linguistic quality framework (deduced from a quality framework for conceptual modeling) to NLP techniques. The result is a quality model that proposes linguistic criteria that may be used for defining quality profiles and selecting appropriate tools for writing, verifying and validating requirements. Fabbrini et al. (2001) have also implemented a tool, based on the quality model, to show the quality model's applicability for quality assuring industrial requirements. Fantechi et al. (2000) has applied both the tool by Fabbrini et al. and SATC to evaluate the quality of 100 use cases. They conclude that although the technique may support quality evaluation, it is not sufficient to completely address correctness and consistency.

Yet another way to adapt the language to formal validation is to explicitly restrict the language used in requirements; an approach that has been proposed by different researchers. Fuchs & Schwertel (2003) have over the years advocated their subset of English that unambiguously can be translated into first-order logic and checked for inconsistencies. The advantage is that it can be used by domain specialists that want the benefits from formal languages,

but who lack the required training.

Cyre & Thakar (1997) define syntax and grammar of restricted English so that requirements may be parsed and semantically analyzed to generate conceptual graphs. The validation of the conceptual models is specifically aimed at resolving ambiguities.

Somé, Dssouli, & Vaucher (1996) goes one step further and restrict the language and semantics to a scenario style, albeit more understandable by the user than formal specification. Complete requirements specifications are then automatically generated from these specification.

Osborne & MacNish (1996) suggest using extensions to a parser with a wide-coverage grammar in order to identify and present syntactic and semantic ambiguities to the requirements analyst. The extensions include a parse selection mechanism, an error diagnostic facility and resource bounds, which together effectively present the five most plausible parses of a requirements sentence. The feedback from the parser is aimed to be used by a requirements analyst to successively refine requirements so that sentences conform to a controlled, restricted language.

Towards formalization, Fliedl et al. (2003) suggest the use of a conceptual pre-design model to bridge the gap between the natural language representations and enable formal validation. The pre-design model is not as technical as common conceptual modeling languages, while still supporting the general principles behind several different conceptual models (e.g. use cases, state charts, etc.) and the mapping to more formal model. This enables validation with both designers and stakeholders in a language closest to their preferred one.

Burg (1996) have developed a comprehensive and sophisticated approach and a supporting environment for specification, verification, and validation of functional requirements. Through an integration of natural language and scenario analysis, two conceptual models are derived, one static and one dynamic. Verification is then supported graphically, lexically, and logically, while validation is supported through natural language paraphrasing (transforming models into language readable by the user or customer) and simulation of the dynamic behavior. In several different ways they show how the approach enhances specification properties and aspects.

Rolland & Proix (1992) describe a system prototype that aims to provide support to problem-statement acquisition, elicitation, modeling and validation. Acquisition of problem statements is performed using a specialization of the Fillmore's case notion (Fillmore, 1968). Conceptual schemas are then

generated through a number of mapping rules from the cases to nodes and arcs. Feedback in (French) natural language is finally generated through a three-step process (extraction, transformation and linearization), each dependent on a set of rules.

Macias & Pulman (1993) also pursue the extraction of conceptual models from natural language requirements. The system they describe uses a well-defined subset of English to forbid the expression of potentially ambiguous sentences, and can translate natural language sentences written in this reduced language into a formal representation, and vice versa, by using a general-purpose, domain-independent NLP engine. In addition to paraphrasing, the system supports natural language queries to the formal model, with answers also expressed in natural language. In related work (Macias & Pulman, 1995), the same authors present a syntax-driven text editing interface that helps prevent the creation of syntactically incorrect requirements (with respect to general language constructs).

Nanduri & Rugaber (1996) use OMT (object modeling technique) guidelines and a link grammar parser for transforming high level specifications parsing into object charts. Although their tool produces object diagrams that may help identify omissions, the approach suffers from several common problems when trying to transform natural language requirements into object models: parser limitations, ambiguity, incompleteness, insufficient domain knowledge and not enough transformation rules.

A similar approach is taken by Mich & Garigliano (2002) who have developed a tool to support object-oriented analysis. The natural language requirements are processed using an integrated, but separately developed, large-scale natural language processing system. Modeling is then performed based on the output and the knowledge stored in the natural language processing system. Its performance is demonstrated using the ATM problem statement in Rumbaugh et al. (1991).

Park, Kim, Ko, & Seo (2000) present an implementation of a requirements-analysis supporting system, which may help to identify potential errors in requirements. Conflicts and inconsistencies are identified by measuring similarity between documents and between sentences. Also, ambiguity is identified by matching the result from part-of-speech tagging with a manually constructed word set that indicate ambiguities (in line with the ideas behind the linguistic quality model by Fabbrini et al. (1998), see above). Their approach to combine syntactic parsing with a sliding window method gives more accurate similarity measures than using them separately.

In a recent paper, Flores (2004) proposes to use NLP techniques to extract relevant sentences from large requirements corpora. The approach uses shallow parsing and contextual exploration networks, based on the presence of certain textual markers in the text, to assign semantic tags to parts of sentences without making recourse to large and unwieldy knowledge bases. Four viewpoints are considered: concept relationships, aspecto-temporal organization (e.g. events and ongoing processes), control (e.g. by machine or user), and causality. The proposed approach seeks to identify inconsistencies in the source documents by reasoning on models built on top of these four viewpoints.

The identification of inconsistencies is also the main goal of Gervasi & Zowghi (2005), who have used natural language parsing and default reasoning techniques to identify several inconsistency-related problems in requirements. In particular, conflicts between different stakeholders (both explicit and implicit) are addressed. The authors also describe a prototype tool implementing the proposed technique, and compare its performance to that of 15 human experts.

4.3 Requirements Management

Surprisingly, there are, beside the approaches presented in this thesis, not yet any specific attempts that directly try to tackle large-scale requirements management challenges by using natural language requirements processing. In particular, the following specific hands-on requirements management activities are open for scrutinized research:

- Match incoming (potentially new) requirements to previously elicited, planned, and already implemented requirements.
- Maintain a separation and find the relationships between customer requests and requirements invented within the organization.
- Identify dependencies and other interrelationships between requirements.
- Support the extraction of requirements from the repository that fit strategic areas, i.e. areas that are of specific importance to the company (e.g. invoicing capabilities, decision-making features).

The investigations presented in this thesis show that these activities are major obstructions in the efficient management of elicited, invented and implemented requirements. Any technique that may support the above require-

Table 1.3: Listing of some similarity measures

Similarity measure	Description
Semantic	Similarity in meaning
Syntactic	Similarity in grammatical structure
Lexical	Similarity in words used
Structural	Similarity in sectional structure
Extensional	Similarity in size
Argumentative	Similarity in rationale
Goal	Similarity in objective
Source	Similarity in the proponent
Function	Similarity in function addressed
Object	Similarity in system parts affected
Temporal	Similarity in time of origin

ments maintenance and management activities, even if partially, can be expected to be warmly accepted in industry.

The approach taken in the research presented in this thesis is to calculate similarity between requirements in order to support the activities above.

5 Requirements similarity

The research in this thesis suggests that a number of problems in the management of large volumes of requirements can be solved or at least alleviated by using a measure of how similar two requirements are. Naturally, many different notions of similarity can be used. In most problems, what is needed is a notion of semantic similarity: a measure of whether two requirements convey the same meaning, and to what extent this meaning is similar. However, other notions of similarity can also be used. A few of these are listed in Table 1.3 and more measures can easily be obtained by considering other metadata about the requirements (e.g., priority assigned, system version targeted, approval responsibility, implemented status, etc.).

Whatever measure is chosen, in order to be applicable to the management of large repositories, it must possess a fundamental property: it has to be computable in a relatively inexpensive way. Any measure requiring significant human intervention will be too costly to be used on large requirement repositories; we are thus forced to focus on similarity measures that can be computed in a completely automatic way.

Unfortunately, given the current state of the art in natural language pro-

cessing and in knowledge representation, it is not feasible to extract meaning in a reliable way from totally unrestricted natural language text as that found in most requirements. This work therefore focuses on lexical similarity as a way of approximating semantic similarity.

On a lexical level, requirements are considered as a sequence of words. The exact definition of what a word is varies with the language and the application. More refined approaches distinguish the various lexical (and at times, morphological) constituents of requirements with more precision. E.g., punctuation (as in ","), contraction markers (as the apostrophe in "can't"), parenthetical structures (as "(") etc. can be considered as words on their own. The process of separating the lexical constituents of a requirement is referred to as *tokenization*, and each word (in this extensive definition) is called a token.

In the work presented, a token is regarded as a sequence of letters and/or digits. Any other characters are regarded as delimiters and thus discarded. Tokens can be further processed in various ways. Most typically, tokens are reduced to their base form, removing morphological inflections (e.g., reducing plural nouns to their singular form, or removing person, mood or aspect information from verbs). This process is called *morphological analysis* and is performed with the help of linguistic rules and lexicons. One variant of morphological analysis is called *stemming*, which associates variants of the same term with a base form. Many applications use a heuristic stemmer, which uses 'rules of thumb' instead of linguistic rules (by removing affixes, such as un- and dis-, or suffixes, such as -ing, and -able). In Paper II, the well-known heuristic Porter (1980) stemmer is used, but in Papers III, IV, and V, a newer one is used that is reported to perform better (Minnen et al., 2001).

Another common operation is *stop word removal*. It consists in dropping, from the sequence of tokens, all those words that have a purely grammatical role and are no significant indicators for content. The grammatical information they convey may be stored in some other form (e.g., in parsing trees) before removing the stop words, if so desired. Again, the details of the process depend on the language at hand, and on the kind of analysis that is to be performed on the requirements. In most cases, stop words coincide with so-called closed class words, e.g. articles and prepositions. Also in this case, a special-purpose dictionary can list exceptions. In the presented cases we have used a stop word list comprising 425 words derived from the Brown corpus (Francis & Kucera, 1982).

Further preprocessing steps are possible (e.g. part-of-speech tagging), but for calculating lexical similarity the steps described above are sufficient. For-

mally, a requirement r, taken from a requirement set $\bullet$, may be considered as a finite sequence $r = \langle v_{i_1}, v_{i_2}, \ldots, v_{i_n} \rangle$ of tokens drawn from a given alphabet $V = \{v_1, v_2, \ldots, v_n\}$, which includes all the tokens that appear in our requirements database.

Using the pre-processing steps described above, V would contain stemmed tokens that do not appear in the stop word list. If order is not considered important, an alternative representation is possible: a requirement r can be considered as a vector $a_r = [w_r(v_1), w_r(v_2), \ldots, w_r(v_n)]$, where $w_r(v_i)$ denotes the weight, or relative importance, of the token v_i in requirement r.

Different weighting schemes are possible. As requirements expressed in feature style (Lauesen, 2002) are more focused than literary text, it is assumed that the tokens remaining after the preprocessing step are all equally valuable. In Paper II, the simplest weighting scheme is applied, assuming that weight coincides with frequency. However, as it is considered that the importance of a token is not linearly proportional to the number of times it occurs, in Paper III, IV, and V, the well-known weighting formula $1+log_2(termfrequency)$ is also used (Macias & Pulman, 1993). Paper III explicitly compares the results obtained by using the two schemes.

Once requirements have been encoded as vectors, it becomes possible to apply standard similarity measures. In Paper II the performance the Dice, Jaccard, and Cosine measures (Manning & Schütze, 2002) are compared. The most significant difference between the measures is how they treat different lengths of the compared requirements.

In Paper III, IV, and V, the Cosine measure was selected as it was considered to generally perform better than the other two. The measure got its name from calculating the cosine of the angle between the vectors that represent the requirements in a vector-space model. Figure 1.2 depicts this representation in the case where there are only three tokens in the token space. Two of the tokens are used in r_1 (blue, button) and two are used in r_2 (red, button). However, the token 'blue' is occurring twice as many times as 'red' and the vector representing r_1 is therefore pointing further away from the 'button' axis.

Formally, given two requirements, p and r, the similarity between p and r is given by the formula in Equation 1.1 (an example of applying the measure can be found in Paper IV).

$$\sigma(p, r) = \frac{w_p(v_i) \cdot w_r(v_i)}{\sqrt{\sum_i w_p(v_i)^2 \cdot \sum_i w_r(v_i)^2}} \tag{1.1}$$

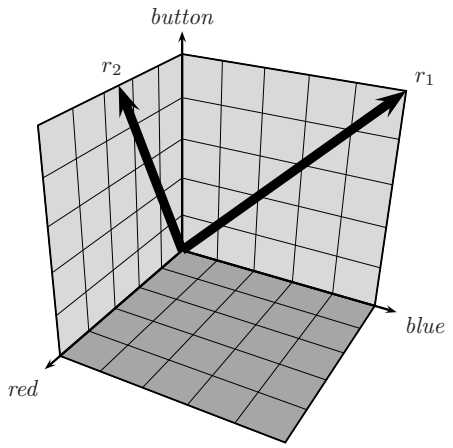


Figure 1.2: Vector-space example

The definition assumes that the vector space employed has a Euclidean distance, uniform across all dimensions. This is of course a gross oversimplification; in practice, the presence or absence of certain terms may be much more important and revealing of true semantic similarity than that of other terms. However, since the main interest is in techniques that work irrespective of the exact domain and language used, and for the sake of generality, this simplification may be accepted, keeping in mind that more refined techniques can be employed in specific domains.

The cosine measure in Equation 1.1 returns a value between 0 and 1 to indicate how similar two requirements are. This value may reflect more or less correctly the actual semantic similarity. In order to evaluate how well the measures perform, the automatically calculated similarity measures must be compared to the true similarity.

Human experts prefer to determine the similarity between requirements on a nominal scale (e.g. identical, similar, not related) in favor of a ratio scale. Therefore, the ratio values must be mapped to the nominal values. One way to do this is to define thresholds. For example, similarity values equal to or greater than 0.8 may correspond to identical requirements, similarity values between 0.2 and 0.8 may correspond to similar requirements and values equal to or less than 0.2 may correspond to dissimilar requirements. It is then possible to use the widely adopted measures of *recall* and *precision* to evaluate the performance of the similarity measures.

Recall is the fraction of requirements that are correctly classified by the system. E.g., if a system correctly classifies 100 of 1,000 known relationships, the recall will be 10%.

Precision is the fraction of the classified requirements that the system got right. E.g., is a system classifies 500 relationships, of which 100 are correct, then the precision is 20%.

Although these measures give a proper indication on how well an automatic system performs, their interpretation is not a trivial task. Usage and interpretation are dependent on the application and the final judgement is subjective (e.g., how good is a system that reaches 85% recall?). Typically, recall and precision co-vary and higher recall may be reached at the expense of precision. Measures that combine recall and precision into one measure have been suggested, but merely transfers the problem into a question of how much more important recall is over precision (Van Rijsbergen, 1979).

Due to the need to interpret the measures in the context of their use, the definitions, usages, and explanations of the measures used in this thesis are found in the papers.

6 Research methodology

The researcher in software engineering typically seeks better (e.g. more efficient, faster, less cumbersome, etc.) ways to develop and evaluate software of acceptable quality. Motivated by real world problems, solutions are sought that are also applicable to the real world.

The research presented in this thesis has mainly been conducted using an engineering approach, where situations have been observed and better solutions have been proposed and evaluated. One fundamental goal has guided the research presented in this thesis and is based on the background presented in the previous section and on the findings in Paper I. This goal may be articulated as follows:

> *To find automated support to the manual work in the management of natural language requirements in large-scale software development.*

The goal and the research focus has been used to formulate relevant research questions, of which the ones addressed in this thesis are found in Section 7. With the research questions as the guide, research projects have been designed using both fixed and flexible design strategies.

In a *strict* fixed strategy, which is also referred to as the quantitative approach, the design is finished before data collection begins and the data collected is usually in the form of numbers. In contrast, the *strict* flexible design, also referred to as the qualitative approach, evolves during data collection and usually involves collection of non-numerical data (Robson, 2002).

The fixed and flexible design strategies may be further classified. It is virtually impossible to cover for all possible forms of enquiry, but the following research strategies are widely recognized (Robson, 2002):

Fixed design research strategies

Experimental strategy A small number of variables are measured and others are controlled. The researcher actively and deliberately introduces some form of change in the situation, circumstances or experience of participants with the view to producing a resultant change in their behavior. In Section 6.1, experimentation as a research method is further elaborated.

Non-experimental strategy A small number of variables are measured while others are controlled. The research does not try to change the situation, circumstances or the participants' experience. This strategy involves the use of observations and surveys as data collection methods.

Flexible design research strategies

Case study A single 'case' or a small number of related 'cases' are studied to develop detailed, intensive knowledge. The study is made in the context of the case. In Section 6.1, the case study as a research method is further elaborated.

Ethnographic study How a group, organization or community live, experience and make sense of their lives and their world is captured, interpreted and explained.

Grounded theory A theory is generated from data collected during the study.

Research in software engineering is young and the subject is cross-disciplinary. Therefore several research approaches and methods have been adopted from other fields. Attempts have been made to characterize research in software engineering but the picture is still not as clear as in many other, more mature research fields (Shaw, 2002). Thus, software engineering research

methodology consensus is still to be reached. A part of the problem lies in defining the boundaries of the field, which differ from typical engineering research that builds on clearly defined scientific principles.

The particular purpose of the research presented in this thesis and the way it is conducted suggest that it may also be classified both as *evaluation research* and *action research* (Robson, 2002). In evaluation research the effect and effectiveness of something is assessed. Both fixed and flexible design may be used, with either qualitative or quantitative methods. In action research the goal is to influence or change some aspect. Both classifications apply to the research in this thesis, which is also revealed by the underlying goal expressed earlier.

6.1 Research methods

Depending on the specific research methods chosen, research may fit more or less accurately to a specific methodology or strategy. There are a vast number of research methods available and the ones to choose is dependent on the type of information that is sought (Robson, 2002). Following is a review of the methods used for the research presented in this thesis.

Case study

The case study research method is used "when the phenomenon under study is not readily distinguishable from its context" (Yin, 1993). This implies that the the case always occurs in a specified social and physical setting. The case itself can be virtually anything; a person, a business, an organization, an innovation, a tool, a service, etc. Thanks to the flexibility of the case study method, it has been used for several years across a variety of disciplines.

Of course, the complexity of real-life events naturally presents challenges to the researcher as there are more variables than there are data points. There is also the difficulty of generalizing the results, as the richness in each case calls for multiple sources of evidence. There are some critics who suggest that findings are easily biased due to the intense exposure to the study of the case. Some even dismiss the case study method as useful only as an exploratory tool. These issues must naturally be taken into consideration when designing and planning a case study. As with other research methods, reliability and validity must be accounted for.

Important to understand, however, is that the case study approach is a fundamentally separate research strategy with its own design (Robson, 2002).

There may be exploratory case studies, explanatory case studies, and even descriptive case studies. Moreover, the method does not prescribe the use of any particular data collection method (Yin, 1994). Interviews, observation, and data archives may all be used to provide quantitative and qualitative data to support empirical investigations into complex real-life events.

To assure validity and reliability of the case study, Yin suggests three principles of data collection (Yin, 1994):

Principle 1: Use multiple sources of evidence The case study becomes more convincing and accurate if several sources of information points to the same conclusions. Triangulation may be performed on data sources, among different evaluators, of perspectives on the same data, and of methods.

Principle 2: Create a case study database The collected data should be organized, collected, and presented in a way that allow inspection of the source of evidence that led to the conclusions. This may be done by (a) provide the data or evidentiary base, and (b) present the research in a report.

Principle 3: Maintain a chain of evidence An external observer must be able to trace the steps, in either direction, from initial research questions to the case study conclusions.

The papers in this thesis present different case studies that together form the argument of adopting linguistic engineering techniques to support requirements management tasks. Principle 1 has been followed by both using different data collection methods and different data sets. As the real industrial requirements are proprietary information of each company, the evidentiary base cannot be provided publicly as suggested by principle 2a. However, principle 2b and principle 3 have been followed by presenting, in each paper, the line of work as clearly as possible to enable researchers to conduct appropriate replications with their own industrial requirements. This thesis introduction further contributes to the chain of evidence.

A further classification of the methods and strategies used may be found in Section 8. A discussion of validity threats in general is found in Section 6.2, and the specific threats to the validity of the presented research are presented in Section 8.1.

Process simulation

Simulation may be applied to a vast number of areas to imitate the operation of a real-world process or system over time (Banks et al., 1996). It may be executed either by hand or by computer to generate artificial historical data, which is observed to investigate the plausible behavior of the real system. The behavior is studied by developing a simulation model, which describes the system through mathematical, logical and symbolic relationships between objects in the system that are of interest. A useful model always simplifies and idealizes, and the boundaries between the system and the model are rather arbitrary defined. However, the usefulness is dependent of the possibility to practically determine all its relevant behavior: analytically, numerically, or by running the model with certain inputs and observe the outputs (Bratley et al., 1987).

The *purposes* of simulation are many (Naylor et al., 1966, pp. 8–9; Banks et al., 1996, p. 4):

1. Enable the study of, and experimentation with, the internal interaction of or within a complex system.
2. Simulate informational, organizational and environmental changes and observe the effects of alterations.
3. Provide knowledge from designing a simulation model that may be of great value towards suggestion of improvements to the system.
4. Obtain insight into the questions of which variables are most important and how variables interact.
5. Pedagogically reinforce analytical solution methodologies.
6. Experiment with new designs or policies prior to implementation, so as to prepare for what may happen.
7. Verify analytical solutions.

The appealing property of simulation, to mimic what does or may happen in a real system, makes it an attractive approach with several *benefits* (Pegden et al., 1995, p. 9):

1. New policies, operating procedures, decision rules, organizational structures, and the like, can be explored without disrupting ongoing operations.

2. New hardware designs, physical layouts, software programs, transportation systems, etc., can be tested before committing resources to their acquisition and/or implementation.
3. Hypotheses about how or why certain phenomena occur can be tested.
4. Time can be controlled; it can be compressed, expanded, etc., allowing to speed up or slow down a phenomenon for study.
5. Insight can be gained about which variables are most important for performance and how these interact.
6. Bottlenecks in material, information and product flow can be identified.
7. A simulation study can prove invaluable to understanding how the system actually operates as opposed to how everyone thinks it operates.
8. New situations, about which there is limited knowledge and experience, can be manipulated in order to prepare for theoretical future events. Simulation's great strength lies in its ability to enable the exploration of "what if" questions.

The research presented in Paper I, involving modeling and simulation of a requirements process, mainly aimed at studying the internal interaction within the requirements process (purpose 1) and to simulate the effects of informational and organizational changes to the process (purpose 2). A primary goal was also a better understanding of the system in order to suggest improvements (purpose 3). Finally, by showing how a simulation model of the requirements process may look like, the organization under study was enabled to experiment prior to implementation (purpose 6). The identified advantages for choosing simulation were the possibility to explore the information flow and new process policies (benefit 1), to reveal bottlenecks (benefit 6), to understand how the process behaved (benefit 7) and to answer what would happen if certain changes were made in the process (benefit 8). More information may be found in Paper I.

However, simulation also has a few disadvantages (Pegden et al., 1995, p. 9). Firstly, model building requires special training and experience. Two models that are constructed by two competent individuals may have similarities but is highly unlikely to be the same. Secondly, simulation results may be difficult to interpret, as it may be hard to determine whether the output depends on randomness or system interrelationships. Thirdly, simulation modeling and analysis can be time consuming and expensive. If enough resources

are not assigned, the model or analysis may be insufficient. Fourthly, a disadvantage identified by Banks et al. (1996, p. 5), simulation is sometimes used when an analytical solution is possible, or even preferable. Solvable queuing models may be used in some circumstances. See for example Regnell et al. (2003) for a suggestion of an analytical model for requirements selection quality.

Thanks to vendors of simulation software, there are model packages and thorough analysis available to address the disadvantages. Moreover, simulation may continually be performed even faster, thanks to advances in hardware.

Experimentation

In the most general sense, an experiment is a test of an idea to see what happens Montgomery (2001); Robson (2002). This type of investigation can consequently be performed in virtually any field. Experimentation allows a systematic way of investigating cause-effect relationships and requires careful planning.

The planning phase of an experiment is vital for the success of an experiment. Success does not necessarily mean that an intended effect has been proven, but that proper conclusions may be drawn based on the collected data. The extent of the planning phase is one characteristic of the experimentation method that makes it stand out compared to other research methods. The planning involves (Juristo & Moreno, 2001; Montgomery, 2001; Robson, 2002; Wohlin et al., 2000):

Selecting the context A real setting, outside the laboratory, is the most desirable as it enables easier generalization to the real world and is likely to make measurements more valid. However, real world experimentation in software engineering involves high risks and may cause unacceptable delays in industry. The alternative is to experiment on a smaller off-line project, which is cheaper to run and easier to control.

Define the goal of the experiment Describe what the experiment aims to investigate and its motivation. Explicitly stated goals and motives helps in keeping the focus throughout the experimentation.

Formulate the hypothesis Hypothesis testing is the basis for statistical analysis of experiments. The hypothesis is stated formally using the *null hypothesis*, which states that any differences between two treatments are

coincidental, and the *alternative hypothesis*, which states what is to be established, i.e. it is selected in favor of a rejected null hypothesis.

Select the variables Two types of variables are chosen and defined, the *independent* and the *dependent* variables. The independent variables are those that can be controlled and are changed in the experiment. The dependent variables are those that are measured. In the selection of variables, one must also select the measuring scale, the range, and the levels at which test will be made.

Select the subjects The selection of subjects must be representative for the population under study. Selection of the population is also called *sampling*, since only a sample of the possible population is used in the experiment. Different sampling techniques are available that impact the generalization error and the power of the statistical tests.

Choose an experimental design Conclusions from an experiment are drawn based on statistical tests. In turn, the statistical tests that may be applied depend on the experimental design. There are a number of designs available, ranging from simple design with only one factor to be analyzed using two subject groups, to more complex experiment with several factors and several subject groups. There are even design that use only one subject group, but the recommendation is then to improve the design (using more than on group) or switch to the case study method (see Section 6.1).

Instrumentation To allow proper replication and to provide means for monitoring the experiment without affecting control, the objects, guidelines, and measurement instrument should be clearly defined. This involves decisions on the artifacts used in the experiment (e.g. code, specifications, etc.), the instructions and other material given to the subjects, and the data collection methods and scales (e.g. post-test in the form of a questionnaire).

Evaluate validity The validity of research results is fundamental to all research. To ensure as high validity as possible, i.e. to ensure a successful experiment, validity threats should be considered already in the planning phase. Identification of validity threats at an early stage, enable the researcher to take measures before involving the experiment subjects, which are the most costly resource in the experiment. Validity threats are elaborated in Section 6.2.

A question that is still under debate is whether it is feasible to use student as subjects in software engineering experiments to draw conclusions about professional software developers. There are studies that argue that under certain conditions this is possible (Höst et al., 2000; Runeson, 2003; Kuzniarz et al., 2003). In paper V an experiment with students is presented and the difference between two groups of students is suggested to be transferrable to industry based on indications from a related industrial situation and evaluation. The experiment is conducted based on the suggestion by Juristo & Moreno (2001) that experimentation in software engineering, as in other fields of science and engineering, shall be conducted through a three-stage process:

Laboratory experiments An innovative idea should first be tested ny the innovator in a laboratory setting where market pressures and financial risks are avoided. The experiment should be made replicable by other researchers, by providing the instruments used.

Real projects A limited case study can be conducted with a real team of early adopters. This enables a better study of the limits of the innovation.

Genuine real-world projects When the idea has passed the two previous levels satisfactorily, a full-fledged experiment may be conducted and a large set of data may be collected during a longer period of time. This final level provides the means to draw the final conclusions about the innovation.

According to Juristo & Moreno (2001), the software community does not take the benefits, in terms of reduced risks and increased useful investments, of empirically testing suppositions seriously (at any level). In addition to the arguments against experimentation and their rebuttals, presented by Tichy (1998) (see Table 1.4), Juristo & Moreno (2001) present further difficulties for experimentation in software engineering:

1. Software developers are not trained in the importance and meaning of the scientific method.

2. Software developers are unable to easily understand how to analyze the data of an experiment or how they were analyzed by others because they are lacking the (statistical) training.

3. That there are no experimental design and analysis books for SE makes things harder to understand. Examples from the field are preferred.

Table 1.4: Fallacies and rebuttals about computer science experimentation (Tichy, 1998; Juristo & Moreno, 2001)

Fallacy	Rebuttal
Traditional scientific method isn't applicable	In order to understand the nature of information processes, computer scientists must observe phenomena, formulate explanations and theories, and test them.
The current level of experimentation is good enough	Relative to other sciences, the data shows that computer scientists validate a smaller percentage of their claims.
Experiments cost too much	There are meaningful experiments that fit the budget of small laboratories. There are also expensive experiments that are worth much more than their cost. And there is a wide spectrum in between.
Demonstrations will suffice	Demos can provide proof-of-concepts in the engineering sense, or provide incentives to study a question further. Too often, however, these demos merely illustrate a potential.
There is too much noise in the way	An effective simplification for repeated experiments is benchmarking. Fortunately, benchmarking can be used for many questions in computer science.
Progress will slow	Increasing the ratio of papers with meaningful validation has a good chance of actually accelerating progress.
Technology changes too fast	If a question becomes irrelevant quickly, it is perhaps too narrow and not worth spending a lot of effort on it.
You'll never get it published	Smaller steps are still worth publishing because they improve our understanding and raise new questions is a thinking that some are not familiar with.

4. Empirical studies conducted to check the ideas of others (i.e. replications) are not very publishable.

5. There is an immense number of variables that influence software development.

6. It is difficult to get global results in software engineering, such as, for example, determining the circumstances under which one technique should be selected instead of another or, alternatively, proving that alternative A is always better.

7. Software engineering is a discipline that is dependent of practitioners. So, the result od several people applying one and the same software ar-

tifact (technique, process, tool, etc.) will almost certainly yield different results.

8. Companies are continuously developing new, increasingly complex, and, ultimately more expensive software systems. This should be a condition for applying the different approaches in a reliable manner. Paradoxically, however, the market is often used as a culture medium for performing these experiments, with the usual risks.

Despite the difficulties there are no strong arguments against experimentation. Experimentation provide a means to make claims supported by real data rather than common beliefs and assumptions.

6.2 Threats to validity

Although well-known strategies, methodologies, and methods have been used to conduct the research and arrive at the conclusions presented in this thesis, the results should be interpreted with respect to the potential threats to validity. Different classification schemes have been presented to systematically address different types of validity threats.

One classification scheme, suggested by Cook & Campbell (1979) as an extension to the work by Campbell & Stanley (1963), groups the validity of research results into four different types, each addressing a specific methodological question (Trochim, 2000). The types are explained below in the context of a causal study, where a potential relationship between a cause and effect is sought.

Conclusion validity Is there a relationship between the cause and the effect? It may be concluded that there is a relationship, that there is a positive relationship, that there is no relationship, etc. In each of these cases, the conclusion validity may be assessed.

Internal validity Assuming that there is a relationship, is the relationship a causal one? A correlation between the cause and the effect in a study does not necessarily mean that the construct is causing the effect. Stated differently, other factors than the independent variables may cause the effects.

Construct validity Assuming that there is a causal relationship, can it be claimed that the treatment reflects the construct of the treatment and that the measure well reflects the idea of the construct of the measure?

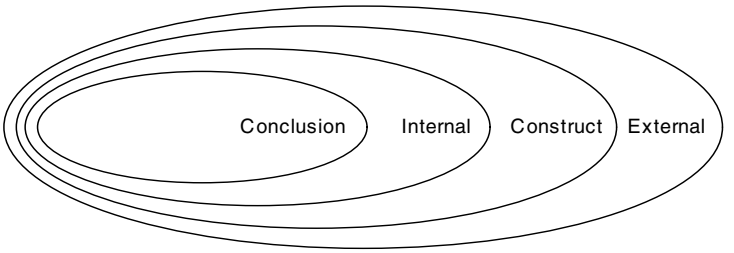


Figure 1.3: Illustration of the relationship between the different types of validity

I.e., was the intended treatment really implemented and was the intended measure really what was measured?

External validity (referred to as *generalizability* by Robson (2002)) Assuming that there is a causal relationship in the study between the constructs of the cause and the effect, can the effect be generalized to other places, times, or people? Claims may be made that the research findings have implications for other similar settings.

As the methodological questions above point out, the validity types build upon each other and each type assumes that the previous validity type is ensured. This relationship is illustrated in Figure 1.3.

Robson (2002) also suggest to consider *reliability*, *objectivity*, and *credibility* as separate threats to validity. Reliability and credibility are closely related to conclusion validity and have to do with the reliability of the measure. Primary causes to reliability threats are *participant error* and *observer error*. Participant error may occur if the participants seek to please the researcher or when fluctuations in the participants performance are not anticipated. Observer error may occur if the researcher has been tired or lacking in concentration when measuring or taking notes, or unconsciously biases the results in line with his or her own beliefs.

Objectivity has to do with the risk that the involved researcher may distort the response from using a certain methodology due to prejudices, values, and specific interests. Robson (2002) gives the example of the possibility of viewing an experiment either as the answer or as an extreme version of a problem.

Indeed, attempts should be made to reduce the threats to validity. There may, for example, be insufficient statistical power to detect a relationship, a sample size may be to small, a measure may be unreliable, variability in the data may be caused by random heterogeneity, etc. By showing that possible

alternative explanations are not credible, the most plausible conclusion may correctly and reliably be reached (Trochim, 2000).

In section 8 the specific threats to the validity of the research presented in this thesis are discussed, with respect to the four-group scheme presented above. Furthermore, Paper I through Paper IV has gone through extensive peer-review processes (three or more researchers have reviewed each paper prior to publication) in which the reliability and credibility of the research has been assessed according to research community standards.

7 Research questions

Four research questions were formulated in 2002 based on the goal presented in Section 6. For clarity, the goal is here restated:

> *To find automated support to the manual work in the management of natural language requirements in large-scale software development.*

The original research questions are stated below. The next section discusses how the research in this thesis has managed to answer these questions.

RQ1 *What are the possible applications of automated relationship analysis techniques to support the requirements analyst?*

RQ2 *Which techniques may improve the accuracy of automated relationship analysis of requirements?*

RQ3 *How may the result from automated relationship analysis techniques best be supported by and visualized in CASE tools?*

RQ4 *To what extent is natural language used to specify requirements in current market-driven software development companies?*

8 Contribution

This thesis addresses the issues of saving time and effort in the management of large amounts of requirements in market-driven software development. A summary of each individual paper can be found in the paper abstracts, which are located at the beginning of each included paper, respectively. This section summarizes the main contribution and relates it to to the research questions, RQ1–RQ4, in Section 7 and to Paper I–Paper V.

C1 *Demonstrated how simulation may be used to predict where congestion occur in the requirements engineering process*

Paper I presents a case where discrete-event simulation has been used to investigate the conditions that result in an overloaded requirements engineering process, with respect to the continuous and complex management of requirements. It is shown that in the investigated process, overload may be avoided either by increasing the number of people in the implementation phase from 30 to 165 or by reducing the number of elicited requirements to less than a fifth. Since only a fraction of all the elicited requirements are supposed to be implemented, the focus should be on reducing the number of requirements in the process. Duplicate identification was identified as one rewarding activity for rejecting many requirements. The results from the paper is therefore also the origin for the subsequent research and results.

C2 *Demonstrated the applicability of robust linguistic engineering techniques for supporting different large-scale requirements management activities, in which the common denominator is the matching of semantically similar requirements*

Paper II, III, and V present three different cases, involving three different requirements engineering processes and three different applications to requirements management. The results and experience reported from these studies show that three different requirements management tasks - duplicate identification, requirements linkage, and requirements consolidation - may be alleviated through the use of linguistic engineering techniques. In addition the these tasks, other tasks have been identified as possible application areas, which are presented in Paper II. As experience was gained from the first study in Paper II and other research was made available, a new morphological analyzer and a more appropriate weighting scheme was selected when moving to the cases in Paper III and Paper IV. In addition, several ideas for future improvement have been identified during the evaluations. These are presented in Section 9.

C3 *Developed a support tool, which utilizes the proposed linguistic engineering techniques, and demonstrated that its usage may increase performance in the task of linking related requirements.*

A tool, presented in Paper IV, has been developed based on the experience gained from the research presented in Paper II and Paper III. The tool has not undergone extensive usability tests so the author does not claim that the tool's design is the best chosen. However, two evaluations presented in Paper V, one with students and one with experts in an industrial setting, reports on a positive attitude towards the tool and indications that it effectively support requirements consolidation.

Table 1.5 presents the relationship between the included papers, and the research questions, contribution, strategies, and methods used. Research question 4 is not explicitly answered by the research included in this thesis. However, the presented studies give evidence that natural language is extensively used in three large software development companies. Moreover, indications that natural language is used in industry is supported by the results from an interview study to which the author of this thesis has contributed with both interview design, realization, and analysis (Karlsson et al., 2002). The study involved 5 market-driven companies of which all used natural language to document requirements. The author's experience from other companies also indicate the natural language is prevalent. That natural language is still widely used is also supported by another study by Mich et al. (2004), which showed that, out of 103 companies, 79% of the companies use common natural language and another 16% use structured natural language (templates, form, etc.) for specifying requirements.

8.1 Validity threats

Following is an account of the validity threats to the research presented in this thesis. The context and further details are found in each paper. Validity threats in general are discussed in Section 6.2.

In the simulation study in Paper I, the major threat to validity concerns the construct, i.e. the model and the degree to which it faithfully represents its system counterpart (Zeigler et al., 2000). Validation was performed through an iterative process of running the model and analyze the output behavior. This process was terminated when the model was considered to capture the

Table 1.5: A mapping between the papers included in this thesis and the research questions, contribution, strategies, and methods used.

Paper	Question	Contribution	Strategy & method	Data collection
I	RQ1	C1	Fixed & flexible, Case study	Simulation, Interview
II	RQ2, (RQ4)	C2	Fixed & flexible, Case study	Content analysis, Questionnaire
III	RQ2, (RQ4)	C2	Flexible, Case study	Content analysis
IV	RQ2, RQ3, (RQ4)	C2, C3	Flexible, Case study	Content analysis
V	RQ2, RQ3, (RQ4)	C3	Fixed, Experimental	Content analysis, Questionnaire

system behavior to the extent demanded by our objectives. Furthermore, representatives from the company under study validated the simulation output to be accurate, thus further assuring construct validity. There may be threats to internal validity of the conclusions on how to avoid bottlenecks, but although the results may be somewhat surprising, they are considered plausible. Improvements to the model were then identified and thus also other potential threats to validity.

In the studies in Papers II through Paper V, in which different computer programs (in Perl, C, C++, and Java) have been implemented for tokenization, morphological analysis, stop word removal and similarity calculations (see Section 5 for a description of these concepts), there may be threats to internal and construct validity. Each step must be correctly performed in order to reach a mathematically correct similarity measure. To address this threat and to minimize faults in the implementation, the programs have been tested using randomly selected requirements. The results from the programs have been compared to results from manually performing the preprocessing and calculation steps (see Section 5). Several faults have been removed thanks to this process, but, naturally, several faults could remain. For example, the implementation of the tokenization and morphological analysis might remove words that should not have been removed (undetected in the evaluation using randomly selected requirements), which could affect the results.

There are two reasons to why these threats are not too problematic. Firstly, the statistical nature of the approach makes it resilient to minor flaws in the

preprocessing steps, comparable to naturally fluctuating quality of the raw input text, i.e the requirements in its original form. Naturally, the specific similarity measures should be calculated correctly according to the formulas, but this has been assured as described above. Validity in terms of implementation of the preprocessing steps is further assured by the several different implementations that have been used, with respect to software design, programming language, tokenization scheme and morphological analysis. Since all results from the different implementations point in the very same direction, it is considered plausible that any potential errors in the implementation will have statistically negligible impact on the results.

Secondly, the approach does not, thanks to the specific application environment, require any measures of performance, accuracy or precision to be near-perfect. It is never claimed that the approach shall be used unsupervised or without human intervention. The approach is suggested to be used as a supportive technique leaving the final judgements to the experts in the field. Specifically, the experiment in Paper V indicates that, with or without errors in the implementation, the approach may give valuable support.

Threats to conclusion validity are potentially present in the studies in Paper II through Paper V. In these studies, the outcome of the automatically calculated similarity measures and the links assigned by experiment subject have been compared to available facts provided by experts. I.e., the duplicate requirements already found by exerts and the links between requirements already assigned by experts have been used as the presumably correct results. These presumably correct results could be incomplete and erroneous. In the study of duplicate requirements identification the outcome of one run was therefore evaluated separately by an expert from the company, who found that there were more relationships between the requirements than initially specified. This indicates that the keys used are more likely to be incomplete than erroneous. This is also plausible due to the lack of time in industry for the analysis of requirements. Nevertheless, human errors are likely to be introduced with or without the automated support. The experiment in Paper V actually indicates that fewer errors are made using automated support.

Another threat to conclusion validity is related to the intricacy of the requirements relationships. As expected, the presented approach does not perform with 100% accuracy or precision. The conclusion that the suggested techniques is a generally reasonable approximation for indicating semantic similarity, based on the available relationships established by experts, could be incorrect. Certainly, there are cases where the approach fails, as specifically

reported in Paper III (204 unidentified requirement links). The reasons for this is a matter for further research into the semantics of requirements. However, the approach is never suggested to replace human judgement, but merely supporting it. The claim is that the approach may save time in requirements management activities and that any saved time could be put on finding more intricate relationships.

A related threat to conclusion validity is that the keys used may be representations of only simple relationships between requirements, which are easily found by experts using manual methods. This may suggest that not much time could be saved after all. This is addressed in Paper III where a reasonable calculation shows the time that could be saved based on the available links established by experts and the presented findings. This is furthermore supported by the experiment and experts' judgement presented in Paper V. Of course, the particular experiment is not made on a very large requirement set, and the obvious question is how it scales up. However, the combined results from the different studies imply that the method does scale up.

The experiment in Paper V is subject for various threats to validity, as experiments generally are. Conclusion validity threats related to statistical tests have been addressed by checking the distribution of the results and by using the correct statistical test based on that. Conclusion validity threats related to subjects are limited as the subject group is rather homogenous. Internal validity threats, due to history, maturity, mortality, etc., are limited due to the short experimentation time. Social threats were also considered limited as the student had nothing to gain in the particular outcome. To reduce construct validity threats, more than one researcher was involved in the experimentation. Another set of requirements would be needed to see whether the particular requirements set actually had an impact on the results. Measures that the subjects were aware of may also affect the results in an uncontrolled manner. The largest threat to external validity is that students have been used as subjects. However, the students are in their third year of software engineering studies and close to start working in industry. Moreover, the participants are familiar with the application domain, which is industry-like, as they participate in a course where they produce a requirements specification for essentially the same system as is the target for the requirements specifications used in the experiment.

The case study strategy, which is used throughout the presented research, entails specific external validity threats. The generalizability of the results may be the most questionable and further studies could be made to support the

approach. However, three different sets of industrial requirements and two sets from educational software projects have been used. The same underlying technique have been used in different settings and the results point in the same direction, giving reasonable evidence that the results are generalizable to other domains and settings in the software industry.

9 Further work

Research in applying linguistic engineering techniques to requirements engineering and management has been conducted over a period of approximately 10 years. According to a study of software technologies it takes in between 15 and 20 years for a technology to evolve from concept formulation to the propagation throughout the community of users (Redwine Jr & Riddle, 1985). Considering the young age of requirements engineering and the disparate attempts made so far, full technology transfer of applying linguistic engineering techniques to requirements engineering management is likely some years away.

Linguistic engineering techniques are widely used in information intensive support systems; however most CASE tools excluded. Tools and techniques are available and may be successfully adapted and further exploited. With the increase in the amount of information written in natural language that large software development companies need to manage, these techniques are worthwhile taking more advantage of. In a state-of-the-practice talk at the International Requirements Engineering Conference in 2004, Dr. Jeremy Dick from requirements management tool vendor Telelogic, also noted that natural language processing techniques is one field to consider with respect to requirements traceability (Dick, 2004).

The approach of calculating similarity between requirements on a lexical level, performs reasonably well considering its simplicity. Most importantly, it provides added support to the management of large repositories of natural language requirements. The support is not aimed at replacing the current way of working, but to complement it in order to save time.

The simplicity of the technique is a deliberate choice. As such, it is robust and requires no or little maintenance or attention, which is important for acceptance in industry. For research purposes, the presented evaluations acts as a baseline to which further research may be compared.

Based on the research presented in this thesis, the following aspects have been identified to deserve further investigation and evaluation:

- Term weighting schemes specifically relevant for the style of requirements. Perhaps the term occurrence itself is more important than how often it occurs. For example, assume that requirement A comprises 10 words, requirement B comprises 10 words, and that requirement C comprises 120 words. Assume that requirement A and B are semantically more similar than any other pair of requirements. Now, suppose that requirementys A, B, and C share the same set of terms. The similarity measure between A and C would then be higher than between A and B, which is not what is expected. Thus, it may be appropriate to also consider the structure of the requirements.

- Similarity measuring techniques that give increased recall and precision. Other measures may be investigated and different measures and techniques could be combined to increase overall accuracy (e.g. part-of-speech tagging, sliding window, and Latent Semantic Analysis (Landauer & Dumais, 1997)).

- Incorporation of semantics to catch more distant similarities. E.g., treating compound concepts as tokens, using a lexicon to deal with synonyms, hypernyms, and hyponyms (e.g. WordNet (Fellbaum, 1998), domain-specific lexicons, etc.).

- Expansion of abbreviations and other mnemonics to their full textual representation. For example, the company presented in Paper III uses two names for each software component, one identifier and one natural language functional name. The identifiers may appear in the requirements and a replacement of the identifiers with their functional name could improve accuracy.

- Feedback to support alternative decisions on similarity. E.g., data mining of the requirements repository where already linked requirements may provide information on which tokens that may indicate higher affinity between requirements. Another idea is to use Bayesian learning (Neal, 1996) in the process where a human assess the outcome from suggested similarities (i.e., a form of relevance feedback).

- Impact of the selection of attributes to include in the similarity calculation. Experienced requirements manager realize that it is difficult to find the perfect structure, but there may be requirements attributes that provide better results. The impact depends on the correctness of the contents in these attributes as it is not self-evident that all attributes are

consistently used (e.g., is a particular requirement a usability requirement or a performance requirement?).

- Ways of visualizing the output from the similarity calculations to support the requirements manager in the navigation among related requirements (e.g., TileBars (Hearst, 1995)).

In order to assure external validity, i.e., generalizability, suggested improvement should be evaluated on real industrial requirements. Preferably, evaluations are also performed through experimentation, both in a laboratory environment and in real industrial projects. The latter will better reveal the cost-benefit of any suggested improvement. Any domain-specific improvements may add a cost to the practitioner. These costs should be motivated and related to the positive effects on the requirements management process.

References

Banks, J., Carson, J. S., & Nelson, B. (1996). *Discrete-event system simulation* (2nd ed.). Upper Saddle River, NJ: Prentice Hall.

Beck, K. (1999). Embracing change with extreme programming. *Computer*, *32*(10), 70–77.

Boehm, B. W. (1976). Software engineering. *IEEE Transactions on Computers*, *25*(12), 1226–1241.

Boehm, B. W. (1984). Verifying and validating software requirements and design specifications. *IEEE Software*, *1*(1), 75–88.

Bratley, P., Fox, B. L., & Schrage, L. E. (1987). *A guide to simulation* (2nd ed.). New York, NY: Springer-Verlag.

Brooks, F. P., Jr. (1975/1995). *The mythical man-month: Essays on software engineering* (Anniversary ed.). Boston, MA: Addison-Wesley.

Burg, J. F. M. (1996). *Linguistic instruments in requirements engineering*. Doctoral dissertation, Vrije Universiteit, The Netherlands. IOS Press: Amsterdam, The Netherlands.

Campbell, D. T., & Stanley, J. C. (1963). *Experimental and quasi-experimental designs for research*. Boston, MA: Houghton Mifflin Company.

Carlshamre, P. (2002). *A usability perspective on requirements engineering - from methodology to product development*. Doctoral dissertation, Linköping Studies in Science and Technology, Linköping University, Sweden. Dissertation No. 726.

Carlshamre, P., & Regnell, B. (2000). Requirements lifecycle management and release planning in market-driven requirements engineering processes. In A. M. Tjoa, R. R. Wagner, & A. Al-Zobaidie (Eds.), *Proceedings of the 11th international workshop on database and expert systems applications process* (pp. 961–965). Los Alamitos, CA: IEEE CS.

Chomsky, N. (2002). *Syntactic structures* (2nd ed.). Berlin: Walter de Gruyter.

Cook, T. D., & Campbell, D. T. (1979). *Quasi-experimentation: Design and analysis issues for field settings*. Boston, MA: Houghton Mifflin Company.

Cybulski, J. L., & Reed, K. (1998, Dec). Computer-assisted analysis and refinement of informal software requirements documents. In *Proceedings of the fifth asia-pacific software engineering conference.* Taipei, Taiwan.

Cybulski, J. L., & Reed, K. (1999, Sep). Automating requirements refinement with cross-domain requirements classification. In *Proceeding of the fourth australian conference on requirements engineering (ACRE'99)* (pp. 131–145). Macquarie University, Sydney.

Cyre, W. R., & Thakar, A. (1997). Generating validation feedback for automatic interpretation of informal requirements. *Formal Methods in System Design*, *10*(1), 73–92.

Daly, E. B. (1977). Management of software development. *IEEE Transactions on Software Engineering*, *3*(3), 229–242.

Davis, A. M. (1993). *Software requirements - objects, functions, & states* (Revised ed.). Upper Saddle River, NJ: Prentice Hall.

Davis, A. M., Jordan, K., & Nakajima, T. (1997). Elements underlying the specification of requirements. *Annals of Software Engineering*, *3*, 63–100.

Dick, J. (2004). *State-of-the-practice talk: Requirements traceability: Whither, why and wherefore. International Requirements Engineering Conference (RE2004)*, Kyoto, Japan.

Fabbrini, F., Fusani, M., Gervasi, V., Gnesi, S., & Ruggieri, S. (1998, May). Achieving quality in natural language requirements. In *Proceedings of the 11th international software quality week (QW'98).* San Fransisco, CA: Software Research Institute.

Fabbrini, F., Fusani, M., Gnesi, S., & Lami, G. (2001, Nov). The linguistic approach to the natural language requirements quality: Benefit of the use of an automatic tool. In *Proceedings of the 26th annual nasa goddard software engineering workshop* (pp. 97–105). Greenbelt, Maryland: IEEE CS.

Fantechi, A., Gnesi, S., Lami, G., & Maccari, A. (2000). Application of linguistic techniques for use case analysis. *Requirements Engineering*, *8*(3), 161–170.

Fellbaum, C. (Ed.). (1998). *Wordnet: An electronic lexical database*. Cambridge, MA: MIT Press.

Fillmore, C. J. (1968). The case for case. In E. W. Bach & R. T. Harms (Eds.), *Universals in linguistics theory* (pp. 1–90). New York, NY: Holt, Rinehart, and Winston, Inc.

Fliedl, G., Kop, C., & Mayr, H. C. (2003, June). From scenarios to KCPM dynamic schemas: Aspects of automatic mapping. In *Proceedings of the 8th international conference on applications of natural language to information systems (NLDB 2003)* (pp. 91–105). Burg (Spreewald), Germany.

Flores, J. J. G. (2004, June). Linguistic processing of natural language requirements: The contextual exploration approach. In B. Regnell, E. Kamsties, & V. Gervasi (Eds.), *Proceedings of the 10th anniversary international workshop on requirements engineering: Foundation for software quality.* Riga, Latvia.

Francis, W. N., & Kucera, H. (1982). *Frequency analysis of english usage: lexicon and grammar*. Boston, MA: Houghton Mifflin.

Fuchs, N. E., & Schwertel, U. (2003). Reasoning in attempto controlled english. In *Proceedings of the international workshop on principles and practice of semantic web reasoning, ppswr* (pp. 174–188). Mumbai, India: Springer Verlag.

Garigliano, R. (1995, Mar). JNLE Editorial. *Natural Language Engineering*, *1*(1), 1–7.

Gervasi, V. (2000). *Environment support for requirements writing and analysis*. Doctoral dissertation, Dipartimento di Informatica, University of Pisa, Italy. Dissertation No. 82.

Gervasi, V., & Nuseibeh, B. (2002). Lightweight validation of natural language requirements: a case study. *Software Practice and Experience*, *32*, 113–133.

Gervasi, V., & Zowghi, D. (2005). Reasoning about inconsistencies in natural language requirements. *ACM Transactions on Software Engineering and Methodology*.

Goguen, J. A. (1996). Formality and informality in requirements engineering. In S. Fickas & A. Finkelstein (Eds.), *Proceedings of the fourth international conference on requirements engineering* (pp. 102–108). Los Alamitos, CA: IEEE CS.

Goldin, L., & Berry, D. M. (1997). AbstFinder, a prototype natural language text abstraction finder for use in requirements elicitation. *Automated Software Engineering*, *4*(4), 375–412.

Hearst, M. (1995). Tilebars: Visualization of term distribution information in full text information access. In I. R. Katz, R. Mack, L. Marks, M. B. Rosson, & J. Nielsen (Eds.), *Proceedings of the acm sigchi conference on human factors in computing systems* (pp. 59–66). New York, NY: ACM.

Hofmann, H. F., & Lehner, F. (2001). Requirements engineering as a success factor in software projects. *IEEE Software*, *18*(4), 58–66.

Höst, M., Regnell, B., & Wohlin, C. (2000, Nov). Using students as subjects – a comparative study of students and professionals in lead-time impact assessment. *Empirical Software Engineering*, *5*(3), 201–214.

Jackson, M. (1995). *Requirements and specifications: A lexicon of software practice, principles and prejudices*. Boston, MA: Addison-Wesley.

Jackson, P., & Moulinier, I. (2002). *Natural language processing for online applications: Text retreival, extraction and categorization*. Amsterdam, The Netherlands: John Benjamins.

Jones, C. (1996). *Patterns of software systems failure and success*. Boston, MA: International Thomson Computer Press.

Jurafsky, D., & Marting, J. H. (2000). *Speech and language processing*. Upper Saddle River, NJ: Prentice Hall.

Juristo, N., & Moreno, A. M. (2001). *Basics of software engineering experimentation*. Boston, MA: Kluwer Academic Publishers.

Kamsties, E., Hörmann, K., & Schlich, M. (1998). Requirements engineering in small and medium enterprises. *Requirements Engineering*, *3*(2), 84–90.

Karlsson, L., Dahlstedt, A. G., Natt och Dag, J., Regnell, B., & Persson, A. (2002). Challenges in market-driven requirements engineering - an industrial interview study. In C. Saliensi, B. Regnell, & K. Pohl (Eds.), *Proceedings of the eigth international workshop on requirements engineering: Foundation for software quality* (pp. 37–49). Essen, Germany: Essener Informatik Beiträge.

Keil, M., & Carmel, E. (1995, May). Customer-developer links in software development. *Communications of the ACM*, *38*(5), 33–44.

Kotonya, G., & Sommerville, I. (1997). *Requirements engineering: processes and techniques*. New York: John Wiley & Sons.

Kristensson, P., Magnusson, P. R., & Matthing, J. (2002). Users as a hidden resource for creativity - findings from an experimental study on user involvement. *Creativity and Innovation Management*, *11*(1), 55–61.

Kuzniarz, L., Staron, M., & Wohlin, C. (2003). Students as study subjects in software engineering experimentation. In *Proceedings of the 3rd conference on software engineering research and practic in sweden* (pp. 19–24). Lund, Sweden.

Landauer, T. K., & Dumais, S. T. (1997). A solution to plato's problem: The latent semantic analysis theory of acquisition, induction and representation of knowledge. *Psychological Review*, *104*(2), 211–240.

Lauesen, S. (2002). *Software requirements: Styles and techniques*. London, UK: Addison-Wasley.

Lubars, M., Potts, C., & Richter, C. (1993). A review of the state of the practice in requirements modeling. In *Proceedings of IEEE international symposium on requirements engineering* (pp. 2–14). Los Alamitos, CA: IEEE CS.

Macias, B., & Pulman, S. G. (1993). Natural language processing for requirements specifications. In *Safety critical systems* (pp. 57–89). Chapman and Hall.

Macias, B., & Pulman, S. G. (1995). A method for controlling the production of specifications in natural language. *The Computer Journal*, *38*(4), 310–318.

Manning, C. D., & Schütze, H. (2002). *Foundations of statistical natural language processing*. Cambridge, MA: MIT Press.

Martin, R. C. (2002). *Agile software development, principles, patterns, and practices*. Upper Saddle River, NJ: Prentice Hall.

Mich, L., Franch, M., & Novi Inverardi, P. L. (2004). Market research for requirements analysis using linguistic tools. *Requirements Engineering*, *9*(1), 40–56.

Mich, L., & Garigliano, R. (2002). Nl-oops: A requirements analysis tool based on natural language processing. In C. A. Brebbia, N. F. Ebecken, P. Melli, & A. Zanasi (Eds.), *Proceedings of the* 3^{rd} *international conference on data mining* (pp. 321–330). Wessex: WIT Press.

Minnen, G., Carroll, J., & Pearce, D. (2001, Sep). Applied morphological processing of english. *Natural Language Engineering*, *7*(3), 207–223.

Montgomery, D. C. (2001). *Design and analysis of experiments*. USA: John Wiley & Sons.

Nanduri, S., & Rugaber, S. (1996). Requirements validation via automated natural langauge parsing. *Journal of Management Information Systems*, *12*(3), 9–19.

Naylor, T. I. H., Balintfy, J. L., Burdick, D. S., & Chu, K. (1966). *Computer simulation techniques*. New York, NY: John Wiley.

Neal, R. M. (1996). Bayesian learning for neural networks. In *Lecture notes in statistics* (Vol. 118). New York, NY: Springer-Verlag.

Novorita, R. J., & Grube, G. (1996, July). Benefits of structured requirements methods for market-based enterprises. In *Proceedings of the sixth annual international symposium on systems engineering (INCOSE'96).* Boston, MA.

Osborne, M., & MacNish, C. K. (1996). Processing natural language software requirements specifications. In *Proceedings of the 2nd international conference on requirements engineering (ICRE'96)* (pp. 229–236). Colorado Springs, CO.

Park, S., Kim, H., Ko, Y., & Seo, J. (2000). Implementation of an efficient requirements-analysis supporting system using similaity measure techniques. *Information and Software Technology*, *42*(6), 429–438.

Parker, S. D., Eason, K. D., & Dobson, J. E. (1993). The change and evolution of requirements as a challenge to the practice of software engineering. In *Proceedings of IEEE international symposium on requirements engineering* (pp. 266–272). Los Alamitos, CA: IEEE CS.

Pegden, C. D., Sadowski, R. P., & Shannon, R. E. (1995). *Introduction to simulation using siman* (2nd ed.). New York, NY: McGraw-Hill.

Porter, M. F. (1980). An algorithm for suffix stripping. *Program*, *14*(3), 130–137. ((reprinted in Readings in Information Retrieval, Morgan Kaufmann, 1997))

Potts, C. (1995). Invented requirements and imagined customers: Requirements engineering for off-the-shelf software. In *Proceedings of the second IEEE international symposium on requirements engineering* (pp. 128–130). Los Alamitos, CA: IEEE CS.

Rayson, P., Emmet, L., Garsida, R., & Sawyer, P. (2000, Jun). The REVERE project: Experiments with the application of probabilistic NLP to systems engineering. In *Proceedings of the fifth international conference on applications of natural language to information systems (NLDB 2000).* Versailles, France.

Redwine Jr, S. T., & Riddle, W. E. (1985). Software technology maturation. In *Proceedings of the eighth international conference on software engineering* (pp. 189–200). Los Alamitos, CA: IEEE CS.

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

Regnell, B., Karlsson, L., & Höst, M. (2003). An analytical model for requirements selection quality evaluation in product software development. In D. C Martin (Ed.), *Proceeding of the 11th IEEE international requirements engineering conference* (pp. 254–263). Los Alamitos, CA: IEEE CS.

Robertson, S., & Robertson, J. (1999). *Mastering the requirements process*. Harlow, UK: Addison-Wesley.

Robson, C. (2002). *Real world research* (2nd ed.). Oxford, UK: Blackwell.

Rolland, C., & Proix, C. (1992, May). A natural language approach for requirements engineering. In *Proceedings of the fourth international conference on advanced information systems engineering (CAISE'92)* (pp. 257–277). Manchester, UK.

Royce, W. W. (1970). Managing the development of large software systems: concepts and techniques. In *Proceedings of IEEE WESTCON* (pp. 1–9). Los Alamitos, CA: IEEE CS.

Rumbaugh, J. E., Blaha, M. R., Premerlani, W. J., Eddy, F., & Lorensen, W. E. (1991). *Object-oriented modeling and design*. Upper Saddle River, NJ: Prentice-Hall.

Runeson, P. (2003). Using students as experiment subjects - an analysis on graduate and freshmen student data. In *Proceedings of the 7th international conference on empirical assessment & evaluation in software engineering.*

Ryan, K. (1997). Commentary on abstfinder: A prototype natural language text abtraction finder for use in requirements elicitation. *Automated Software Engineering*, *4*(1), 415.

Sawyer, P. (2000). Packaged software: Challenges for RE. In *Proceedings of sixth international workshop on requirements engineering: Foundation for software quality* (pp. 137–142). Essen, Germany: Essener Informatik Beiträge.

Sawyer, P., & Cosh, K. (2004, June). Supporting MEASUR-driven analysis using NLP tools. In B. Regnell, E. Kamsties, & V. Gervasi (Eds.), *Proceedings of the 10th anniversary international workshop on requirements engineering: Foundation for software quality.* Riga, Latvia.

Sawyer, P., Sommerville, I., & Kotonya, G. (1999, Jun). Improving market-driven RE processes. In *Proceedings of international conference on product focused software process improvement (PROFES'99)* (pp. 222–236). Oulu, Finland.

Siddiqi, J., & Shekaran, M. C. (1996). Requirements engineering: The emerging wisdom. *IEEE Software*, *13*(2), 15–19.

Somé, S. S., Dssouli, R., & Vaucher, J. G. (1996). Toward an automation of requirements engineering using scenarios. *Journal of Computing and Information*, *2*(1), 1110–1132.

Sommerville, I. (2001). *Software engineering* (6th ed.). Harlow, UK: Pearson Education.

Sommerville, I., & Sawyer, P. (1997). *Requirements engineering - a good practice guide*. Chichester, UK: John Wiley & Sons.

Sutton, D. C. (2000). Linguistic problems with requirements and knowledge elicitation. *Requirements Engineering*, *5*, 114–124.

Tichy, W. F. (1998). Should computer scientists experiment more? 16 reasons to avoid experimentation. *IEEE Computer*, *31*(5), 32–40.

Trochim, W. M. (2000). *The research methods knowledge base* (2nd ed.). Cincinnati: Atomic Dog Publishing.

Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, *59*, 433–460.

Van Rijsbergen, C. J. (1979). *Information retrieval* (2nd ed.). Dept. of Computer Science, University of Glasgow.

Wieringa, R., & Ebert, C. (2004). Guest editors' introduction: RE'03: Practical requirements engineering solutions. *IEEE Software*, *21*(2), 16–18.

Wilson, W. M., Rose, L. H., & Hyatt, L. E. (1996). Automated quality analysis of natural language requirement specifications. In *Proceedings of the 14th annual pacific northwest software quality conference* (pp. 140–151).

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2000). *Experimentation in software engineering - an introduction*. Norwell, MA: Kluwer.

Yeh, A. (1992). Requirements engineering support technique (REQUEST) - a market driven requirements management process. In *Proceedings of the second symposium on assessment of quality software development tools* (pp. 211–223). Los Alamitos, CA: IEEE CS.

Yin, R. K. (1993). *Applications of case study research* (Vol. 34). Thousand Oaks, CA: SAGE publications.

Yin, R. K. (1994). *Case study research: Design and methods* (2nd ed., Vol. 5). Thousand Oaks, CA: Sage Publications.

Zeigler, B. P., Praehofer, H., & Kim, T. G. (2000). *Theory of modeling and simulation - integrating discrete event and continuous complex dynamic systems* (2nd ed.). San Diego, CA: Academic Press.

PAPER I

Exploring Bottlenecks in Market-Driven Requirements Management Processes with Discrete Event Simulation

Martin Höst, Björn Regnell, Johan Natt och Dag, Josef Nedstam, Christian Nyberg

Journal of Systems and Software, 59, 323–332, 2001

I

Abstract

This paper presents a study where a market-driven requirements management process is simulated. In market-driven software development, generic software packages are released to a market with many customers. New requirements are continuously issued, and the objective of the requirements management process is to elicit, manage, and prioritize the requirements. In the presented study, a specific requirements management process is modeled using discrete event simulation, and the parameters of the model are estimated based on interviews with people from the specific organization where the process is used. Based on the results from simulations, conditions that result in an overload situation are identified. Simulations are also used to find process change proposals that can result in a non-overloaded process. The risk of overload can be avoided if the capacity of the requirements management process is increased, or if the number of incoming requirements is decreased, for example, through early rejection of low-priority requirements.

1 Introduction

Requirements Engineering (RE) in a market-driven context, where succeeding versions of a software package are released to a market, is getting increased attention (Lubars et al., 1993; Potts, 1995; Yeh, 1992). When developing software for a market, rather than for a single customer, the pressure on short time-to-market is evident. An effective engineering of software requirements is an important success factor for meeting market demands. RE involves activities such as analyzing and prioritizing requirements, and maintaining a database of requirements that may be implemented in the future. This part of the software process requires resources, and the allocation of resources to activities related to requirements selection and release planning is crucial to the continuous delivery of competitive software releases. Traditional RE has mainly been focused on the bespoke situation where a specific system is developed based on a contract with a specific customer. The market-driven situation, however, has special challenges regarding scheduling constraints and stakeholding (Sawyer, 2000), and there is an industrial need for process improvement in this area (Sawyer et al., 1999).

One way of analyzing process improvement proposals is to carry out pilot studies or controlled experiments within the specific organization (Wohlin et al., 2000). However, this requires much resources and an alternative approach is to carry out simulations of the organization instead (Kellner et al., 1999; Pfahl and Lebsanft, 2000). This is an engineering approach that is chosen in many other areas, and it can, of course, be applied in evaluation of software development processes too. After the new processes have been analyzed through simulation they can be analyzed in experiments and case studies. In this way simulations can be a natural part of technology transfer and evaluation. Simulations may reduce the risk of implementing process changes that are not resulting in improvements. Since many people in the organization often are involved in experiments and pilot-studies, it may be a large problem if the wrong changes are introduced and evaluated. This can very well damage the continued process improvement work in the organization for a long time. Thus, there is a clear opportunity for simulation as a first step in the evaluation of new software process technology.

The objective of the presented study is to investigate if simulation can help in exploring bottlenecks and overload situations in RE processes. The object of simulation is a specific process called REPEAT (Regnell et al., 1998), which is used by a leading CASE-tool developer for real-time systems development (Telelogic AB). The REPEAT process is a result of an improvement

programme that started in 1995, as Telelogic considered efficient RE a key success factor. After the introduction of REPEAT, a significant improvement in delivery precision and product quality was gained. However, after a number of releases with REPEAT, it was realized that market pressure resulted in a number of further challenges regarding through-put and congestion (Regnell et al., 1998). This led to a research project with the objective of further understanding and improving market-driven RE. The presented work is a part of this effort.

All figures and data in this paper refer to the period 1998-1999. Since then, Telelogic has grown considerably, and Telelogic has continuously introduced improvements in order to meet the challenges of the market. The principal results presented in this paper are thus relevant for understanding market-driven requirements management in general, rather than characterizing the current and future situation at Telelogic. In the following, all references to the "current" or "actual" situation relates to the time-frame from 1998-1999. The simulation study presented in this paper, applies discrete event simulation (Banks et al., 1996) using a queuing network model (King, 1990). A major objective of the simulation study is to explore the conditions under which the process becomes overloaded. It is also investigated which resources are needed in order to handle a certain frequency of new requirements. Simulations are carried out in order to explore the conditions that result in an overloaded process, and to find changes to the process that may remove bottlenecks. The paper is structured as follows. In Section 2, the REPEAT process is presented, and Section 3 presents the simulation model. The results of the performed simulations are presented in Section 4. In Section 5, conclusions and suggestions for further research in the field are presented.

2 The REPEAT process

REPEAT manages requirements continuously by controlling a product pipeline in which three releases are developed in parallel. The product pipeline delivers two new product releases per year. REPEAT covers typical RE activities, such as elicitation, documentation, and validation, and the process has a strong focus on requirements selection and release planning. A schematic picture of the process is shown in Figure 1.

REPEAT is instantiated for each release, and each process instance has a fixed duration of 14 months. Each REPEAT instance consists of five different phases separated by milestones at pre-defined dates. The Elicitation phase

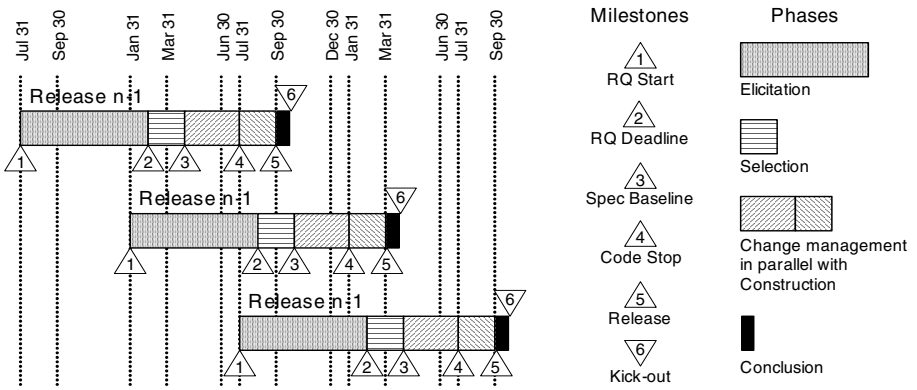


Figure 1: The milestones and phases of the REPEAT process, aligned with a fixed release schedule.

deals with the collection and initial classification of requirements. The Selection phase includes detailed specification of each requirement and release planning. The Change Management phase is active in parallel with construction (design, implementation, and testing of requirements for the coming release) and manages changes in requirements priorities due to events such as emergence of high-priority requirements and delays. The Conclusion phase includes post-mortem documentation. Each of these phases are further described below.

2.1 Elicitation

The elicitation phase includes two activities: collection and classification. Collection of requirements is made by an issuer that fills out a web-form and submits the requirement for storage in an in-house-built database. Requirements are described using natural language and given a summary name by the issuer. An explanation of why the requirement is needed is also given. The issuer gives the requirement an initial priority P, which suggests in which release it may be implemented. P is a subjective measure reflecting the view of the issuer, and is measured on an ordinal scale with three levels, as shown in Table 1.

2.2 Selection

The goals of this phase are:

1. to select which requirements to implement in the current release

2. to specify the selected requirements in more detail

3. to validate the requirements

The output of this phase is a requirements document which includes a selected-list with a detailed specification and effort estimation in hours of all selected requirements, and a not-selected-list including the requirements that are postponed to the next release. The selected requirements are divided into a must-list and a wish-list. The must-list comprises requirements that are estimated to take 70% of the available effort, while the wish-list comprises requirements that are estimated to take 60% of the available effort. This implies that if the effort estimations are correct, half of the wish-list will be implemented, and the rest will be reconsidered for implementation in the next release. However, all the requirements on the wish-list are specified, so if the estimations are not correct there will still be a number of specified requirements to implement in the release.

2.3 Change management during construction

This phase of the REPEAT process is carried out in parallel with the design, implementation, and testing of the requirements, and handles changes in the priorities of the requirements. There are two sub- phases of this phase, one before code-stop (3-4 in Figure 1) and one after code-stop (4-5 in Figure 1). After code-stop no implementation is carried out. Instead the focus is on testing. If new priority-1-requirements are issued, these may be allowed to affect ongoing construction, and in the change management phase the requirements on the must- and wish-list may be rearranged so that new and more important requirements can be incorporated. The 70%-60% rule for the must- and wish-lists must, however, still hold, implying that some less important requirements should be postponed in order to incorporate the new, more important, requirements.

2.4 Conclusion

In this phase metrics are collected and a final report is written that summarizes the lessons learnt from this REPEAT enactment. During 1998 and 1999, the number of unimplemented requirements in the requirements database has increased, and the REPEAT process has at times been in a state of congestion. Process simulation gives the opportunity of investigating the behavior of the process under different circumstances. Results from simulations may provide

quantitative measures, which can act as decision support when allocating resources to different activities in REPEAT.

3 The simulation model

Based on the REPEAT process model (Regnell et al., 1998), an initial simulation model was created, including some major simplifications. The model was then iteratively refined and specialized until it provided an adequate degree of abstraction. As a last step an interview with personnel at Telelogic gave the actual values for the model parameters, along with a confirmation that the model was sufficiently accurate.

3.1 Structure of the model

The REPEAT process simulator is a queuing network model and is implemented using discrete event simulation (Banks et al., 1996). The simulated model is depicted in Figure 2. Requirements enter the simulator from the environment. A requirement must pass the three phases elicitation, selection and construction in order to be included in a release. (The conclusion phase found in Figure 1 was not included in the simulation model as it is independent from the rest of the phases and does not affect congestion and throughput).

In the elicitation phase, incoming requirements are entered into the system and given an initial priority. In the selection phase, the requirements that are to be included in the release are selected, and in the last phase the requirements are constructed. The phases are modeled as processes with a queue of incoming requirements, and a pool of servers which represent the employees. Each phase is thus modeled as a FIFO queue with m servers. Requirements enter the system according to a Poisson-process, and the elicitation phase is therefore an M/G/m queue, while the two other phases are G/G/m queues.

In the elicitation phase, every requirement receives a priority. All requirements having normal priority, i.e. priority 2, are transferred to the selection phase within the current release. Priority 3 requirements are postponed to the selection phase of the next release. Priority 1 requirements are moved to the selection phase of the previous release.

In the simulation model all releases have their own resources. That is, when a release is instantiated in the model, a number of servers in each phase are created. The servers in the selection phase are idle during elicitation, waiting for the selection phase to start. The servers in the construction phase are

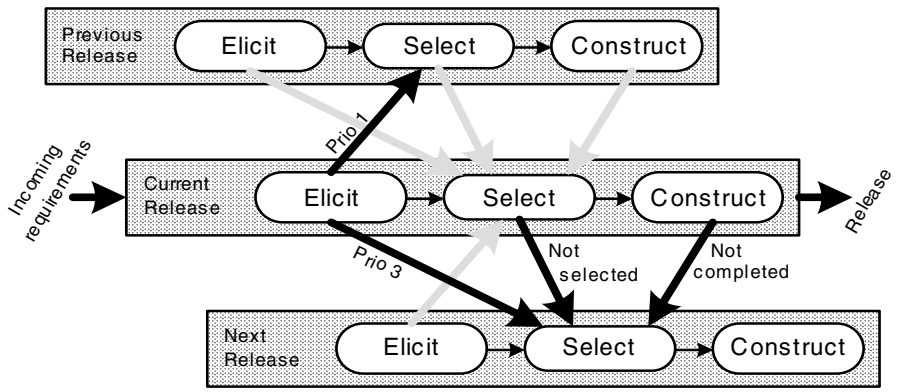


Figure 2: Simulation model.

idle during elicitation and selection, while waiting for the construction phase of the previous release to finish. The personnel that are represented in the simulation model by the servers, are in reality the same persons represented by the servers in the previous release. This is a simple way of modeling that the same persons divide their time between different activities.

During selection, the time each requirement will spend in construction is estimated. A must-list and a wish-list is constructed according to the description of the REPEAT process given above. Requirements that enter either of these lists are transferred to the construction phase of the current release. Requirements that are not selected for either of these lists are sent to the selection phase of the next release, where they may or may not be selected for construction.

When the servers start working they check if it is possible to perform the job next-in-line within the deadline of the release. When the deadline for the release approaches, some requirements may be left uncompleted and sent to the selection phase of the next release. This occurs because there is a parameter-controlled error in the estimation of the required work, and because the wish-list includes more requirements than is possible to construct during one release in order to get a better utilization of the available resources.

All this means that a requirement may pass the selection phase several times during its lifetime. A requirement requires serving time in every phase it passes, which imply a certain amount of overhead when re-routing a requirement to another release.

3.2 Parameter estimation

The simulator accepts a set of input parameters which specify the simulated situation. These input parameters include the number of requirements entering the process each day, the number of available servers (employees) for each phase, and the average time spent on a requirement in each phase (see further Table 1).

The actual values for the parameters are based on data from interviews with an expert from Telelogic. The requirements are modeled to have an exponentially distributed intensity of arrival, i.e. they arrive according to a Poisson process. The distributions of the serving times in the various phases can be modeled in a number of ways. In (Höst and Wohlin, 1998) it is shown that a suitable way to model serving times based on subjective estimates given by domain experts is to use triangular distributions. Based on a triangular distribution, the interviewed expert estimated the smallest possible value, the most likely value, and largest possible value of the serving time for each phase. Data from interviews also provided estimations of parameters regarding the number of employees in each phase, the number of requirements of different priorities, and the average estimation error that is made when estimating the serving time in the construction phase.

The interviews also exhibited that if a requirement has been in one selection phase, and later is sent to a selection phase in another release, it requires only about a fifth of the time spent in the original phase.

The interviews with the expert also concerned general experiences with the REPEAT process. After five releases the requirements database contained almost 2,000 requirements. For each of the five releases, about 75 requirements were implemented, which imply that the requirements database contained about 1,625 unimplemented requirements.

3.3 Discrepancies between simulator and process

There are two significant simplifications in the model. First of all, the server model does not completely match the actual use of employees. In reality there is a single pool of employees containing a number developers working on a number of modules. The single pool of developers work in all three phases (elicitation, selection and construction) for all releases. The simulation model, however, has one pool of servers for every phase of every release. To adjust this, each phase has a parameter indicating when it starts, and the construction phase has a parameter indicating when it finishes, i.e. the release deadline.

Table 1: Simulation parameters[a]

Parameter	Case 1	Case 2	Case 3	Case 4
Time between two consecutive release start-ups	126	126	126	126
Time from start of release to start of selection phase	126	126	126	126
Time from start of release to start of construction phase	168	168	168	168
Length of construction phase	126	126	126	126
Mean time between two consecutive requirements	0.33	0.33	0.33	0.33
Number of servers in the elicitation phase	30	30	1	30
Elicitation time per requirement[b]	(0.010, 0.031, 0.062)	(0.010, 0.031, 0.062)	(0.010, 0.031, 0.062)	(0.010, 0.031, 0.062)
Number of servers in the selection phase	30	30	16	30
Selection time per requirement[b]	(1, 2, 10)	(1, 2, 10)	(1, 2, 10)	(1, 2, 10)
Number of servers in the construction phase	30	30	165	30
Construction time per requirement[b]	(1, 45, 91)	(1, 45, 91)	(1, 45, 91)	(1, 45, 91)
Fraction of requirements of priority 1	0%	10%	10%	10%
Fraction of requirements of priority 3	0%	25%	25%	25%

[a] The unit of parameters representing time is working days.

[b] These parameters are defined according to a triangular distribution (lowest possible value, most likely value, highest possible value) as described in Section 3.2.

The servers of each phase are idle when the phase is inactive. This solution gives an adequate accuracy validated by the interview results.

The second simplification is the way the must- and wish-lists are constructed. The model just takes the first incoming requirements and puts them into the must-list until it is full. Thereafter the wish-list is filled, and late jobs are not selected. This means that priority 1 requirements rarely get selected, as they are sent to the previous release, and arrive there late, when the lists are already full. This simplification can be addressed by changing the simulation model so that requirements are inserted into the lists in a way more similar to the real situation, e.g. by inserting a new job into a random position in the list, whereby the last jobs are pushed off the list. The random distribution can in turn be dependent on the priority of the requirement, or other factors, such as how old the requirement is. These possible enhancements are, however, not implemented in the simulation model as the simulator is not used for investigating the quality of the outcome, but for exploring timing, capacity and throughput.

Another simplification regards the estimated construction effort. Data from the interviews specify the distribution of the total time spent on construction for each requirement. However, large requirements can in reality be divided into several smaller requirements, which in turn can be scheduled over many releases. Therefore, the triangular distribution of the serving times in the construction phase was modified. The maximum serving time was changed from the original 170 days to 91 days, and the most common serving time was changed from 19 days to 45 days in order to produce the same workload. Otherwise the largest jobs would never be implemented in the simulation model.

Other simplifications include the fact that the simulator model never rejects requirements, which is done to a small extent in the actual process. However, requirements are removed so rarely in reality that we do not believe that it affects the validity of the results very much.

In general, we believe that the identified simplifications have insignificant impact on the principal results of the simulations.

4 Model implementation

The model was implemented as a discrete event simulation model in SDL (ITU-T, 1999). A discrete event simulation model was chosen, because it is a straightforward way of implementing models that represent networks of

queues. SDL was chosen because it is based on real-time processes and it supports the creation of discrete event simulators. Another advantage of choosing SDL is that the case tool that the modeled company develops can be used to develop systems in SDL. In fact, the model is implemented with the case tool developed by Telelogic. This means that almost all people at the company understand the notation of the model. The presented research represents the first attempt to model the REPEAT process in a simulation model. In the future it would be possible to carry out not only discrete event simulations, but also systems dynamics simulations of the process.

After the model was implemented, a simple version was created by making all serving times exponentially distributed. Special-case parameters were used which enabled analytical validation of the simulator through e.g. Little's theorem (King, 1990). This proved that the simulator was implemented according to the queuing network model. The complete simulator was then validated by comparing throughput and congestion against data from the expert interviews. The simulator matched the real number of requirements implemented per release, and therefore also the real number of requirements waiting to be implemented after five enactments of the REPEAT process.

5 Results

The results from simulations of the REPEAT process are based on a number of executions of the simulator with various input parameter settings in order to verify the simulation model and draw conclusions from it. The simulation model can be used to analyze many different characteristics of requirements management processes, and the simulation results may act as a valuable decision support.

Four simulation cases are presented in order to illustrate the usage of the simulator and to show the impact of process changes.

The first case is a baseline situation where no prioritization of requirements are made, i.e. all requirements have priority 2. This case is primarily used to verify the model and is presented and analyzed to facilitate comparison with the following cases. The second case introduces prioritization and is based on the actual situation as determined by the data from interviews. In the third and fourth case, changes to the simulation parameters are introduced in order to investigate what amount of increase in capacity or decrease in work load is needed to make the process stable. These values have been found by observing the result for a number of different values. The parameter

values that are used in the simulations are summarized in Table 2.

5.1 Case 1: No priorities

A baseline situation, to which other cases can be compared, is when every elicited requirement is selected and subsequently constructed within the same release, and all requirements have priority 2. In this case, no requirements are re-routed between releases based on priority (requirements are only re-routed when time constraints forces requirements to be forwarded to the selection phase of the next release).

Figure 3 shows the total number of requirements in the selection phases for a number of releases. The selection phases are shown for up to five full releases, with the left-most curve corresponding to release 1 and the subsequent curves corresponding to the following releases. Release 6 and 7 can thus be viewed in part.

The y-axis shows, for each release, the sum of requirements in the queue and requirements currently being processed by the servers. From the time when selection begins, there are always requirements to handle and the servers are never idle. For example, in release 1 there are approximately 400 requirements waiting in the queue ready to be analyzed. Thus, from the time when selection begins the servers are constantly busy until all requirements have been handled.

An important conclusion that can be made from the figure is that this process is overloaded. In the fifth release there are approximately 1,600 requirements waiting to be handled in the next selection phase and the amount increases to the next release. There are approximately an additional 250 requirements for each release.

There is no rejection of requirements in the simulation model. All requirements entering the elicitation phase are kept in the system until released. The requirements that the construction servers do not manage to implement are forwarded to the next release. As the figure shows, the number of requirements in the queue is constantly increasing. Rejecting some requirements, such as duplicates or obsolete requirements, would reduce the queue build up. From the interviews it is found that in reality only about 5-10% of the total number of requirements are rejected. This low rejection rate is not enough to make the process stable.

In release 2, a slight deflection at day 180 from the start of the simulation can be noticed. This happens when requirements no longer are forwarded from the previous release.

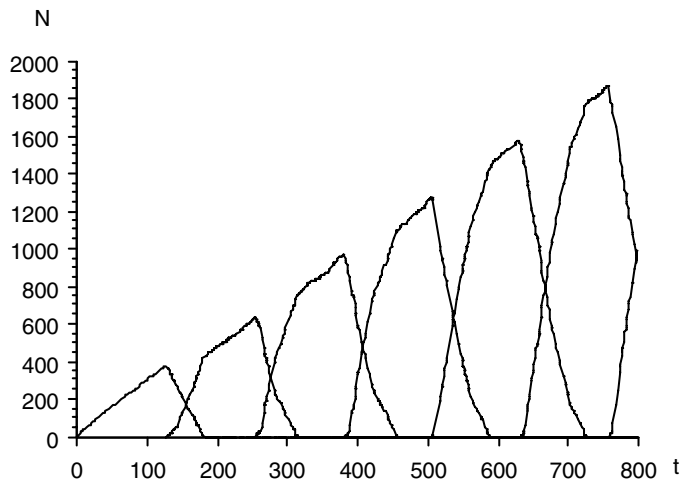


Figure 3: Case 1: No priorities. Number of requirements in the selection phases of releases 1-5 and parts of release 6 and 7.

In release 1, the slope of the first half of the curve corresponds to that of the arrival intensity. In the subsequent releases the slope of the curves, after the deflections, also corresponds to the arrival intensity. We can therefore draw the conclusion that the elicitation phase does not get overloaded; it is selection and construction that is in a state of congestion. This conclusion can be verified by a separate analysis of the elicitation phase.

Figure 4 shows the results from the simulation of the construction phases of subsequent releases. As before, the graph shows releases 1 through 5 from left to right. The construction phase is situated later in time and consequently we can only see a part of release 6.

This graph shows both the number of requirements in the queue overlaid with the number of requirements currently being handled in the servers. Focusing on one release, as shown magnified in Figure 5, it can be seen that the queue builds up rather fast. The fast build-up of the queue (105 requirements in about 30 days) can be derived from the selection phase graph (Figure 3). By observing how many requirements that are handled from the point when the selection phase begins and 30 days ahead, it is clear that the decrease corresponds to the number of requirements arriving to the construction phase during the same time period.

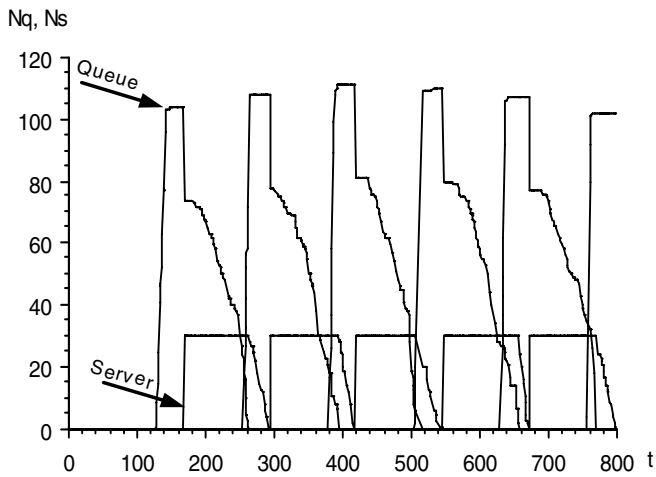


Figure 4: Case 1: No priorities. Number of requirements in the construction phases of releases 1-5.

The increase of the number of requirements in the construction queue of release 1 abruptly stops at 105 requirements. The reason for this is that when the total effort of the requirements transferred to the construction phase equals the available time in the construction phase, no more requirements arrive from the selection phase.

When construction begins there is a large drop from 105 requirements in the queue to only 75 requirements. This is when all 30 available servers are occupied at once. Then, the servers are constantly busy until there are no more work to do, continuously taking care of the requirements in the construction queue.

It may be tempting to read off the graph that 105 requirements are implemented in the first release. This is unfortunately not true. As it is stated in Section 2, only about half of the wish-list will be implemented. Since the needed effort is not equal for every requirement in the construction phase, it is not possible to use the graph to calculate the number of requirements that is actually implemented. For each release, a certain percentage of the requirements arriving to the construction phase will actually be implemented. The remaining requirements are sent to the selection phase of the next release.

In Figures 4 and 5 it can be seen that when a succeeding release receives requirements to the selection phase, the construction phase of the current re-

Figure 5: Case 1: No priorities. Number of requirements in the construction phase of release 1.

lease has not yet finished and implementation is still undertaken. When there are no more requirements to implement the servers representing employees enter an idle state and are ready to get to work in the next release. This can be seen in release 1 approximately after day 260.

5.2 Case 2: Actual situation

Using the parameters determined from expert interviews (see Section 3.2) the actual situation can be simulated. Prioritization is now introduced, as explained in Section 2. This will result in some requirements being transferred to the next release and a few requirements being sent to a previous release still under development.

Figure 6 shows the selection phases of release 1 through 5 as before. We can see that there is not much difference from the previous case. As before, the selection phase is overloaded. This is because of the time constraints in the construction phase, and because there is no rejection of requirements. If the capacity is not enough to take care of all requirements, priorities will not completely solve the situation. Priorities will however make the organization focus on the most important requirements.

The reason that no curve, from release 2 and onwards, start at zero re-

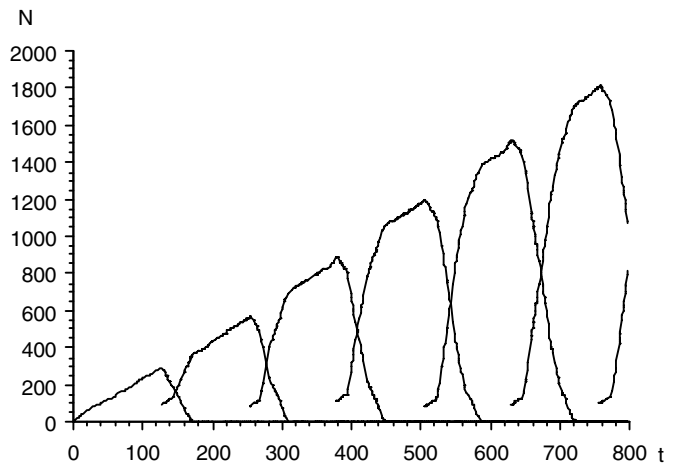


Figure 6: Case 2: Actual Situation. Number of requirements in the selection phases of releases 1-5.

quirements requires an explanation. In the simulation model a subsequent release and its queue is not started until it actually should be, as shown in Figure 3. Thus, when the simulation of a release is started, there are some requirements already waiting from a previous release. Here the first part of the curve corresponding to the initial build-up of the queue is not shown.

Since no conditions for the construction phase has been changed compared to case 1, Figure 77, showing the construction phase, does not differ much. About the same number of requirements, on average, are implemented when we add prioritization. The analysis of the construction phase made for case 1 is appropriate for this case as well.

5.3 Case 3: Increased capacity

A first obvious change of the process in order to remove bottlenecks would be just to reallocate resources or adding more people. This alternative is simulated in order to ascertain what the impact would be and to find the amount of resources needed (or the needed increase in productivity) to reduce the bottlenecks.

Figure 8 shows the selection phases of 13 releases and Figure 9 shows the construction phase of release 1 after increasing the number of servers in

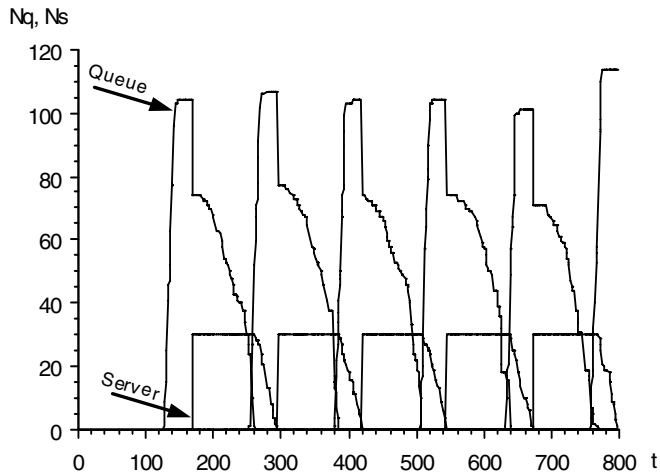


Figure 7: Case 2: Actual Situation. Number of requirements in the construction phases of releases 1-5.

these phases in order to make the process stable. Many releases are shown in Figure 8 to support the assumption of stability. To make it stable, 16 employees are required in the selection phase and as many as 165 persons in the construction phase. This means an increase in construction capacity by a factor $165/30 = 5.5$. Elicitation is not a bottleneck, as it is enough with only one elicitation server (see Table 2).

It can be seen in Figure 9 that when the construction starts (day 168) there is enough server capacity available to take care of every requirement in the queue and additional ones arriving in the following 10 days. However, the utilization of the resources in the construction phase is lower than before.

5.4 Case 4: Decreased work load

Another approach to remove the state of congestion is to reduce the number of requirements that arrive to the elicitation phase and thereby reduce the number of requirements that are forwarded to the selection phase.

In Figure 10, the impact of reducing the arrival intensity is shown. Here the arrival intensity has been decreased from the value from reality of 3 requirement a day to only 0.55 requirements a day. This means a decrease in the rate of incoming requirements from 3 to $1/1.8 = 0.55$, which corresponds

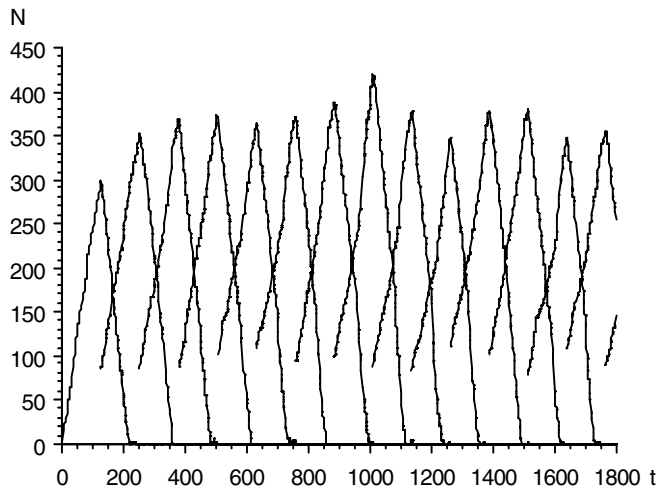


Figure 8: Case 3: Number of requirements in the selection phases of releases 1-13.

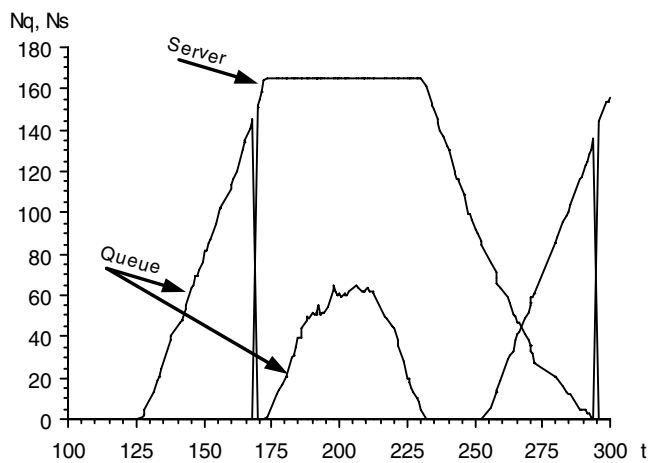


Figure 9: Case 3: Number of requirements in the construction phase of release 1.

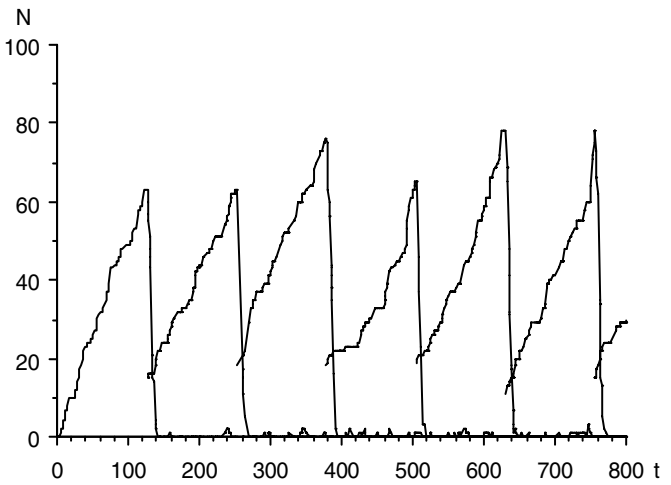


Figure 10: Case 4: Number of requirements in the selection phases of releases 1-5.

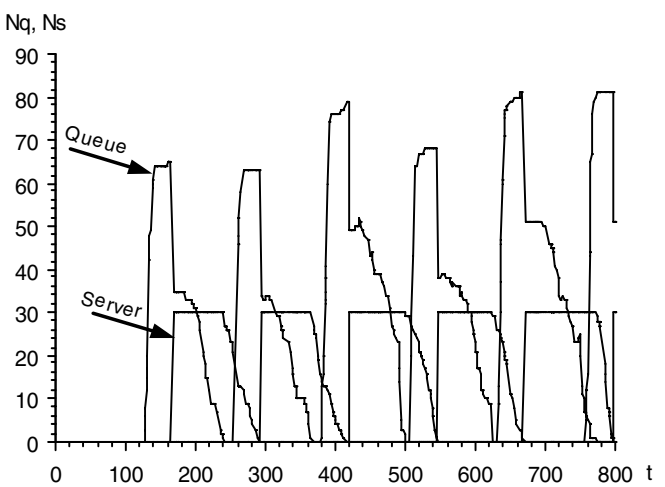


Figure 11: Case 4: Number of requirements in the construction phases of releases 1-5

to $0.55/3 = 18\%$ of the actual value. This reduces the maximum number of requirements in the selection phase to about 80 requirements, which is a notable decrease.

As Figure 11 shows, only about 80 requirements are received to the construction phase for each release. However, now there are resources enough to implement every requirement. The requirements in the selection phase can all be forwarded to the construction phase and implemented.

6 Conclusions

This paper presents a study that shows how discrete event simulation can be used in order to explore overload conditions of an industrial software requirements management process for packaged software. The simulation model is created based on a previous study of the process (Regnell et al., 1998) and the simulation parameters are estimated based on interviews with a process expert with in-depth knowledge of how the process performed during the studied period of 1998-1999. The situation of overload that has been observed in reality can also be observed when executing the simulation model.

It can be concluded from the simulations that there are at least two different ways of changing the process in order to avoid congestion:

- The capacity of the requirements management process can be improved, either by increasing the number of employees or by improving productivity. The simulations show that it is necessary to increase the capacity of the construction phase by a factor 5.5 in order to completely remove the bottlenecks.

- The workload on the requirements management process can be decreased. The simulations show that it is necessary to decrease the rate of issuing new requirements to 18% of the initial value. One way of lowering the workload is through early prioritization (Karlsson et al., 1998) and thereby rejecting requirements that will not be implemented at an early stage in the process. Of course, this is an area that needs further investigation, as it is always difficult to remove issues early in a process. If the objective is to remove requirements in order to lower the effort required in analysis, then an analysis effort is required in order to know which requirements to remove. Prioritization of requirements and release planning in packaged software development is acknowledged to be an important research area (Regnell et al., 2000).

In conclusion, the presented simulations represent a feasible way of analyzing the investigated process. However, the simulation model can be improved in a number of ways. One improvement regards a more realistic modeling of the must- and wish-lists (see Sections 2-3). Another area of further work is to make a more thorough analysis of the real process by interviewing more people representing more roles in the organization about their opinions concerning both the parameter values of the model and the validity of the simulation results.

It is also interesting to extend the simulation model by modeling the servers as a single pool of resources where each resource (employee) has certain competencies. This may lead to a more realistic simulation model, where the same persons are involved in many tasks, and, as in reality, the lack of a certain competency may be a bottleneck.

A promising benefit of the presented approach is the potential of achieving validated decision support that can facilitate informed decisions on improvements of software processes in general, and market-driven requirements engineering processes in particular.

Acknowledgements The authors would like to thank all people involved in the development and investigation of REPEAT, in particular Per Beremark (Group Quality Manager at Telelogic) without whom this work would not have been possible. We would also like to thank Carina Andersson and Lena Karlsson, both with the Dept. of Communication Systems, for providing suggestions for improvements of the simulation model. The presented research is partly funded by the National Board of Industrial and Technical Development (NUTEK).

References

Banks, J., Carson, J. S., and Nelson, B. (1996). *Discrete-Event System Simulation*. Prentice Hall, Upper Saddle River, NJ, 2nd edition.

Höst, M. and Wohlin, C. (1998). An experimental study of individual subjective effort estimations and combinations of the estimates. In Werner, B., editor, *Proceedings of 20th International Conference on Software Engineering*, pages 332–339, Los Alamitos, CA. IEEE CS.

ITU-T (1999). *Specification and Description Language (SDL)*. International Telecommunications Union, Geneva, Switzerland. ITU-T Recommendation Z.100-11/99.

Karlsson, J., Wohlin, C., and Regnell, B. (1998). An evaluation of methods for prioritizing software requirements. *Information and Software Technology*, 39:939–947.

Kellner, M. I., Madachy, R. J., and Raffo, D. M. (1999). Software process simulation modeling: Why? what? how? *Journal of Systems and Software*, 46:91–105.

King, P. J. B. (1990). *Computer and Communication Systems Performance Modelling*. Prentice-Hall, London, UK.

Lubars, M., Potts, C., and Richter, C. (1993). A review of the state of the practice in requirements modeling. In *Proceedings of IEEE International Symposium on Requirements Engineering*, pages 2–14, Los Alamitos, CA. IEEE CS.

Pfahl, D. and Lebsanft, K. (2000). Using simulation to analyse the impact of software requirement volatility on project performance. In *Project Control: The Human Factor, Proceedings of the combined 11th European Software Control and Metrics Conference and the 3rd SCOPE conference on Software Product Quality*, pages 267–275, Maastricht, The Netherlands. Shaker.

Potts, C. (1995). Invented requirements and imagined customers: Requirements engineering for off-the-shelf software. In *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, pages 128–130, Los Alamitos, CA. IEEE CS.

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

Regnell, B., Höst, M., Natt och Dag, J., Beremark, P., and Hjelm, T. (2000). Visualization of agreement and satisfaction in distributed prioritization of market requirements. In amd Klaus Pohl, A. L. O. and Rossi, M., editors, *Proceedings of the Sixth International Workshop on Requirements Engineering: Foundation for Software Quality*, Essen, Germany. Essener Informatik Beiträge.

Sawyer, P. (2000). Packaged software: Challenges for RE. In *Proceedings of Sixth International Workshop on Requirements Engineering: Foundation for Software Quality*, pages 137–142, Essen, Germany. Essener Informatik Beiträge.

Sawyer, P., Sommerville, I., and Kotonya, G. (1999). Improving market-driven RE processes. In *Proceedings of International Conference on Product*

Focused Software Process Improvement (PROFES'99), pages 222–236, Oulu, Finland.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2000). *Experimentation in Software Engineering - An Introduction*. Kluwer, Norwell, MA.

Yeh, A. (1992). Requirements engineering support technique (REQUEST) - a market driven requirements management process. In *Proceedings of the Second Symposium on Assessment of Quality Software Development Tools*, pages 211–223, Los Alamitos, CA. IEEE CS.

PAPER II

A Feasibility Study of Automated Support for Similarity Analysis of Natural Language Requirements in Market-Driven Development

Johan Natt och Dag, Björn Regnell, Pär Carlshamre, Michael Andersson, Joachim Karlsson

Requirements Engineering, 7(1), 20–33, 2002

II

Abstract

In market-driven software development there is a strong need for support to handle congestion in the requirements engineering process, which may occur as the demand for short time-to-market is combined with a rapid arrival of new requirements from many different sources. Automated analysis of the continuous flow of incoming requirements provides an opportunity to increase the efficiency of the requirements engineering process.

This paper presents empirical evaluations of the benefit of automated similarity analysis of textual requirements, where existing Information Retrieval techniques are used to statistically measure requirements similarity. The results show that automated analysis of similarity among textual requirements is a promising technique that may provide effective support in identifying relationships between requirements.

1 Introduction

1.1 Background

The market-driven development organization faces many challenges that differ from those found in organizations developing bespoke software. Software is developed for a large market, rather than for a specific customer, new versions are developed in a succession of releases, and there is a high pressure on short time-to-market (Lubars et al., 1993; Sawyer et al., 1999; Deifel, 1999). To meet market demands it is important to have an effective and efficient requirements engineering process. Special demands are set as requirements arrive continuously at a high rate from many different sources during the whole development process (Regnell et al., 1998). As there is no single specific customer to negotiate with, requirements must be invented within the developing organization based on foreseen end-user needs (Potts, 1995). These invented requirements may come from sources such as marketing, support, development, testing, usability evaluations, and technology forecasting and are often collected for storage in a database. The requirements engineering activities are then focused on analyzing and prioritizing the requirements in the database and on maintaining the database for the future.

In this study we have focused on a large software developing company, Telelogic AB, that develops a CASE tool for the world-wide telecommunications market. Their development process is described in (Regnell et al., 1998) and its main properties are: Releases are pipelined to enable a new release every sixth month while each release takes 14 months to complete. Elicitation is continually active and a requirement may be issued at any time by an issuer that foresees a market need. Each requirements is stored in a database as an entity described in natural language. Each requirement has a life-cycle progression through specific states. The Telelogic development process has shown having high resemblance with another market-driven development process independently developed and used at an Ericsson company (Carlshamre and Regnell, 2000). Requirements are continuously collected through a web form and are stored in a database for further analysis (Regnell et al., 1998). The requirements are described in natural language and are of varying quality and nature. Some requirements are brief ideas while others are detailed descriptions of new features with accompanying code. Many requirements are short-worded and poorly written.

During the development of a release the requirements engineer (or analyst) must handle the diverse and large set of requirements that is available in

the database and resolve ambiguities, find relationships, eliminate duplicates, etc. As shown in a study of the Telelogic requirements process (Höst et al., 2001), these activities are causing a congestion that may be avoided by cutting down heavily on the number of elicited requirements or making early and strict prioritization.

The trade-off between analyzing only a subset of all the collected requirements and not collecting that many requirements to give time for proper analysis may be difficult to make. Extra information could be extracted if all requirements are collected (for example, duplicates may indicate that certain issues are more important than others). However, trying to handle all incoming requirements may increase the risk of relationships between requirements being overlooked or discovered too late, which may cause problems in prioritization (Karlsson and Ryan, 1997) and release planning (Carlshamre and Regnell, 2000).

Consequently, there is a wish to find requirements relations early, without spending too much time on in-depth analysis. These relationships should preferably be found even when specification quality is low and even if requirements are short, poorly worded or misspelled. One possible approach, investigated in this paper, is to assist the requirements engineer through automated analysis of the textual information in the requirements. This approach may help the requirements engineer to handle the large set of requirements by automatically finding and make suggestions on relationships between requirements.

Two different automatic text processing approaches may be used to aid the requirements engineer in the situation described above: the statistical approach or the linguistic approach. In this paper we focus on the statistical approach, which originates from the work by H. P. Luhn (Luhn, 1957). There are several reasons that we choose to explore this approach:

1. The ideas have not, as we far as we know, been applied to analyst the type of requirements that is collected in the situation we describe. (see further Section 1.2).

2. The statistical approach has been thoroughly tried and examined and has been found fairly successful for automatic text analysis (Van Rijsbergen, 1979).

3. The linguistic approach is still regarded expensive to implement and not always more effective than well-executed statistical approaches (Mitra et al., 1997).

4. Before proceeding with more advanced methods, the statistical approach may help reveal the nature of the requirements in a market-driven organization.

5. A baseline produced from empirical investigation using real industry requirements is needed to compare against further improvements.

The results of the presented work show that, for a particular set of requirements, a simple similarity analyzer that uses the statistical text processing approach identifies a large fraction of the requirements duplicate pairs found by experts. The duplicates are important to find to avoid doing the same job twice, assigning the same requirement to different developers, or getting two solutions to the same problem. The portion of requirement pairs incorrectly identified as duplicates is shown to have little negative impact on the value of the method. Further effort may thus be fruitful to assist the requirements engineer in handling the large set of requirements found in a market-driven development organization.

1.2 Related work

The role of natural language processing in requirements engineering is discussed in (Ryan, 1993) where the conclusion is drawn that natural language processing techniques must be realistic and effort has to be made to identify where such techniques may be useful. It is argued that the validation of requirements still have to be an informal, social process. Thus, an automated system could or should not replace the human requirements engineer. Such systems are still not feasible or cost-effective to construct.

Various attempts have been made to use automated techniques to assist the analysis of requirements written in natural language:

1. Gervasi (2000) use lightweight formal methods (low cost, partial analysis) to partially validate a syntactically correct NASA Software Requirements Specification (SRS) document. A glossary was manually produced from the SRS to aid the method.

2. Ambriola and Gervasi (1997) present a web-based environment where Model-Action-Substitution-rules and a domain- and system-specific glossary are used to extract abstractions and build models.

3. Rayson et al. (2000) report on a project called REVERE, where statistical and probabilistic natural language processing methods are used to assist the analysis of complex and voluminous texts.

4. Park et al. (2000) present a system that uses a sliding window model and a parser to support the analysis of requirements using a similarity measuring technique.

5. Rolland and Proix (1992) present an environment that generates conceptual specifications from problem space descriptions written as sentences in natural language.

6. Osborne and MacNish (1996) describe an approach to resolve ambiguities where only a controlled language is allowed when writing requirements in order to facilitate for a lexicon and grammar enabled parser.

7. Cybulski and Reed (1998) describe an elicitation method and a supporting management tool that help analyzing and refining requirements by using a parser, semantic networks, a domain-mapping thesaurus, and faceted classification schemes to allow proper formalization of requirements written in natural language.

8. Chen et al. (1994) present ideas where concepts in texts from electronic meetings are automatically classified by using automatic indexing, cluster analysis and hopfield net classification.

9. Landauer and Dumais (1997) present the Latent Semantic Analysis (LSA) computational model for generation of a representation from large corpora. The representation captures the similarity of meanings of words and sets of words.

Although relevant and promising for several areas and approaches in requirements engineering, the above attempts do not address the situation described in the previous section. The main concerns in the context of this work are the following:

- Requirements are considered to be found in a separate document that is to be analyzed, quality assured and produced before implementation begins. This is not the situation in the market-driven organization where requirements arrive continuously and may, at any time, affect both previous, current and coming releases of the software.

- The initial quality of the requirements are often considered to be adequate for semantic parsing. This may not be the case when requirements are collected from many different sources and stored in a database.

- Real industrial requirements are not always used to validate the methods or techniques presented. Accuracy and efficiency is not always reported.
- The semantic nature of invented requirements may not share the properties of regular corpora used in many linguistic approaches.
- Simple, robust methods can act as a baseline for better understanding and further improvements and comparisons of techniques.

Several approaches seem promising but we believe that more effort need to be put into this field to reach consensus on which methods, techniques, approaches and tools that may be appropriate for different types of developing organizations. In this paper we focus on the market-driven organization and do not present a new model or a full-featured approach. Rather, the feasibility of using automated similarity analysis is empirically investigated using real industrial requirements and a benchmark is provided to which further effort may be compared.

1.3 Paper structure

The paper is structured as follows. In Section 2 the situation of requirements similarity analysis in market-driven development is described. Section 3 explains how automated similarity analysis of natural language requirements may be performed. Section 4 presents a case study where actual requirements collected from industry have been analyzed. The case study explores the quality of a simple automatic similarity analyzer. In Section 5, further applications of automated support are presented together with a small study using the analyzer from Section 3 to investigate if similar requirements also are interdependent. Section 6 identifies possible further work and improvements. In the final section the results are discussed and conclusions presented.

2 Requirements similarity analysis

Requirements carry information on which decisions are based. This information can be either explicit or implicit. The explicit information constitutes all the written text, drawn charts, and other artifacts that are used as the basis for communicating requirements. The implicit information are all the assumptions, rules, standards, and the domain knowledge possessed by the requirement issuers and the requirements analyst. When natural language requirements arrive at a rapid flow from many different issuers, a quick analysis

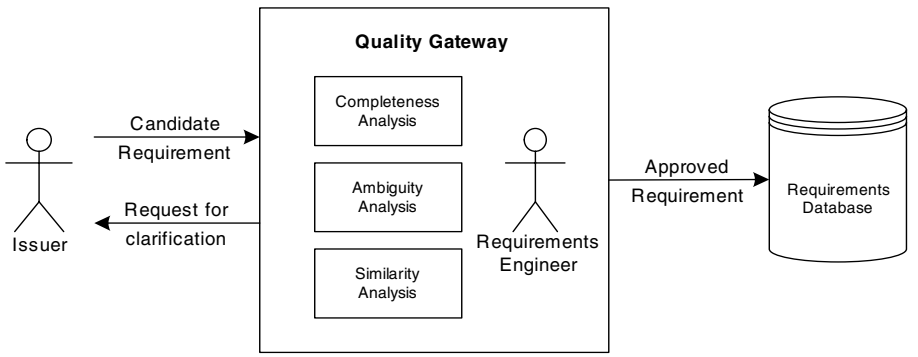


Figure 1: Requirements Quality Gateway with three examples of quality assuring activities.

is required to guarantee requirements quality before they are used as a basis for further decisions. Although the linguistic quality of the requirements may be low it is often left unattended as the requirements make sense. Rather, the information explicitly stated may not give sufficient decision support. For this reason the requirements engineer uses implicit and explicit information accompanied with personal skills to analyst the requirements for completeness, ambiguity, similarity, etc. Completeness analysis is performed to ensure that enough information is included in the requirements to enable further refinement, such as setting priority, estimating effort and deriving new requirements (see example requirements in Fig 4). Ambiguity analysis is performed to identify the risks of multiple interpretations among requirements. Similarity analysis is discussed below.

If supplementary information is needed to accept the requirement, the analyst may have to consult the issuer to make sure that the issuer and the analyst share the same interpretation. Thus, the requirements engineer acts to assure the quality of each requirement before allowing it to be further refined in the continuous requirements engineering process (Carlshamre and Regnell, 2000).The situation is illustrated in Figure 1 where example activities have been identified in the *quality gateway*.

The activities in the quality gateway are typically performed manually as there are few supportive tools available. The activities are tedious and time-consuming, but necessary in order to assure software quality and to satisfy market needs. It would therefore be highly beneficial if some of these tedious activities could be partly automated.

This paper focuses on similarity analysis, which is performed in order to

find requirements that may be merged, grouped, eliminated or linked. For example, two similar requirements may be merged into one or may simply be grouped together to make sure they are handled simultaneously during development. A requirement may be similar to another to the extent that it is regarded as a duplicate and thus eliminated. Furthermore, two requirements may be similar in a certain aspect that establishes some kind of interrelationship (such as dependencies between requirements and requirement decompositions). The requirements engineer may also find it desirable to split large requirements into two or more requirements, which may become similar or related to each other and other requirements in the database.

When the requirements engineer decides whether two requirements are similar or not, it is with regard to the implications for further development. Of course these decisions are made by humans, but computer analysis of explicit information expressed in natural language may supply the requirements engineer with information regarding similarity to support these decisions.

3 Automated similarity measurement

When the requirements engineer decides whether two requirements are similar or not, it is with regard to the implications for further development. Of course these decisions are made by humans, but computer analysis of explicit information expressed in natural language may supply the requirements engineer with information regarding similarity to support these decisions.

Statistical approaches to automated similarity measurement are widely used in information retrieval (IR), which is a well established discipline concerned with automated storage and retrieval of documents written in natural language (Frakes and Baeza-Yates, 1992). The presented work is based on existing IR techniques applicable in the analysis of natural language requirements. Figure 3 provides an overview of the steps in similarity measurement, where a similarity metric $S_{A,B}$ is calculated for a pair of textual requirements (A, B). The calculation of a similarity measure (further described in Section 3.1) is made subsequent to a number of preprocessing steps (elaborated in Section 3.2). The assessment of similarity metrics is described in Section 3.3.

3.1 Similarity measures

In order to find relationships between requirements that may be merged, grouped or eliminated, a quantification of the degree of association between the requirements is needed. Several similarity measures are available, but no

comparative studies exist that give a definite answer to which one to choose in this particular situation. In this paper we have therefore used three simple and well-known similarity measures to calculate the similarity between sentences: the Dice, Jaccard and cosine coefficients (Salton, 1989). These measures all take the words in two sentences and calculate the similarity based on how many words they have in common. The coefficients are defined as follows, where A and B are requirements:

$$S^{D}_{A,B} = \frac{2|words_A \cap wordsB|}{|words_A| + |wordsB|} \tag{II.1}$$

$$S^{J}_{A,B} = \frac{|words_A \cap wordsB|}{|words_A| + |wordsB| - |words_A \cap wordsB|} \tag{II.2}$$

$$S^{C}_{A,B} = \frac{|words_A \cap wordsB|}{\sqrt{|words_A||wordsB|}} \tag{II.3}$$

All three measures have the desired property of normalization, which imply that they give a value between 0 and 1 to indicate how similar a pair of sentences are, where 0 means that the sentences have no words in common and 1 means that the sentences are identical. The empirical investigation reported in Section 4 applies these measures to textual software requirements.

3.2 Preparing the source data

Before the similarity measure can be calculated the words of each sentence have to be extracted. This is achieved through lexical analysis, which takes an input stream of characters and converts it into a stream of words or tokens. This immediately raises the question of what should count as a word or token. For example, digits, hyphens, punctuation and letter case bring some problems that have to be considered. It is not technically difficult to solve these problems, but the chosen lexical analysis policy will affect the similarity measure. For example, preserving letter case will distinguish the words like 'System' and 'SYSTEM' and thus produce lower similarity measures. How to choose the policy thus depends on what type of data is to be analyzed and the expected outcome.

Frequently occurring words like 'a', 'the', 'of', etc., will inadequately boost the similarity measures. These words, known as stop words, are therefore filtered out before similarity calculation. Which words to eliminate again

depends on the type of data. It is reasonable to start out with a known stop word list that has been derived from general text.

Another issue is the *morphological variants* of words, i.e. the word forms. Words that are written in different forms usually carry the same information and should thus be considered equal. Therefore, words should be reduced to their ground form so that an automated word matcher would report a positive match. The technique used to reduce words to their ground form is called stemming and produces a stem from a word. For example, both the words 'replace' and 'replacement' may result in the stem 'replac' and consequently the words would be considered equal. There are several ways to stem words, such as affix removal, successor variety, table lock-up, and n-gram (Frakes and Baeza-Yates, 1992). In this paper we have used an affix removal stemmer, the Porter algorithm (Porter, 1980), which consists of a set of condition/action rules. It is a compact algorithm that has been shown to give good results in IR (Frakes and Baeza-Yates, 1992). The similarity measure may be calculated by counting the number of stems produced from each requirement and the number of stems the requirements has in common. The common stems may be found using *exact match* or *inexact match*. Exact match requires the stem to be exactly equal, whereas inexact match calculates the similarity between the stems. Spelling errors may call for inexact match but brings the difficulty of choosing a good algorithm and a threshold level for match. The analyzer used in this paper is designed to require an exact match between stems. The low linguistic quality of the requirements will of course affect the similarity measure. However, we have chosen not to include spelling correction, as we are interested in the performance of using a simple technique. It is also questionable if there is time for manual pre-processing in industrial settings.

3.3 Assessing the quality of similarity measures

In order to evaluate the technique used to suggest similar requirements, a notion of quality is needed. We have chosen to use a contingency table, which defines a number of quality aspects in similarity measurement. Assume that $S(r_i, r_j)$ is a function that takes a pair of requirements and gives a similarity measure between 0 and 1. In addition we select a threshold value t, which acts as a selection criterion. If $S(r_i, r_j) > t$ then (r_i, r_j) is considered to be a suspected duplicate pair. Assume also that there exists a set of pairs of requirements that are identified as actual duplicate pairs. The similarity measure hence provides an approximation of this set of actual duplicate pairs, and the quality of the estimation may be defined according to Figure 2 (Salton,

	Below similarity threshold	Above or equal to similarity threshold	Total
Actual non-duplicates	A True negatives	B False positives	A+B
Actual duplicates	C False negatives	D True positives	C+D
Total	A+C	B+D	A+B+C+D

$$\text{True positives rate} = \frac{D}{(C+D)}$$

$$\text{False positives rate} = \frac{B}{(A+B)}$$

$$\text{Accuracy} = \frac{(A+D)}{(A+B+C+D)}$$

Figure 2: Assessment scheme with contingency table

1989).

The resulting pairs that have a similarity value above or equal to the threshold level are regarded as duplicate pairs suggested by the analyzer. Matches between actual duplicate pairs and those suggested by the analyzer are denoted true positives. The actual duplicate pairs not identified by the analyzer are consequently denoted false negatives, i.e. they were wrongly suggested as non-duplicate pairs. The analyzer may also suggest duplicate pairs that actually were non-duplicate pairs. These are denoted false positives. The rest are denoted true negatives and constitute all the requirement pairs that fell below the threshold level and were correctly suggested as non-duplicate pairs. The accuracy of the analyzer is defined as the sum of the true negatives and the true positives divided by the total number of possible requirement pairs and indicates how well the actual duplicate pairs and non-duplicate pairs are identified. The total number of requirement pairs is calculated as $A + B + C + D$, which is equal to $(n \cdot (n-1))/2$, where n is the number of requirements.

The contingency table will help reveal the performance of the method. In order to evaluate the feasibility of the analyzer, a deeper investigation of the requirement pairs is needed. Taking any two identified pairs, they may or may not involve the same particular requirements. For example, the requirement pairs (A, F) and (C, F) share the requirement F. If the analyzer assigns similarity values above zero to each of these pairs and a similarity value equal to zero to the pair (A, C) it would nevertheless be interesting to look at the

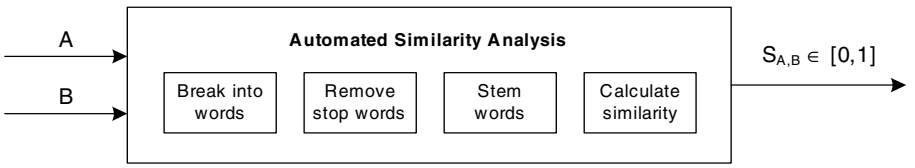


Figure 3: A functional view of automated similarity analysis between requirement A and B, producing a measure $S_{A,B}$ ranging from 0 to 1.

three involved requirements together. We denote these preferred groupings of requirements as n-clusters, where n is the number of requirements in the cluster. The two single pairs in the previous example will thus form a 3-cluster. The cluster distribution can be derived by calculating the transitive closure of a graph in which the nodes correspond to requirements and edges correspond to pairs of requirements (r_i, r_j) with $S(r_i, r_j) > t$. The sizes of the clusters and the number of clusters reveal the usefulness of the automated similarity analysis. It may be desirable to have many requirements grouped into n-clusters where n is the greatest number of requirements that the requirements analyst is capable of handling simultaneously. Example cluster distributions are presented in Figure 9 and Figure 10.

4 Empirical investigation

In order to investigate the potential benefits of automated similarity analysis, we have applied the similarity measures described in Section 3.1 to real industrial requirements. The measures were used to see if automated analysis can correctly determine if a certain requirement is a duplicate of an already existing requirement.

For the investigation we have developed a computer program to perform the tasks specified in Figure 3. The pre-processing steps are handled by a lexical analyzer, a stop word remover and a stemmer (explained in Section 3). The stop list remover excludes words with low discrimination value, and consists of 425 words derived from the Brown corpus (Francis and Kucera, 1982). For the stemming of words, the Porter algorithm is applied (Porter, 1980). The similarity calculation produces a list of requirement pairs along with a value for each pair representing the similarity measure.

Telelogic, a large software developer, has allowed us restricted access to a requirements database of 1,920 confidential requirements. Telelogic devel-

Table 1: Number of requirements in the database and in the different sets prepared for analysis.

Status	Original	A_{full}	$A_{reduced}$
New	406	406	12
Assigned	428	428	31
Classified	601	601	601
Implemented	252	252	252
Rejected	103	103	103
Duplicates	130	101	90
Total	1,920	1,891	1,089
Duplicate pairs	-	142	124

ops software development tools for a wide market and handles requirements arriving at a high rate from several different stakeholders (about three requirements a day (Höst et al., 2001)). The requirements are submitted through a web interface and thereafter managed by requirements engineers (Regnell et al., 1998).

In Figure 4, two examples of requirements from the database are shown. Many of the attributes are set at different stages in the requirements process, reflecting the refinement of the requirement from submitted to implemented or rejected (Regnell et al., 1998). The stage is represented by the 'Status' and the possible stages are shown in the leftmost column in Table 1. The table also shows, in the second column, the distribution of the 1,920 requirements over the different stages.

4.1 Preparations

When a requirements engineer analyzes a requirement, the requirement is checked on many different properties. Three related properties are (1) whether or not it is regarded as a duplicate of another requirement already in the database, (2) if it is possible to merge it with another requirement and (3) if it should be split into two or more requirements before further analysis. If a requirement has one of these properties, it is assigned the 'Duplicate' status and an appropriate action is taken. When a requirement is merged, all the information is added to the requirement it is merged with. When a requirement is split, the information is distributed over two or more new requirements. When a requirement is a pure duplicate (property 1 above), no further action

Figure 4: Two example requirements denoted duplicates in the database. These two requirements were also suggested as duplicates by the similarity calculator at the 0.75 threshold level using the cosine similarity measure.

RqId	RQ96-270
Date	
Summary	Storing multiple diagrams on one file
Why	It must be possible to store many diagrams on one file. SDT forces to have 1 diagram per file. It's like forcing a C programmer to have not more than one function per file... The problem becomes nasty when you work in larger projects, since adding a procedure changes the system file (.sdt) and you end up in a mess having to "Compare systems".
Description	Allow the user to specify if a diagram should be appended to a file, rather than forcing him to store each diagram on a file of its own.
Dependency	4
Effort	4
Comment	This requirement has also been raised within the multiuser prestudy work,but no deeper penetration has been made. To see all implications of it we should have at least a one-day gathering with people from the Organizer, Editor and InfoServer area, maybe ITEX? Här behövs en mindre utredning, en "konferensdag" med förberedelser och uppföljning. Deltagare behövs från editor- och organizergrupperna, backend behövs ej så länge vi har kvar PR-gränssnittet till dessa.
Reference	
Customer	All
Tool	Don't Know
Level	Slogan
Area	Editors
Submitter	x
Priority	3
Keywords	storage, diagrams, files, multi-user
Status	Classified

RqId	RQ97-059
Date	Wed Apr 2 11:40:20 1997
Summary	A file should support storing multiple diagrams
Why	ObjectGeode has it. It's a powerful feature. It simplifies the dayly work with SDT. Easier configuration management. Forcing one file for each procedure is silly.
Description	The SDT "Data model" should support storing multiple diagram on one file.
Dependency	4
Effort	1-2
Comment	Prestudy needed
Reference	
Customer	All
Tool	SDT SDL Editor
Level	Slogan
Area	Ergonomy
Submitter	x
Priority	3: next release (3.3)
Keywords	diagrams files multiple
Status	Classified

is taken with the information.

As shown in Table 1, 130 of the 1,920 requirements were either duplicates, merges or splits. In the analysis, only those that are 'true' duplicates are considered, since we know beforehand that merges and splits will match partially and thus bias the result. When these were removed, 101 requirements remained. The resulting set is shown in column 3 of Table 1 (set A_{full}).

Some of the 101 duplicates involved more than one requirement. This means that a requirement may be denoted a duplicate of two other requirements. To resolve this we parsed every identified duplicate and constructed a set of unique duplicate pairs. However, doing this creates a set of duplicate pairs that may be related (which addresses the discussion about clusters at the end of Section 3.3). Therefore, we calculated all these relations and created new duplicate pairs to denote the relation. For example, if requirement A initially was denoted a duplicate of requirements B and C, and requirement D was denoted a duplicate of requirement C, we would first create the duplicates pairs (A, B), (A, C) and (D, C). Then we would add the pairs (B, C), (A, D) and (B, D) to fully reflect all possible relations. This is acceptable since the duplicate relation is transitive. That is, if both A and D are duplicates of C, then A would also be a duplicate of D.

According to the requirements database manager, not all the requirements having status New or Assigned had been analyzed for duplicates, and it was only certain that those having priority 1 had been analyzed. Therefore, we considered removing all requirements with status 'New' or 'Assigned', not having priority 1. After doing this we noticed that some duplicate pairs referred to the removed requirements. Thus, we decided to analyst two sets: one with all requirements and one with the 'New' and 'Assigned' requirements with priority not equal to 1 removed. As the second set does not include all the requirements addressed in the duplicate pairs, those pairs were removed from the duplicates pair set. The resulting number of requirements and duplicate pairs are shown in column 4 in Table 1 (set $A_{reduced}$).

The textual information used to represent each requirement was collected from the 'Summary' field, which corresponds to a short requirement title, and the 'Description' field, which corresponds to a further explanation (see the examples in Figure 4). As these fields were empty for a subset of the requirements, three different requirement sets were prepared from each of sets A_{full} and $A_{reduced}$. The first set comprised all the requirements that had a non-empty 'Summary' field. The second set comprised all the requirements that had a non-empty 'Description' field. The third set comprised all the require-

Table 2: Final sets prepared for the analysis.

	B_{full}		$B_{reduced}$	
Non-empty field	Requirements	Duplicate pairs	Requirements	Duplicate pairs
Summary	1,830	142	1,085	124
Description	1,570	99	915	86
Summary or Description	1,887	142	1,088	124

ments that had a non-empty 'Summary' field or a non-empty 'Description' field (NB. Not exclusive or. Requirements having a non-empty 'Summary' field and a non-empty 'Description' field were included in the last set). In the analysis of the sets using both fields, the two fields were treated as one. Table 2 shows the number of requirements in each of the sets after the requirements with the empty fields had been removed.

4.2 Results

The similarity calculator was run once for each of the prepared requirements sets to calculate the three similarity coefficients described in Section 3.1. The quality was assessed by producing contingency tables for nine different threshold levels as explained in Section 3.3. The threshold levels ranged from 0 to 1 with a 0.125 interval. All the possible combinations resulted in 162 contingency tables (3 measurements · 2 sets · 3 fields · 9 thresholds = 162 tables).

In Table 3, nine contingency tables are shown for the analysis on the 'Summary' field of set B_{full} using the cosine similarity measure. The number of possible unique pair-wise comparisons, which is the same as the total number of possible unique requirement pairs, is denoted $A + B + C + D$ in the contingency table in Figure 2, and corresponds to the sum of each column in Table 3. The first row shows the number of correctly identified duplicate pairs and decreases as the threshold increases. Most requirement pairs are, correctly, considered as non-duplicate as shown in the second row. Their number increases with the threshold level. The third row shows how many duplicate pairs the analyzer identified that actually were not identified as duplicate pairs by the experts. Finally, in the fourth row are all the actual duplicate pairs that the analyzer did not find.

The number of false positives and negatives at threshold level 1 may raise some questions. There may be false negatives because requirements concerning exactly the same issue may be worded differently. The reasons that there

Table 3: Contingency table data for the summary field in set B_{full} using the cosine similarity measurement.

	0+	0.125	0.25	0.375
True positives (D)	114	114	105	80
True negatives (A)	1,578,213	1,578,581	1,628,049	1,666,093
False positives (B)	95,180	94,849	46,555	8,111
False negatives (C)	28	28	35	61

	0.5	0.625	0.75	0.875	1
True positives (D)	62	47	42	31	30
True negatives (A)	1,670,881	1,672,945	1,673,247	1,673,341	1,673,349
False positives (B)	2,864	499	146	52	44
False negatives (C)	80	93	100	111	112

may be false positives are several:

1. A requirement may be partially implemented and result in new requirements. The implemented requirement and the new requirements may then have the same information in some textual attributes. Since none of these requirements are marked as duplicates in the database the automatic analyzer may produce a false positive.

2. The compared textual attributes may be wrong and misleading, not reflecting the actual meaning of the requirement.

3. Two requirements may be highly related and concern the same issue and have the same information in one textual attribute. Nevertheless, they do not have to be duplicates.

4. If all non-matching words in two requirements happen to be stop words, and thus eliminated before the similarity calculation, the reduced requirements may give a similarity measure of 1 but actually have different wordings.

The rate of true positives, the rate of false positives and the accuracy (see Section 3.3) were plotted to compare the measurements and to see which would generate the best result. In Figures 5 through 8, four graphs are shown to support the conclusions on:

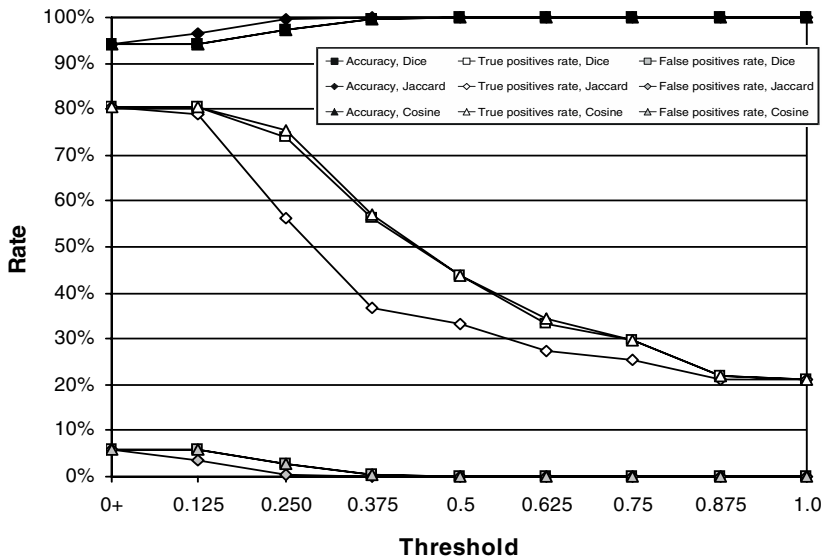


Figure 5: Similarity analysis performance using the summary field in set B_{full}.

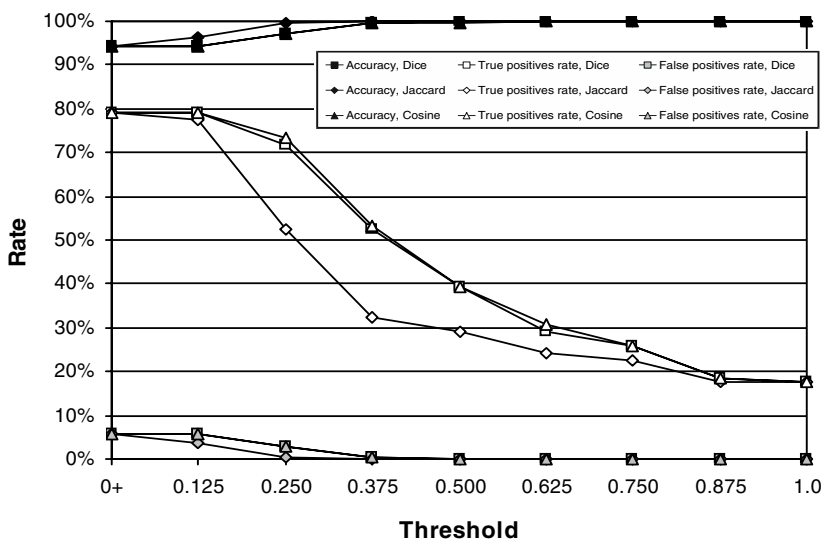


Figure 6: Similarity analysis performance using the summary field in set $B_{reduced}$.

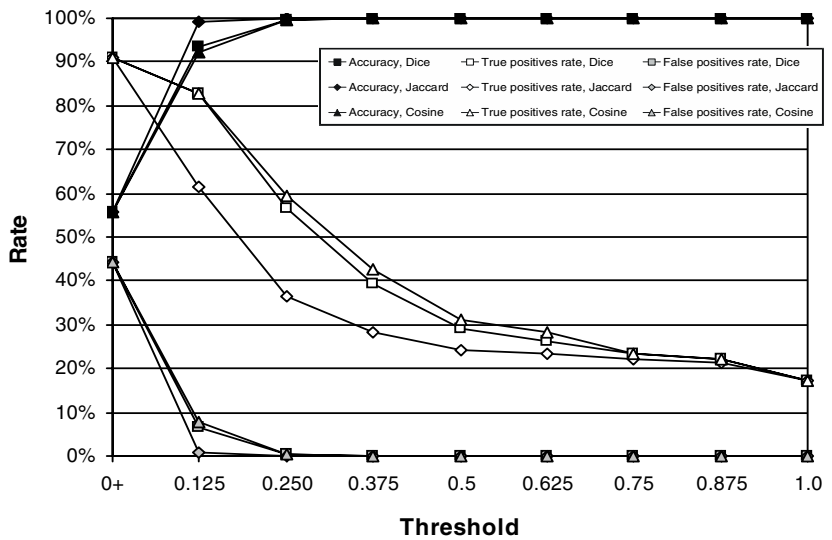


Figure 7: Similarity analysis performance using the description field in set B_{full}.

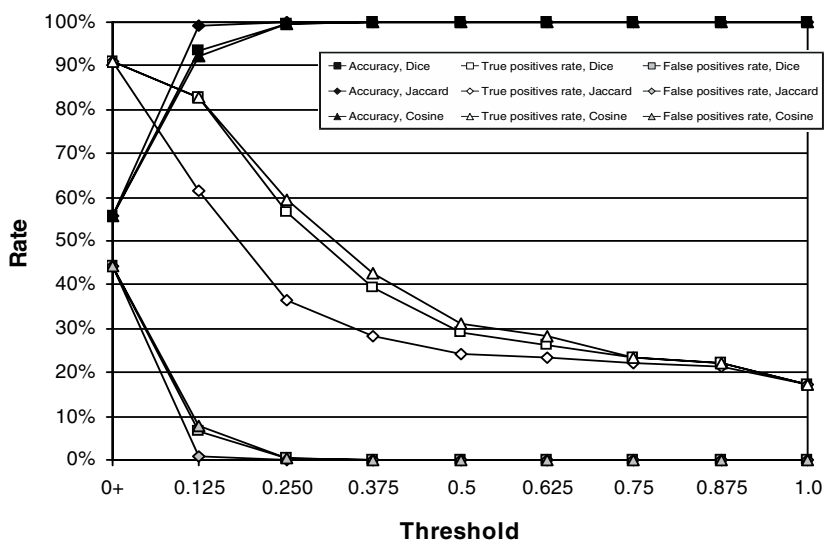


Figure 8: Similarity analysis performance using the summary and description fields in set $B_{reduced}$.

- which measurements may be considered the best
- whether or not the requirements with status 'New' or 'Assigned' and not priority 1 should be ignored
- which fields or combination of fields give the best results.

The graphs show that the rate of correctly identified duplicate pairs (the true positives rate) decreases from 80% or 90% at threshold level 0+ to about 20% at threshold level 1. The lowest degree of similarity is found when there is only one single word matching. Each measure will then give a similarity value just above 0 and thus, using threshold level 0+, suggest exactly the same set of duplicate pairs (every similarity measure but zero between two requirements results in a suggested duplicate pair). Correspondingly, the highest degree of similarity is found when all words match. Each measure will then give a similarity measure of 1 and produce exactly the same set of duplicate pairs. Between these threshold levels the curves differ slightly, which shows that the similarity measures perform differently. The Dice and cosine similarity coefficients show no significant difference, but the Jaccard coefficient performs slightly worse. Thus, for this particular set of requirements, the Dice or cosine coefficient is preferable.

The false positive rate is very low, decreasing from 5.69% down to 0.01%. The accuracy of the similarity analyzer is as high as 94.3% at the lowest threshold level and increases to near 100% at threshold level 1. This curve suggests that the Jaccard coefficient is a better choice, contradicting the choice based on the positives rate.

Looking at Figure 5 and Figure 6, which show the results from using only the 'Summary' field, we can see that there is no considerable difference between the results for set B_{full} and $B_{reduced}$. This implies that either (1) there are 'New' and 'Assigned' requirements with lower priorities that have been analyzed and found to be duplicates, of which some are identified by the program, or (2) the requirements have not been analyzed and few matches were found by the program. Alternative 1 seems more plausible and is also confirmed by the contingency table - more duplicates are identified which must be related to the 'New' and 'Assigned' requirements with lower priorities.

Figure 5 and Figure 7, showing the results from using the 'Summary' or the 'Description' fields respectively (from set B_{full}), differ on the low and high threshold levels. At threshold level 0+, the true positives rates is as high as above 90% using a combination of the 'Summary' and the 'Description' fields. However, the false positives rate is substantially higher and the true negatives

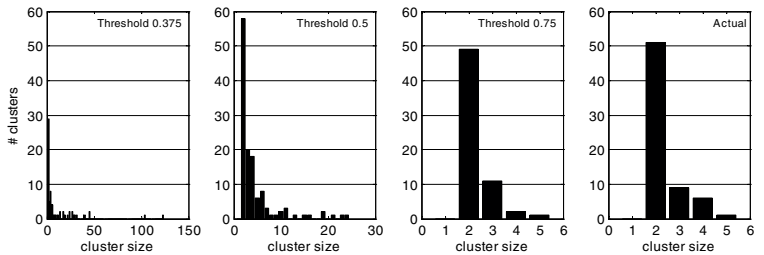


Figure 9: Requirements cluster distribution for the $B_{reduced}$ set using the cosine measure on the 'Summary' field. The three leftmost graphs show the number of clusters of different sizes for various thresholds compared to the actual cluster distribution on the right.

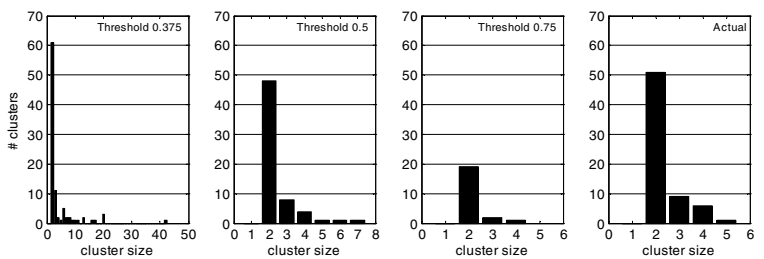


Figure 10: Requirements cluster distribution for the $B_{reduced}$ set using the cosine measure on the 'Summary' and the 'Description' fields. The three leftmost graphs show the number of clusters of different sizes for various thresholds compared to the actual cluster distribution on the right.

rate has also dropped significantly. The conclusion from this comparison is that using only the 'Summary' field gives more accurate answers. The reason for this is that the 'Description' field contains too much noise that incorrectly boosts the similarity measures.

Finally, Figures 5, 7, and 8 support the rather evident: a combination of the 'Summary' and the 'Description' field results in a combination of the results from using the 'Summary' and the 'Description' fields separately.

The high number of requirement pairs identified at threshold level 0+ in Table 3 may at first seem very discouraging. However, calculating the cluster distribution of all the positives (true and false) as explained in Section 3.3 gives support to the following conclusions and the usefulness of the result.

Table 4: Result of expert analysis of the false positives for the set B_{full} at the threshold 0.75 using the cosine measure on the summary field.

Relationship	Count
Duplicate	28
Similar	13
Related	8
Part of	5
Not related	21

The cluster distributions for the $B_{reduced}$ set are shown in Figure 9 and Figure 10. Each figure shows four graphs. The first three show the cluster distribution using the cosine measure on the 'Summary' and the 'Summary' + 'Description' fields respectively. The last graph in each row shows the cluster distribution for the actual duplicates found by the experts.

The graphs show that with increasing threshold the number of clusters of larger size decreases. For example, in Figure9 at threshold level 0.375 there is one very large cluster involving 123 different requirements.

What is noteworthy about this is that the presented study is made on a very large set of requirements but that in reality the requirements arrive continuously, a few at a time. The similarity analysis can thus be made incrementally on a smaller set of requirements, avoiding the need for interpreting the results of similarity analysis of the entire set of requirements at one time. The cluster distribution shows that if we analyst one randomly selected requirement from the database (which may represent a newly submitted requirement), the worst case would be that the analyzer suggests a cluster of 123 requirements to be identical (Figure 6a, leftmost graph). This is thus the maximum number of requirements the requirement analyst must handle simultaneously. As the number may seem too high for the lower thresholds, it is reasonable to suggest that too large clusters may be ignored, as they are probably irrelevant.

Considering both performance and cluster distribution, we may also conclude that the Dice and cosine measures are superior. The true positives rate has already been shown to be higher, and the higher false positives rate is compensated by the suggestion of analyzing a group of related requirements simultaneously, instead of checking each of the several thousand possible duplicate pairs.

Another interesting issue is whether the automated analyzer reveals duplicate pairs that the experts missed. To explore this we let an expert analyze the 75 false positives suggested when using the cosine measurement on the 'Summary' field for set B_{full} at threshold level 0.75. Table 4 shows the surprising result from the analysis. It turned out that 37% of the suggested duplicate pairs were actually missed by the experts! For that threshold level, the true positives rate would then increase from 26% (Figure 6) to almost 40%, the already low false positives rate would decrease, and the already high accuracy would increase. The analyst did not regard two requirements in a pair as duplicate or similar if they were to be implemented in different parts of the software. The table also shows the additional relationships identified, which thus imply that only 21 of the 75 pairs identified would be completely wrong.

The manual analysis also indicated that the analyzer might have a problem when there are too few words in the fields. One suggestion would then be to use the 'Description'field only when the 'Summary' field has too few words.

Furthermore, the threshold value can be tuned based on the requirements engineer's consideration of the best trade-off between few false positives and many true positives.

In summary, it may be concluded that:

1. The similarity analysis technique gives reasonably high accuracy considering its simplicity.

2. For incremental analysis of requirements, given that related requirements are grouped into clusters, the Dice and cosine may be considered the superior measures.

3. A large explanatory field tends to give a worse result, as the discrimination between requirements declines. However, if one field has too few words it may be worth using other lengthy fields.

4. The grouping of suggested duplicate requirements into clusters reduces the analysis burden considerably.

5 Further applications

There are numerous conceivable applications of automated similarity analysis beyond identifying duplicates. The following briefly describes some of these application areas, of which we have only evaluated one so far.

5.1 Requirements interdependencies

Requirements interdependencies are important to identify and keep track of for requirements prioritization and release planning purposes, as interdependencies may govern what partitions of a particular set of requirements are allowed from a functional perspective, or eligible from a cost/value perspective. Carlshamre and Regnell (2000) describe a number of salient interdependencies found in a study of empirical data The relationship between similarity and interdependency is evident in the case where we have two requirements R1 and R2, with the exact same 'Summary' field. This would be a true duplicate pair in the previous sense, but it would also represent an OR interdependency, which imply that either one of the requirements could be implemented. The existence of common keywords may indicate other types of interdependencies as well. For example, if there are several requirements that include the word 'sorting', it may be wise to consider implementing these together to save development resources, which would represent an interdependency regarding cost of implementation.

To investigate whether the similarity measurement technique could be used to support the identification of interdependencies in a set of requirements, we applied the same analysis technique as described in Section 3.1 to five different sets of 20 high-priority requirements, previously studied manually by experts (for further information on the results of the manual study, see Carlshamre et al. (2001)). Among the total of 100 requirements, there were in total 155 pair-wise interdependencies manually identified by experts from each of the five organizations.

Results Each set of 20 requirement slogans were relieved of stop words and reduced to stems, before being separately fed to the similarity calculator using the cosine coefficient. The automatic analyzer reported 70 similar pairs on a 0+ threshold (9, 18, 21, 10 and 12 pairs in each set respectively), of which 25 were true positives. Table 5 shows the frequencies of actual dependencies in relation to the similarity measure using the assessment scheme presented in Figure 2.

A chi-square test (Siegel and Castellan, 1988) gives a p-value less than 0.0001, which shows that the similarity measure varies significantly with actual dependencies.

Thus, by checking for lexical similarity, this particular case demonstrates that it is a promising technique to support the interdependency identification process by automatic analysis. Although the accuracy may not suffice

Table 5: Contingency table for dependencies and similarities.

	Similarity=0	Similarity>0	Total
Actual non-dependencies	750	45	795
Actual dependencies	130	25	155
Total	880	70	950

for this technique to be used on its own, automatic lexical analysis may be used in conjunction with other techniques to reduce the effort of identifying interdependencies.

5.2 Requirements gathering

When a stakeholder is proposing a new requirement, it may be valuable to know if a similar requirement has already been implemented and, if so, in what release. If a similar requirement has not been implemented, it may be desirable to know if a similar requirement has been proposed.

5.3 Strategic fit

A company may define key areas that are of specific importance for the requirements work (e.g., usability, decision-making features or invoicing capabilities). When such requirements are proposed, they can be identified by a similarity analysis approach and thus more easily be given the appropriate management attention.

5.4 Defect tracking

Companies with mature software products that have gone through series of releases often have many defects to track and analyst. As new defects are reported, a similarity analysis approach can aid testers to identify if similar defects have been reported earlier.

5.5 Support issues

Some companies allow their customers to get feedback on support issues through their web sites. Similarity analysis approaches can help the customer to enter questions in natural language and more easily analyst the questions and find suitable answers.

6 Further improvements

There are a number of potential improvements that can be made to the presented requirements similarity measurement method, including the following suggestions to be evaluated in further research:

- Process issues such as when similarity analysis should be used, who should perform the analysis and how the analysis is cost-efficient to perform.
- How different ways of representing requirements affect the results. Which representation is best suited for high precision in automatic similarity analysis?
- Different attributes' impact on similarities. Use of other attributes may increase precision.
- Improve method accuracy. Examples include: the use of a domain-specific stop list, a thesaurus with general synonym words, spelling correction prior to the automated similarity analysis and by not discriminating between words with a short editing distance.
- Smart algorithms: some words may be over-represented in the set of false positives. Removing these words may improve the precision. This is an example of where it may be possible to make the algorithm self-adjustable based on human corrections.
- Evaluate linguistic methods that may provide more precise analysis of natural language requirements on a semantic level. This may include the use of ontologies or word nets.
- Ways of visualizing the results from automated similarity analysis and supporting the requirements engineer in the navigation among related requirements.

In order to make these improvements and to make the methods more general it is of course desirable to apply the methods to other requirement sets from industry. Also, it is of great interest to compare different approaches and combinations of approaches. The implementation cost and computational effort needed for statistical methods, linguistic methods and other computational models (such as the LSA approach in Landauer and Dumais (1997)) are of great interest for applications aimed at market-driven organizations.

7 Conclusions

Automated similarity analysis is a promising technique for supporting requirements engineers to identify requirements duplicates and interdependencies. This conclusion is drawn on the basis of empirical studies on industrial requirements. Automated analysis is, in the particular cases of the presented investigations, able to identify as many as 80% of the actual duplicates and still only incorrectly classify about 6% of all the possible requirement pairs.

When using automated similarity analysis for interdependency identification, a significant correlation was found between similarity and interdependency. The results show a correct classification of 16% of the actual interdependencies.

We do not believe that the presented technique can replace human judgement, but our results suggest that automated similarity analysis on a syntactic level using information retrieval techniques may be effective in pinpointing true duplicates and interdependencies. Further studies are needed in order to increase the understanding of the benefits and limits of automated analysis of natural language requirements (Ryan, 1993). It is especially important to conduct further research in real situations, where new requirements are continuously arriving from multiple sources, and where requirements are analyzed incrementally by a requirements engineer with domain expertise. In these investigations it is also of importance to consider the relationship between effort needed to put a method to work in a market-driven company and the efficiency of the method. Conducting real-world studies is a necessary means for valid assessments of the benefits and costs of decision support systems in a market-driven requirements engineering context.

Acknowledgements This work is partly funded by the National Board of Industrial and Technical Development (NUTEK), Sweden, within the REMARKS project (Requirements Engineering for Market-Driven Software Development) grant 1K1P-97-09690. A previous version of this paper was published at the Seventh International Workshop on Requirements Engineering: Foundations for Software Quality (REFSQ'2001). We would like to direct warm thanks to Per Runeson, Martin Höst and Thomas Olsson, all at the Department of Communication Systems, Lund, for their valuable input and enthusiastic suggestions.

References

Ambriola, V. and Gervasi, V. (1997). Processing natural language requirements. In Lowry, M. and Ledru, Y., editors, *Proceedings of ASE-97: The 12th IEEE Conference on Automated Software Engineering*, pages 36–45, Los Alamitos, CA. IEEE CS.

Carlshamre, P. and Regnell, B. (2000). Requirements lifecycle management and release planning in market-driven requirements engineering processes. In Tjoa, A. M., Wagner, R. R., and Al-Zobaidie, A., editors, *Proceedings of the 11th International Workshop on Database and Expert Systems Applications Process*, pages 961–965, Los Alamitos, CA. IEEE CS.

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

Chen, H., Hsu, P., Orwig, R., Hoopes, L., and Nunamaker, J. F. (1994). Automatic concept classification of text from electronic meetings. *Communications of the ACM*, 37(10):56–73.

Cybulski, J. L. and Reed, K. (1998). Computer-assisted analysis and refinement of informal software requirements documents. In *Proceedings of the Fifth Asia-Pacific software engineering conference*, Taipei, Taiwan.

Deifel, B. (1999). A process model for requirements engineering of ccots. In *DEXA Workshop on the Requirements Engineering Process: Innovative Techniques, Models, Tools to Support the RE Process*, pages 316–320, Los Alamitos, CA. IEEE CS.

Frakes, W. B. and Baeza-Yates, R. (1992). *Information Retrieval: Data Structures and Algorithms*. Prentice Hall, Englewood Cliffs, NJ.

Francis, W. N. and Kucera, H. (1982). *Frequency analysis of English usage: lexicon and grammar*. Houghton Mifflin, Boston, MA.

Gervasi, V. (2000). *Environment Support for Requirements Writing and Analysis*. PhD thesis, Dipartimento di Informatica, University of Pisa, Italy.

Höst, M., Regnell, B., Natt och Dag, J., Nedstam, J., and Nyberg, C. (2001). Exploring bottlenecks in market-driven requirements managament processes with discrete event simulation. *The Journal of Systems and Software*, 59:323–332.

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

Landauer, T. K. and Dumais, S. T. (1997). A solution to plato's problem: The latent semantic analysis theory of acquisition, induction and representation of knowledge. *Psychological Review*, 104(2):211–240.

Lubars, M., Potts, C., and Richter, C. (1993). A review of the state of the practice in requirements modeling. In *Proceedings of IEEE International Symposium on Requirements Engineering*, pages 2–14, Los Alamitos, CA. IEEE CS.

Luhn, H. P. (1957). A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of Research and Development*, 1(4):309–317.

Mitra, M., Buckley, C., Singhal, A., and Cardie, C. (1997). An analysis of statistical and syntactic phrases. In *Proceedings of the 5th International Conference "Recherche d'Information Assistee par Ordinateur"*, pages 200–214, Montreal, CA.

Osborne, M. and MacNish, C. K. (1996). Processing natural language software requirements specifications. In *Proceedings of the 2nd international Conference on Requirements Engineering (ICRE'96)*, pages 229–236, Colorado Springs, CO.

Park, S., Kim, H., Ko, Y., and Seo, J. (2000). Implementation of an efficient requirements-analysis supporting system using similaity measure techniques. *Information and Software Technology*, 42(6):429–438.

Porter, M. F. (1980). An algorithm for suffix stripping. *Program*, 14(3):130–137. (reprinted in Readings in Information Retrieval, Morgan Kaufmann, 1997).

Potts, C. (1995). Invented requirements and imagined customers: Requirements engineering for off-the-shelf software. In *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, pages 128–130, Los Alamitos, CA. IEEE CS.

Rayson, P., Emmet, L., Garsida, R., and Sawyer, P. (2000). The REVERE project: Experiments with the application of probabilistic NLP to systems engineering. In *Proceedings of the Fifth International Conference on Applications of Natural Language to Information Systems (NLDB 2000)*, Versailles, France.

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

Rolland, C. and Proix, C. (1992). A natural language approach for requirements engineering. In *Proceedings of the Fourth International Conference on Advanced Information Systems Engineering (CAISE'92)*, pages 257–277, Manchester, UK.

Ryan, K. (1993). The role of natural language in requirements engineering. In *Proceedings of the First IEEE International Symposium on Requirements Engineering (RE'93)*, pages 80–82, San Diego, CA.

Salton, G. (1989). *Automatic Text Processing*. Addison-Wesley, Reading, MA.

Sawyer, P., Sommerville, I., and Kotonya, G. (1999). Improving market-driven RE processes. In *Proceedings of International Conference on Product Focused Software Process Improvement (PROFES'99)*, pages 222–236, Oulu, Finland.

Siegel, S. and Castellan, Jr, N. J. (1988). *Nonparametric Statistics for the Behavioral Sciences*. McGraw-Hill, Singapore, 2nd international edition.

Van Rijsbergen, C. J. (1979). *Information Retrieval*. Dept. of Computer Science, University of Glasgow, 2nd edition.

PAPER III

Speeding up Requirements Management in a Product Software Company: Linking Customer Wishes to Product Requirements through Linguistic Engineering

Johan Natt och Dag, Vincenzo Gervasi, Sjaak Brinkkemper, Björn Regnell

Proceedings of the 12th International Requirements Engineering Conference, RE2004, 283–294, Kyoto, Japan, September 2004. IEEE CS.

III

Abstract

The development of large, complex software products aimed for a broad market involves a continuous, massive inflow of customers' wishes (collected from the market) and product requirements (generated inside the developing organization). The interrelationships between these two sources of requirements should be identified and maintained to enable well-founded development decisions. Unfortunately, the manual linkage that is routinely performed today is cumbersome, time-consuming, and error-prone.

This paper presents a pragmatic approach based on linguistic engineering to support the linkage between customer wishes and product requirements. An evaluation with real requirements from industry is presented, showing that such automatic support could make linkage faster. Based on these data, we estimate that considerable time savings are possible. The results, together with the identified enhancement, are promising for improving software quality and saving time in industrial requirements engineering.

1 Introduction

The success of market-driven companies developing software products essentially depends on three parameters: (1) when new releases reach the market, i.e. how well the market window is targeted, (2) what content they have, and (3) the associated cost for development (Novorita and Grube, 1996; Sawyer et al., 1999). Profit and market share in combination with user satisfaction are the driving forces of requirements analysis, prioritization, and selection or rejection for implementation (Keil and Carmel, 1995). To find market opportunities and to keep customers satisfied, new requirements must be elicited continuously, and old requirements must be re-evaluated with respect to evolving market needs. This continuous elicitation puts a high pressure on the organization as requirements arrive from different sources and emerge in different projects (Höst et al., 2000). Requirements management in a product software company bridges the market interaction (existing customers, prospects, analysts) with the product development planning (content, resource planning, timing). We therefore have to distinguish between two major groups of requirements: customer wishes and product requirements. Customer wishes are expressions of the perceived market need. These are naturally subject to frequent change; as the product changes, so does the market need. Customer wishes make up a vital and valuable source of information for decision-making. They also enable better communication with each customer by using the customer's own perspective. Product requirements are the requirements that the developing company believes are worth to pursue, stated from the developing company's perspective. These are also used as a basis for product release planning, as well as for feasibility studies and, if selected for implementation, to start actual software development. Customer wishes and product requirements often emerge independently of one another and for several reasons it is essential to keep them separated. For example, several customers may express their wishes slightly different and without referring to the software or business architecture. However, a product requirement addressing all the differently stated wishes may include additional information that is required for decision-making and development but that should not be communicated back to the customers. Naturally, there are multiple associations between these two groups of requirements, which must be found and maintained. The links between customer wishes and product requirements constitute a conclusive piece of information for requirements prioritization and release planning. As always, resources are limited and only a subset of the product requirements may be implemented in the next release of the product.

The linkage process is cumbersome. Each time a new customer wish arrives, it is a difficult and time-consuming task to find those that may be related to the wide variety of product requirements. In current practice, this task is often accomplished using simple search facilities, with consequences on effort and missing links. A well thought-out hierarchical requirements organization may help (e.g., based on the software architecture), but as the product gets more complex, requirements will not always fit nicely into such a structure. Moreover, evolution of the architecture, the product, and the company focus deteriorates the maintenance of the requirements hierarchies.

In this paper we investigate the possibility of giving automatic support to the manual work of requirements linking. This is carried out using a pragmatic linguistic engineering approach (Garigliano, 1995). Our long-term objective is to engineer an integrated, supporting, and semi-automatic system that deals with natural language requirements and which satisfies the constraints in the particular industrial setting described. The most vital constraints are the cost-benefit of such a system and the varying textual quality of the requirements, to which a system must be less sensitive. We have selected a set of robust techniques, well known within statistical natural language processing (Manning and Schütze, 2002), that we use to calculate similarities between requirements based on word occurrences. These similarities may be used to present, for a selected customer wish, a top list of product requirements that are candidates for linkage. We present an evaluation using real requirements and manually identified links, received from industry. It turns out that the selected techniques may properly support linkage in an industrial setting for up to 50% of the links. Improvements are suggested together with interesting alternatives and supplementary approaches. The paper is structured as follows. In Section 2 the case study environment is presented. Section 3 describes, in more detail, the requirements set used in our study. In Section 4 we further describe the envisioned new supported situation and, in particular, the selected techniques. This is followed in Section 5 by a presentation of our evaluation. A discussion of related work can be found in Section 6 which is followed by a survey of interesting further work in Section 7. Section 8 concludes the paper.

2 Requirements management case study

Baan, now part of SSA Global, develops large complex applications aimed for enterprise resource planning (ERP), customer relationship management

(CRM), supply chain management (SCM), product lifecycle management (PLM), and business intelligence (BI). The applications are designed and developed as a framework with separate functional components for different business functions and business process elements. By the end of year 2000, the framework consisted of 250 modules and 10,000 components, comprising around 4.5 MLOC. Between 1998 and 2002, the third author designed and introduced a new requirements management process due to the high complexity of their development situation:

- the comprehensive and vital domain knowledge,
- the large volume of requirements,
- their distributed development organization,
- and the complex dependencies of requirements.

New requirements management concepts were introduced in order to make decision making more transparent and to enable more controlled requirements change management. To support the new requirements management process, Baan also developed their own requirements management tool called the Baan Requirements Database (BRD). Developed in MS Access, the BRD has successfully been used in a distributed setting (although, due to performance reasons, the system has recently been ported into the BaanERP development platform.). Obviously, the BRD is also used to collect requirements on the BRD itself. The requirements management process is depicted in Figure 1.

Requirements management is part of the overall release development process, which also consists of development management to develop the new releases, and delivery management to control the software component delivery to customers. The newly introduced concepts are as follows:

Market Requirements A customer wish related to current or future markets, defined using the perspective and context of the user. An example is found in Table 2.

Business Requirements A generic customer wish to be covered by Baan solutions described in Baan's perspective and context. An example is found in Table 3.

Release Initiation A formal document that triggers a release project in Baan containing high-level strategic topics for business requirement selection.

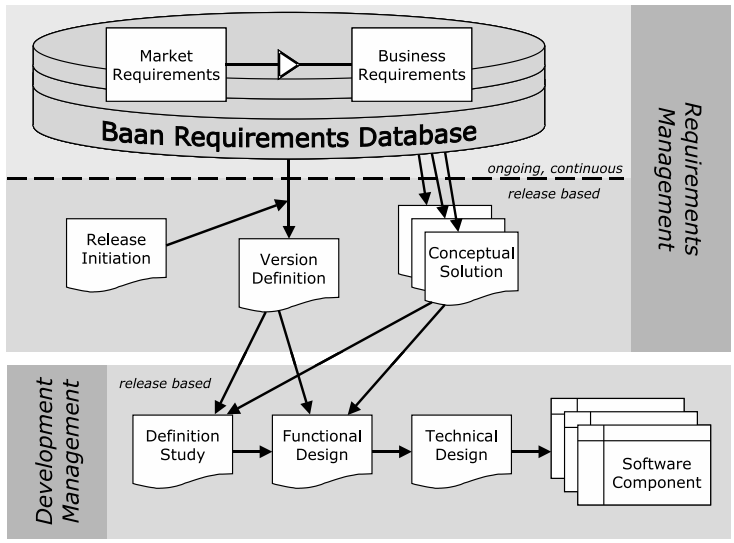


Figure 1: The Baan Requirements Management Process.

Version Definition A document with the listing of business requirements of the new release with the needed personnel resources.

Conceptual Solution A document with a sketch of the business solution for one (preferred) or more business requirements

As shown in Figure 1, the requirements management activities are executed in two modes. Continuously, new Market Requirements (MR) and Business Requirements (BR) are being inserted into the BRD as soon as possible after their receipt or creation, respectively. Only after the company management decides to start a new release project, a Release Initiation document triggers the writing of the corresponding Version Definition (VD) and Conceptual Solutions (CS). Preferably, one CS covers one BR for the sake of simplified (de-)selection of BRs into the new release. The VD and the CS documents are then input for the development processes, which include the writing of design documents (Definition Study, Functional Design, Technical Design) and the coding of the software components.

MRs and BRs that cover the same underlying functional requirement are linked to each other. The relationship between MRs and BRs is essentially of many-to-many cardinality. MRs are copied into the BRD as-is, i.e. without altering the original text as specified by the customer. Maintaining a good relationship with customers is facilitated by providing timely feedback to the

customer on their input for new product functionality. The customer receives an informative message after input review and after completion of the release. Therefore, in case several customers suggest the same functional extensions, then these are each recorded in separate MRs. These MRs are later linked to the same BR.

BRs should reflect a coherent well-defined extension of the product and are created by Product Managers responsible for (a part of) the product. A BR description includes the effort in man-days required for development. Experience with implementing Requirements Management in some product software companies learns us that transparent decision making during release planning requires the BRs to be of a similar workload size (i.e. for Baan between 20 to 80 mandays). Too many small requirements make the list of BRs in the VD too long and cumbersome to manage. Too large requirements do not provide adequate insight in the content of the next release, and hinder effective communication. As customers do not specify their MRs according to these guidelines, it may well be that an MR is very large and therefore linked to many different BRs. Non-coherent MRs dealing with dispersed functional areas are also linked to different BRs.

Linking MRs to BRs and the other way round is of a daily routine for the product managers. Each time a new MR is inserted into the BRD, it is first checked by searching whether there is one or more BRs that already include the specified functionality. This process is very time consuming, as the tool only allows text search in the requirement description. Similarly, when a new BR is created, the corresponding MRs need to be found in the BRD, since the objective is to satisfy as many customers as possible. Finding

Table 1: Number of elicited and linked requirements.

	Business Requirements		**Market Requirements**	
Year	**Elicited**	**Linked**	**Elicited**	**Linked**
1996	0	0	183	113
1997	5	4	683	262
1998	275	169	1,579	388
1999	709	261	2,028	502
2000	669	167	1,270	397
2001	1,000	153	864	224
2002	1,121	340	1,695	514
Total	**3,779**	**1,094**	**8,302**	**2,400**

all MRs that are covered by the BR at hand is virtually impossible, because of the large number of MRs and due to the time-consuming understanding of MR content. Advanced automated assistance to the MR-BR linking can improve the quality of the requirements management process and save costly man-hours of the product managers.

3 Case study requirements data

In this section we provide descriptive statistics on the requirements set used in our study. The total number of business and market requirements elicited at Baan between 1996 and 2002 is found in Table 1. These requirements constitute the basis for the calculations presented in the coming sections. Table 1 also presents the number of requirements that manually have been linked to one another. This information is used to evaluate the outcome of the automatic calculations. Also, the table shows that links between BRs and MRs may cross year boundaries. Figure 2 shows the distribution of how many MRs that are linked to each BR, and vice versa. A one-to-one relationship is obviously the most common. Furthermore, it shows that it is more common to link several MRs to one BR, as opposed to the other way around.

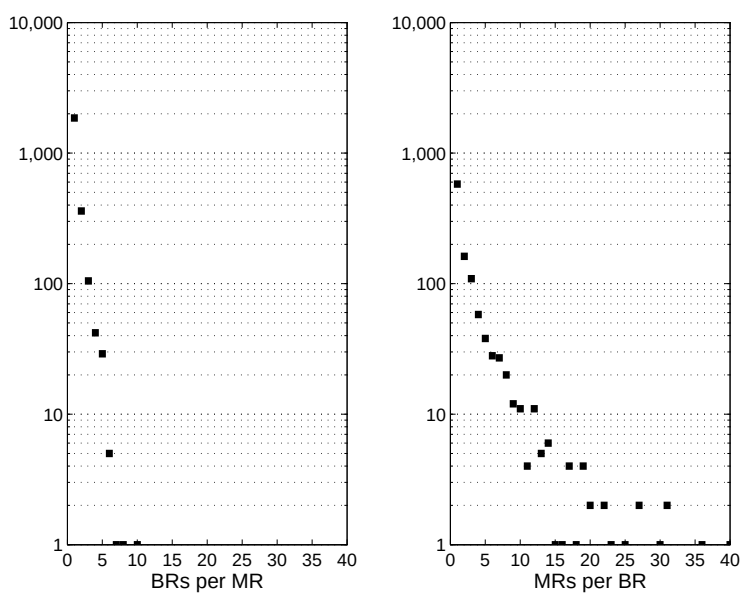


Figure 2: Number of linked requirements.

Table 2: Example market requirement.

Field	**Example** ***[Proprietary information]***
Id	MR10739
Example	*[Request raiser's company]*
Request Person	*[Request raiser]*
Date	1996-05-29
Label	Pricing and Containerization
Description	Specifically what I am interested in is containerization and pricing. For a prospect I am working with (pretty much a distributor of electonic components) I need pricng by type of package by cusotmer type (wholesale or retail). I think pricing by container solves this problem, but I understand to use this feature the item must be a process item and I don't know if this is good or bad. If I must use process what do I gain or lose, like do I have to run a seperate MRP etc. Do I have to have one process company and one non-process company. They have mainly an assembly operation with no process involved. If process would be to cumbersome how difficut a mod would it be to disconnect containerzation from process.
Keywords	Pricing, order planning
Priority	Medium
Type	Functionality
Status	Closed/Completed
User name	*[Requirement submitter]*
Comments	020699: functionality is available in BaanERP in the Pricing module
Agreement	None

Examples of an MR and a linked BR are found in Table 2 and Table 3 respectively (some proprietary information has been left out). Although these two examples can not reveal the full picture, they are representative for the content and form of the two types of requirements. In the label and description fields we find the principal information that constitutes the requirement. The contents in these fields are written in natural language using the corporate language for documentation within Baan (US English). In the current situation, the association between the requirements would be found by, for example, searching for the term *container* in either of the corresponding sets of requirements. The dates of the requirements suggest that it is most likely that the MRs have been searched for possible linkage. Among the MRs we

Table 3: Example business requirement.

Field	**Example** ***[Proprietary information]***
Id	BR10025
Date	1998-01-27
Label	Statistics and containers
Description	1. Container (end item) in statistics Purchase and sales statistics used to be maintained only at main item level. But now it has also become possible to build statistics at container level. There are two aspects: printing statistics in the number of containers for a main item selecting and/or printing statistics at container level 2. Displays in statistics Displays are compositions of end items (for example, an attractive display of different types of cake). The statistics will be updated at both the levels of display item and container (which is part of the display). Prevention of duplicate counting, and correct pricing must be arranged in a procedural manner.
Keywords	Process industries
Type	Usability
Status	Assigned
User name	*[Requirement submitter]*
Comments	Warehousing only

would get 37 hits if searching the label field and 318 hits if searching the description field. Five MRs were currently linked by experts (all five MRs were submitted earlier than the BR). Four links would be found through the label field, but the fifth link would only be found if selecting a new search term (e.g. *statistics*, 40 and 99 hits correspondingly). Based on this and similar cases, we expect the proposed technique to make this search and link procedure more efficient.

The requirements' textual quality varies, e.g. spelling errors (underlined), so in order to determine the requirements lexical and syntactic quality (Fabbrini et al., 1998), we calculated term frequency statistics for different term categories. The term categories we were mainly interested in were correct words, misspellings, and abbreviations. Due to the very large number of terms used (see top row in Table 4), in order to reduce the manual effort needed for the quality assessment, we restricted ourselves to the subset consisting of only terms starting with 'a'. To further speed up the process, we first

Table 4: The requirements' lexical and syntactical quality (term beginning on letter 'a').

			Market Requirements				Business Requirements			
			Distinct		Frequency		Distinct		Frequency	
		Total number of terms:	27,239		659,325		10,431		334,059	
Letter A		**Total letter 'a' terms:**	1,247	(100%)	68,090	(100%)	793	(100%)	36,081	(100%)
		In WordNet 2.0:	662	(53%)	53,125	(78%)	538	(68%)	28,558	(79%)
		Actual words not in WN:	23	(1.8%)	12,675	(19%)	19	(2.4%)	6,679	(19%)
	'Non-words'	**Abbreviations:**	88	(7.1%)	1,009	(1.5%)	36	(4.5%)	584	(1.6%)
		Spelling Errors:	202	(16%)	284	(0.42%)	104	(13%)	124	(0.34%)
		Non-English:	142	(11%)	543	(0.80%)	14	(1.8%)	14	(0.039%)
		References:	67	(5.4%)	272	(0.40%)	62	(7.8%)	78	(0.22%)
		Persons:	36	(2.9%)	128	(0.19%)	4	(0.5%)	12	(0.033%)
		Merged:	22	(1.8%)	47	(0.069%)	15	(1.9%)	29	(0.080%)
		Code:	5	(0.40%)	7	(0.010%)	1	(0.13%)	3	(0.0083%)

used WordNet 2.0 to automatically determine proper English terms. WordNet, developed at Princeton University, is a free lexical reference system in which English nouns, verbs, adjectives and adverbs have been organized into synonyms sets (Fellbaum, 1998). A manual check of the terms that were identified through WordNet was made to see if there were any terms that should be reclassified. We then manually classified the terms that were not found in the WordNet database. The results are shown in Table 4.

The left-most column shows the term categories that we identified. From top to bottom they are:

In WordNet 2.0 The terms that had an appropriate meaning in WordNet, i.e. an actual word, and excluding reclassified terms. For example, the term *au* was found in WordNet as (1) the chemical notation for gold, or (2) the abbreviation for astronomical unit. However, the term was not used in neither of these senses, but as a part of a web domain or as the French preposition. As English was the stipulated language we decided to reclassify the term as a reference.

Actual words not in WN The terms that were not in WordNet, but manually identified as actual words. The numbers are not surprising, as WordNet only comprises four parts of speech and all senses are not represented in WordNet 2.0.

Abbreviations Non-standard abbreviations (e.g., accts for accounts) and other domain-specific abbreviations (e.g., ACP).

Spelling errors Misspelled words.

Non-English Words in another language than the stipulated. In most cases, in particular for the MRs, this is acceptable, as many non-English customers prefer to put an explanation in their own language within parentheses.

References Identifiers, names of companies, and product names (e.g. AOL, AG0000083).

Persons Names of people.

Merged Run-together words (e.g. articlegroup).

Code Pure programming code (e.g. AddShow).

Table 5: Most frequent terms.

BR rank	MR rank	Term	Frequency	$\frac{MRfreq.}{BRfreq.}$
1	1	item	2,117	2.30
2	2	line	1,609	1.83
3	6	process	1,484	1.50
4	15	datum	1,350	1.41
5	9	plan	1,281	1.66
6	136	erp	1,243	0.37
7	4	time	1,242	1.93
8	5	sale	1,137	2.09
9	7	change	1,078	2.03
10	40	warehouse	1,057	1.08
11	14	date	1,003	1.95
12	12	invoice	994	2.01
13	21	base	962	1.59
14	26	requirement	962	1.44
15	3	customer	956	2.72

As shown in the table, the terms in the 'non-word' categories are, fortunately, not used particularly frequently. We will utilize this fact in the next section.

Table 5 shows the 15 most frequent terms used in the BRs and the corresponding rank in an ordered term frequency list for the MRs. The table shows that many of the terms are used with comparable frequency in the BRs and MRs. This is also indicated by the Spearman rank order correlation coefficient r_s (Siegel and Castellan, 1988) (also see (Kilgarriff, 2001) for a discussion on statistics for corpora comparison). Calculated on the intersection of the two frequency lists, we get $r_s \approx 0,78$, significant at the $p < .00003$ level. The intersection constitute 4,660 terms in total. 1,899 terms only occurred in the BR frequency list and 8,234 terms only occurred in the MR frequency list. These unique terms had very low frequency in their corresponding frequency list and, more importantly, essentially comprise non-English words and misspellings. Thus the correlation coefficient do give a good indication of the shared term usage. This gives support for the technical approach chosen, to calculate similarity based on word occurrences, which is described in the next section.

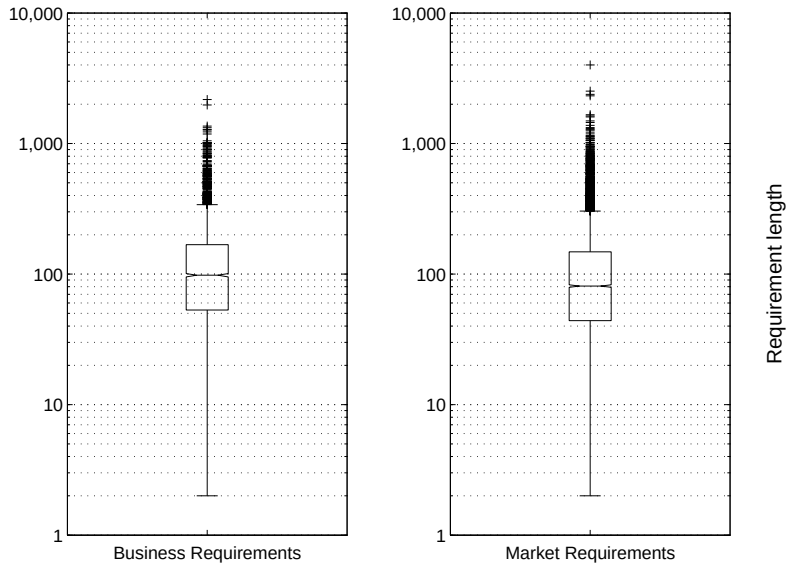


Figure 3: Requirements' length.

The final statistical data that we present reveal the requirements' lengths. Figure 3 depicts the distribution of the requirements' lengths for the BRs and MRs. The BRs are typically somewhat more verbose. In each set there are some requirements that comprise more than 500 words, but the most common requirement length is around 100 words. The reason for the enormous length of some requirements is that they contain complete mail conversations that discuss the requirement scope.

4 Technical approach

The envisioned automated support to the manual work of requirements linkage should be well integrated into the BRD (Section 2), giving relevant suggestions on corresponding requirements when requested. Figure 4 illustrates where this automated support would be adopted into the requirements management process (Figure 1). It is important to understand that nothing is linked automatically. Based on the similarity calculations, *suggestions* are given to a human who may or may not assign the suggested links. Our expectation is that relevant suggestions will be provided faster this way than if a human would have to select several different search terms and, for each of these, search through the database.

The challenge is to suggest requirements that are potential candidates for linkage without the aid of any conceptual models, predesign or requirements models, as none are available at the time of submittal of a new requirement. Modeling the 12,000 requirements, even incrementally, prior to selecting only a small subset for implementation is simply not regarded as cost-beneficial. Approaches using natural language processing techniques in order to model, validate, and help understand requirements are available but are not directly applicable here (see Section 6 for a further discussion). These approaches may present interesting opportunities, but regarding the very large amount of requirements we start off by choosing a more pragmatic angle.

We first define the notion of a link in more technical terms. In Section 2, we defined a link between a customer wish and a product requirement as an indication that they refer to the same software functionality - or, in other words, that they express the same intent. So, two requirements should be linked if they have the same meaning, although expressed in a different style and language. Unfortunately, it is still an unsolved task to functionally represent meaning in a way that can successfully be used by automatic systems (Manning and Schütze, 2002). Thus, we cannot make use of such a system without human intervention. We therefore choose to recast the challenge into suggesting semantic similarity based on term occurrences. We assume that MRs and BRs refer to the same functionality if they use the same terms, i.e. the same terminology. In an RE context this may be a reasonable assump-

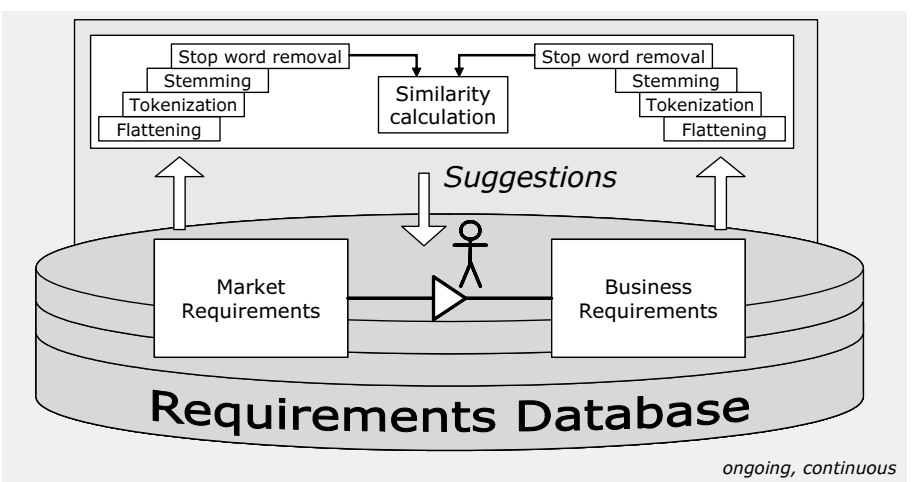


Figure 4: New setting with automated support.

Table 6: Intermediate results from preprocessing part of the example requirement MR10739.

Stage 1: Flattened	Stage 2: Tokenized	Stage 3: Stemmed	Stage 4: Stop words removed
Pricing and Containerization Specifically what I am interested in is containerization and pricing. For a prospect I am working with (pretty much a distributor of electonic components) I need pricng by type of package by cusotmer type (wholesale or retail).	pricing and containerization specifically what i am interested in is containerization and pricing for a prospect i am working with pretty much a distributor of electonic components i need pricng by type of package by cusotmer type wholesale or retail	price and containerization specifically what i be interest in be containerization and price for a prospect i be work with pretty much a distributor of electonic component i need pricng by type of package by cusotmer type wholesale or retail	price containerization specifically containerization price prospect pretty distributor electonic component pricng type package cusotmer type wholesale retail

tion, as in this case the language used tends to be more precise than in literary text, and moreover both customer wishes and product requirements refer to the same domain (that is characterized by the same basic language). This is supported by the term usage (of which an extract is presented in Table 5). Whether this assumption is valid is a matter of empirical validation, which this paper addresses. It should be noted however that this is the same assumption made in many text retrieval systems that have been in widespread use in other fields (e.g., medical literature, legal references, etc.).

4.1 Preprocessing

Before submitting the requirements to an automated process for establishing proper links, a number of preprocessing steps are performed. In detail, the process is as follows:

1. Requirements are first flattened, by merging the label and description fields, and discarding other administrative information (e.g., dates of submission). Thus, the complex data items maintained in the BRD are reduced to plain strings. This choice is based on the results in an

earlier similar study (Natt och Dag et al., 2002), where the recall rate was found to be higher when using both a summary and a description field.

2. Each requirement is then tokenized, by using a custom-made tokenizer based on Flex/Lex rules (Lesk and Schmidt, 1975). In this step, terms are identified, and the original requirement (a string of characters) is transformed into a sequence of tokens. Particular care is taken to identify tokens that represent references to standards and other documentation (e.g., "ISO-8859-1"), as these occur frequently and are particularly meaningful for linking purposes. Other numeric references are left out altogether, as it turns out that they introduce too much unneeded noise in the data.

3. Stemming is then applied to each token to remove affixes and other lexical components not needed for comparison purposes. We use the `morpha` morphological analyzer described in (Minnen et al., 2001) for stemming. For example, after this step both "managed" and "managing" are transformed into "manage", thus simplifying further processing.

4. Common terms that have a purely syntactic role (stop words) are then removed, as they do not provide useful information for establishing correct links. Articles, prepositions, and a few other closed-class words are discarded in this step.

As an example, Table 6 shows how a part of the MR in Table 2 is transformed into a sequence of terms by the preprocessing stages.

4.2 Linking and clustering

Each requirement, transformed accordingly, is then represented using a vector of terms with the respective number of occurrences (the so-called vector space model (Manning and Schütze, 2002)). Table 7 presents an extract of the vector space matrix for the requirements set used in this paper. The matrix shows how many times a term appears in each requirement (notice that such a matrix is usually very sparse, and can be stored and queried efficiently). From the matrix it may also be derived how many terms the requirements have in common, i.e. the overlap. This can be used as an intuitive starting point for the similarity measure.

Table 7: Vector-space matrix extract.

Term	MR10739	BR10025	BR10031
container	1	6	4
containerization	2		1
item	2	5	11
level		4	3
main		2	1
package	1		1
price	3	1	
print		2	
process	7		1
purchase		1	1
sale		1	1
sequence			1
statistics		8	
type	2	1	

In the vector space model, each term can be seen as a dimension in an n-dimensional space. The number of occurrences of each term in a requirement is taken as the position of the requirement along the axis representing the term. Thus, a whole requirement can be represented as a point in the n-dimensional space. Very similar requirements will result in very closely clustered points in this space. Although a simple Euclidean distance measure could be used to identify similar requirements, better results can be obtained by using a measure that considers frequency of terms, rather than count of occurrences (Manning and Schütze, 2002). This is particularly true in our context, since often BRs are much more detailed and longer than the succinct customer wishes. The cosine correlation measure is often chosen in text retrieval applications for this purpose, as it does not depend on the relative size of the input (Manning and Schütze, 2002). In our case, we use the following measure:

$$\sigma(f,g) = \frac{\sum_t w_f(t) \cdot w_g(t)}{\sqrt{\sum_t w_f(t)^2 \cdot \sum_t w_g(t)^2}} \qquad \text{(III.1)}$$

where f and g are two requirements, t ranges over terms, and $w(t)$ denotes the weight of term t. The term weight is typically a function of the term frequency, since while the number of times a word occurs is relevant, its relevance

decreases as the number gets larger (Manning and Schütze, 2002). One common approach is therefore to use a term weight of $1+log_2(term\ frequency)$, which we use in this paper.

Once the similarity measure is defined, suggesting potential links for an incoming requirement is a matter of sorting pre-existing requirements according to their similarity to the new one, and offering the most-similar requirements to the user as candidates for establishing links. Of course, there is no guarantee that two requirements that are similar according to the $\sigma(\cdot)$ measure are indeed related: we assess how effective this technique is in industrial context in the next section.

At this point it is also worth noting at least two challenges, raised by the matrix extract, that the current preprocessing steps fail to handle. The stemming rules do not reduce the verb *containerization* and the noun *container* to the same stem. From a semantic point of view this is perfectly correct, but as the two terms concern the same domain concept their association should be utilized to increase the similarity measure. The current realization of the vector-space model will not make that possible.

The other potential problem has to do with synonyms (e.g. the term *purchase* would perhaps preferably be related to the term *buy*). Although synonyms may be relevant to address it is not certain that it will improve the measure considerably. I connection to synonyms it could be more promising to also take hypernyms and hyponyms into consideration (Jackson and Moulinier, 2002). Nevertheless, this is a matter for further research (see Section 7).

5 Evaluation

5.1 Evaluation technique

In order to evaluate how well the approach performs when it comes to identify correct links, we use the manually identified links as the "presumably correct" answer. Our goal is to find out how many of these links the automatic system can retrieve. Retrieval results of this kind are traditionally evaluated by recall, precision, fallout, accuracy and error (Manning and Schütze, 2002). How these measures are to be calculated and interpreted is dependent on the application. Accuracy and error are often not very interesting as the number of correctly left out items (true negatives) usually is huge, which will give a high accuracy.

For the industrial setting described in Section 2 it is of interest to be pre-

Table 8: Top list example

MR10013		
Pos	**Req.**	**Similarity**
1	BR10012	0.45
2	**BR10156**	0.43
3	BR10006	0.42
4	BR10536	0.38
5	**BR10987**	0.36
6	BR10273	0.36
7	BR10740	0.34
8	BR10419	0.33
9	**BR10622**	0.24
10	BR10082	0.21
11	BR10283	0.18
...	...	...

sented, for a particular requirement of one type, with a list of candidate requirements of the other type. A top list is thus constructed by sorting the requirements by similarity. The size of the top list will thereby represent our similarity threshold.

A top list size of 1 is not necessary, nor wanted. A top list size of 7 ± 2 could be a good compromise (Miller, 1956). It enables us to quickly spot one or more correctly related requirements, while taking into account that we are not able to reach 100% recall or precision (proper experimentation with presenting the resulting top list to the Product Managers is still required). In Table 8 the situation is illustrated, where an example extract of a top list for one MR is shown. In the table we have shaded those requirements grey that would fall outside the top list and which are typically not part of the shown result. The BRs in the top list that are correctly related to the MR are highlighted.

In this situation it is not critical that a correct suggestion is presented at position 1 but, of course, the higher the position the better. We could then use the ranked recall measure (Jackson and Moulinier, 2002), but as we would like to relate the recall to a threshold (i.e. the top list size) we choose to compute recall for different top list sizes. Recall is the proportion of the target items that a system gets right (e.g. a system retrieving 100 of the 1,000 known relevant items has a recall of 10%) and we use the following adapted

procedure:

1. Calculate the complete similarity matrix. The similarities are computed as described in Section 4.
2. For each requirement of one type, sort the requirements of the other type on similarity (as shown in the example in Table 8).
3. Calculate the overall recall for a top list of size n as:

$$Recall(n) = \frac{\sum_{i=1...\#req} targeted(n)}{\#actual\ links} \tag{III.2}$$

where

$$\begin{aligned} targeted(n) = &\text{ for requirement } i, \\ &\text{number of correctly identified} \\ &\text{links within a top-}n\text{ list.} \end{aligned} \tag{III.3}$$

We may then plot a recall curve as a function of the top list size.

However, step 3 is further adjusted to take the many-to-many relationships into account. Suppose we have the situation shown in Table 8. In the presented top list of 7 requirements, we find that 2 of them are correct. In an interactive situation, these may be marked for linkage and could then be removed from the top list. When they are removed, the requirements previously at position 8 and 9 may be revealed. In this example we are lucky, and the final third requirement is shown, which is easily spotted. As long as correct suggestions are shown in the top list we can reveal more suggestions without exceeding the selected top list size. Consequently, the recall rate may be slightly greater. This calculation procedure is more appropriate considering the specific industrial setting in which we expect to dynamically set the requirements relationships based on the presented top list.

5.2 Evaluation results

The results from our calculations are found in Figure 5. The figure shows the recall curve for the top lists of suggested BRs for each MR. The solid line represents the recall curve for calculating similarity using $1+log_2(termfrequency)$, and the dashed line for calculating it using just the term frequency. As expected, mitigation improves recall (typically 10%).

As can be seen we never reach 100% recall. This is because there are some links that could not be identified at all, i.e. some linked requirements have

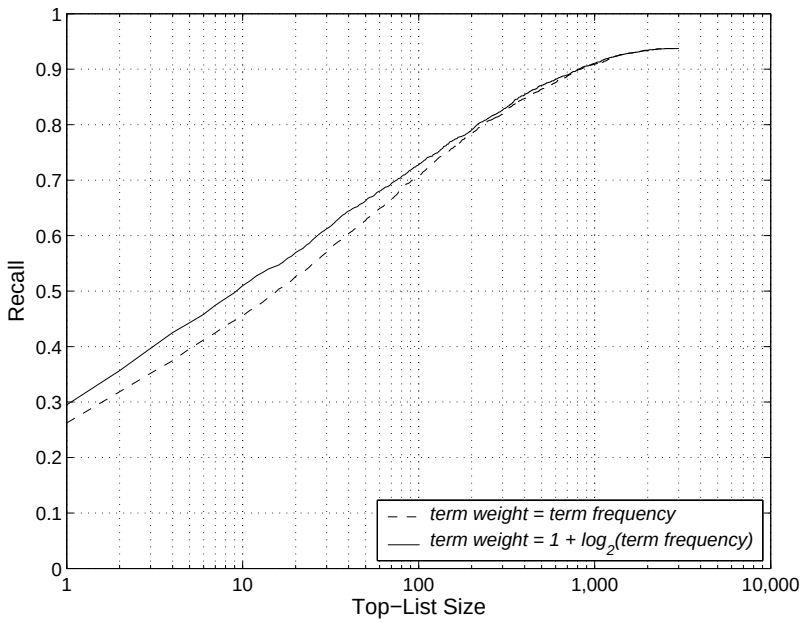


Figure 5: Recall for linking an MR to BRs.

no terms in common. There were 204 links that could not be identified, which result in a maximum recall of $(3,259 - 204)/3,259 \approx 94\%$ (but that would require a top list of 3,000 requirements, which is quite unreasonable). Looking at the requirements comprised by the 204 links that could not be retrieved we found that:

- The links comprised 101 BRs and 158 MRs, thus including many-to-many links.
- The majority of the requirements were sparingly described, consisting of just a single line of text. In some cases there was no description at all. This is not necessarily wrong in the Baan RM process perspective (an empty BR is allowed to be created and directly linked to an MR). These special cases do however affect the results negatively.
- Some requirements were completely written in languages other than English. This should not be allowed without an additional English description.

- Many of the BRs and MRs seemed to describe completely different things. For a better understanding of these abstractions further analysis is required, which is beyond the scope of this paper. It is also of interest to understand how these links were assigned in the first place.

Figure 5 shows that, for a reasonable top list size of 10, we reach a recall of 51%. This is good considering the pragmatic approach taken and the impact on the saving of time that could be made in industry.

In an industrial setting, linkage is performed in both directions, i.e. MRs are searched to find which are related to a particular BR, and vice versa. We therefore calculated recall for the case where we are presented with a top list of suggested MRs for each BR. We can expect a difference, as the relative position of MR M within the top list for BR B, is not equal to the relative position of BR B within the top list for MR M (i.e., their individual positions are based on different similarity sets). We found that for a top list size of 10 we reach a recall of 41%. As we have no data on which direction was used originally to manually identify the links, we can only state that we reach a recall between 41% and 51% (for a top list size of 10). Notice however that we could also imagine a multi-page list of results, akin to the user interface of web search engines. In this case, we can expect the user to access up to two or three more pages of results if no convincing link is found in the first page, or if it is suspected that more requirements can be linked. In this case, in the most favorable setting with up to four pages we reach a recall close to 65%.

To give an indication of the time that could be saved, we make a rough estimate based on the statistics presented and another measure reflecting how many requirements that could be completely linked just by browsing a top-10 list. We found that for 690 of the BRs, the recall rate would be 100% using a top list size of 10, i.e. every related MR for each of the BRs would be found within a top-10 list. The 690 BRs are linked to 1,279 MRs, giving an average of 1.85 MRs per BR, but in order not to exaggerate the gain we assume that, in the manual case, one search term is enough to find all the links for one requirement. Supported by the search hit example in Section 3 we further assume that a search would return approximately 30 hits. Thus, in the manual case the worst case scenario would be to browse 30 requirements. With a top list size of 10, the worst case scenario with automated support would be to browse 10 requirements. Up to 66% effort could consequently be saved. If we assume that it takes about a quarter of a minute (15 seconds) to accept or reject a requirement as a link, we find that the gain is $(690 \cdot 30 \cdot 0.25) - (690 \cdot 10 \cdot 0.25)$ minutes $= 3,450$ minutes $= 57.5$ hours. The critical

reader may protest that in a real setting it is not possible to know the stop criteria, i.e. how to know if a presented top-10 list comprise all the possible links for an arbitrary requirement. While that is true, it is also true in the current situation. The stop criteria will unfortunately always be unknown. The above calculation only gives a comparative evaluation of the effort that may be saved.

Finally, we also found it interesting to answer another question: how effectively can we find at least one correct link, i.e. for each one-to-many relationship how well is it possible to spot at least one of them? Just looking at the best-positioned suggested requirement, we reach a recall rate of 58% for a top list size of 10, which seems promising. This gives a reason to incorporate clustering techniques, i.e. to cluster the requirements of one type and use that additional information when producing the top list.

5.3 Validity

There are four main validity threats that may adversely affect our results:

1. Correctness of the similarity calculations.
2. Completeness of actual links.
3. Degree of link intricacy.
4. Copy-and-paste of requirement content.

To address the first threat we have manually validated all the programs we developed. This was done for a randomly selected subset, for which we manually performed all the required steps and compared that to the automatically calculated results. Some minor bugs were found and corrected and for any changes to the code we regression tested all the programs.

The second threat has currently not been avoided. Working manually, Baan's product managers may have missed some relevant link, which our system has identified. Such a link would be considered incorrect in our evaluation. However, this threat is not problematic, since if the missing links are accounted for we will get higher recall. How much higher is beyond the scope of this paper (see the next section for a discussion on further work).

The third threat involves the difficulty of drawing the correct conclusion based on the kind of links that are among the correctly suggested ones. It may well be the case that the remaining links are much more difficult to find using the proposed techniques or not. Stated differently, the presented results

may not be as promising as we may think. However, we do not claim to reach 100% accuracy and we do not aim at completely replacing the current practice. This threat should nevertheless be investigated further.

The final threat has to do with the fact that some BRs may have been created using the exact same text (or slightly modified) as in a specific MR to which it is then linked. Of course the system should spot these, but the recall curve may show a better result than is reasonable in an industrial setting. It is beyond the scope of this paper to manually analyze all 3,259 requirements links. A quick analysis was made to see if there were any requirements pairs that were assigned a similarity of 1. We conclude that although these were few (45), it is of interest to look specifically at those that have been assigned high similarity measures. It is a matter of further work to address this threat systematically.

6 Related work

A recent study shows that several software development companies, in particular the customer-oriented, use common natural language for specifying requirements in the early phases (Mich et al., 2004). Due to the nature of the requirements management process in many companies, we believe that natural language will be used for several years ahead. This motivates the research efforts made within the field.

The underlying activity for supporting the linkage between BRs and MRs may be classified as *requirements similarity analysis*. The approach taken in this paper is based on the assumption that similar requirements have terms in common. This may certainly be an insufficient assumption for achieving very high recall rates and complementary approaches may offer improvements.

Similarity between textual requirements has not been studied extensively, but linguistic engineering techniques to aid other requirements engineering activities have been proposed that may present opportunities for improvements. In addition to our own previous work (Natt och Dag et al., 2002), there are a few that are related specifically to requirements similarity analysis.

For example, Goldin and Berry have developed a tool to extract abstractions from requirements sets (Goldin and Berry, 1997). The tool finds commonalities between requirements by using a sliding window technique that compares sentences character-by-character. They avoid some of the weaknesses in confidence and precision from using parsers or counting isolated words. The result from the tool by Goldin and Berry is a number of ab-

stractions, selected based on the requirements' common content. Relating to our work, instead of extracting the abstractions, a similarity measure could be calculated based on this overlap.

A sliding window approach is also used by Park et al., this time on a word-by-word basis in order to index sentences (Park et al., 2000). They also use a parser to produce an alternative index. Similarity is then calculated for both sets and aggregated into a final, more accurate similarity measure. However, the requirements set used for the evaluation is small and larger sets may present more noise than is revealed in their evaluation. Nevertheless, their study shows how different techniques may be combined to improve the recall rate and this seems to be the most rewarding approach.

Other research efforts within requirements validation may also be incorporated to improve the similarity measures. This includes the work by Fabbrini et al. (a requirements quality model (Fabbrini et al., 2001)), Cybulski and Reed (unifying the requirements terminology (Cybulski and Reed, 1999)), Rolland and Proix (conceptual modelling (Rolland and Proix, 1992)), Fliedl, Kop and Mayr (conceptual predesign (Fliedl et al., 2003)), Osborne and MacNish (restricted language (Osborne and MacNish, 1996)), and Denger et al. (writing more precise requirements (Denger et al., 2001)). However, restrictions emerge from the specific setting described in Section 2. For example, it is not possible to force customers to write their requirements in a controlled language (proposed in (Cybulski and Reed, 1999; Osborne and MacNish, 1996; Denger et al., 2001)). As stated in the introduction and in Section 2, this is not even desirable. Furthermore, the problem addressed in this paper is not the difficulty of validating or understanding the requirements, but rather the challenge to handle the large amount of requirements in the decision phase prior to any software design efforts. Thus, it is too early to do any modelling (as presented in (Rolland and Proix, 1992; Fliedl et al., 2003)). Even if it was found to be valuable to model all the 12,000 requirements, it would most likely require too much interactive manual labor.

In the end, it is a matter of the cost-benefit of the techniques to be used, not only what is virtually possible. The effort of getting support systems up and running and integrated into the requirements engineering process as well as making them perform good enough must be balanced against the benefit they provide.

7 Further work

The results presented in this paper are promising for further work and improvements. There are several issues that may have a positive impact on the recall curve:

- Incorporate and aggregate similarity measures using other available techniques (e.g. sliding window, part of speech tagging, etc.). For example, there are also arguments against the cosine measure (as it assumes Euclidian distance), which makes it interesting to investigate probabilistic measures.

- Reuse the information from already linked requirements. In a real setting, most requirements in the database would be already linked. Firstly, they could be used to get more accurate similarity measures, as more textual information would be available in the calculation. Secondly, they could in some cases be left out from the presented top lists. Thirdly, they could be used as a learning set: for each pair of terms (t, t') we could compute a bonus based on how many times the pair appears in pre-linked requirements (e.g., t in a MR, t' in a BR linked to the MR). When comparing to a new requirement, we could consider equal terms to match with a full score (e.g., 1.0), and different terms to match with their "bonus" (smaller) score. This way, a future occurrence of t would suggest that we should consider requirements containing t' as well.

- Expert validation. It is possible that not all links have been found in the manual work. Thus, it is possible that requirements at a high position in the resulting top lists actually should be linked. Experimentation with and interviewing of product managers about the missing links and the reasons (if any) for their rejection could lead to significant improvements.

- Incorporate semantics to catch more distant similarities. For example, tokenization and stemming could be replaced with a part of speech tagger (e.g. the Brill tagger (Brill, 1992)), compound concepts could be treated as terms, and WordNet or another lexicon could be used to deal with synonyms, hypernyms, and hyponyms.

To enable better matches it is also of interest to incorporate checking mechanism in the requirements submission stage, e.g. check of spelling (as

implemented in Caliber RM) and language use. Such mechanisms would improve the results from the similarity calculations without complicating the technical design. The above issues should be addressed together with analysis of the requirement links that were assigned low similarity measures. That could reveal the nature of natural language requirements and how it would be possible to generically deal with them.

8 Conclusions

In this paper we have presented an approach to speed up requirements linking using Linguistic Engineering techniques. We have shown that for an industrial setting where numerous customer wishes and product requirements are elicited, there is valuable support to be given using already well-known, robust techniques.

We have shown that more than half of the links may be correctly suggested and thus found in an easier way than they are today. An estimation based on the evaluation also shows that for 63% of the linked product requirements, all links would be found within a ranked list of 10 suggested customer wishes and time savings of more than 65% could be made.

We argue that the linkage could be made quicker by pushing a button and select from a list of requirements, rather than choosing and typing different search terms. Even if the case is that only the easy 50% are found, it will still be easier and faster. Further investigations will reveal how simple or advanced links that may be found using further improved techniques.

A significant contribution of our work is that the approach has been evaluated using a large set of real industrial requirements. The varying quality that is always found in authentic requirements is a real challenge to natural language processing tools. Therefore, we believe it is of high importance to make these empirical validations before any further steps are taken.

Further technical improvements are as always possible, making way for important savings of time in industrial Requirements Engineering. We do not expect these savings to be of an order of magnitude, but if effort as indicated in Section 5.2 could be saved, we would call it considerable.

Linguistic Engineering techniques have not yet been fully exploited to support software product development. The challenge is to consider all the criteria to yield acceptance: usability, cost-benefit, flexibility, robustness and efficiency, to mention a few (Garigliano, 1995). The presented results are promising for a step towards well-engineered systems to aid Requirements

Management in companies that rely on communication in natural language.

Acknowledgments. The authors wish to thank Pierre Breuls and Wim van Rijswijk at Baan in Barneveld for kindly providing the requirements database. The Ernhold Lundström Foundation covered travel expenses for trips to Italy and the Netherlands. Per Runeson and Lena Karlsson provided valuable input, for which the authors are very grateful.

References

Brill, E. (1992). A simple rule-based part of speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing*, pages 152–155, Trento, Italy.

Cybulski, J. L. and Reed, K. (1999). Automating requirements refinement with cross-domain requirements classification. In *Proceeding of the fourth Australian Conference on Requirements Engineering (ACRE'99)*, pages 131–145, Macquarie University, Sydney.

Denger, C., Dörr, J., and Kamsties, E. (2001). A survey on approaches for writing precise natural language requirements. Technical report, Fraunhofer Institut Experimentelles Software Engineering (IESE), Kaiserslautern, Germany.

Fabbrini, F., Fusani, M., Gervasi, V., Gnesi, S., and Ruggieri, S. (1998). Achieving quality in natural language requirements. In *Proceedings of the 11th International Software Quality Week (QW'98)*, San Fransisco, CA. Software Research Institute.

Fabbrini, F., Fusani, M., Gnesi, S., and Lami, G. (2001). The linguistic approach to the natural language requirements quality: Benefit of the use of an automatic tool. In *Proceedings of the 26th Annual NASA Goddard Software Engineering Workshop*, pages 97–105, Greenbelt, Maryland. IEEE CS.

Fellbaum, C., editor (1998). *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, MA.

Fliedl, G., Kop, C., and Mayr, H. C. (2003). From scenarios to KCPM dynamic schemas: Aspects of automatic mapping. In *Proceedings of the 8th International Conference on Applications of Natural Language to Information Systems (NLDB 2003)*, pages 91–105, Burg (Spreewald), Germany.

Garigliano, R. (1995). JNLE Editorial. *Natural Language Engineering*, 1(1):1–7.

Goldin, L. and Berry, D. M. (1997). AbstFinder, a prototype natural language text abstraction finder for use in requirements elicitation. *Automated Software Engineering*, 4(4):375–412.

Höst, M., Regnell, B., and Wohlin, C. (2000). Using students as subjects – a comparative study of students and professionals in lead-time impact assessment. *Empirical Software Engineering*, 5(3):201–214.

Jackson, P. and Moulinier, I. (2002). *Natural Language Processing for Online Applications: Text Retreival, Extraction and Categorization*. John Benjamins, Amsterdam, The Netherlands.

Keil, M. and Carmel, E. (1995). Customer-developer links in software development. *Communications of the ACM*, 38(5):33–44.

Kilgarriff, A. (2001). Comparing corpora. *International Journal of Corpus Linguistics*, 6(1):97–133.

Lesk, M. E. and Schmidt, E. (1975). LEX - a lexical analyzer generator. *Computer Science Technical Report*, 39.

Manning, C. D. and Schütze, H. (2002). *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA.

Mich, L., Franch, M., and Novi Inverardi, P. L. (2004). Market research for requirements analysis using linguistic tools. *Requirements Engineering*, 9(1):40–56.

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63:81–97.

Minnen, G., Carroll, J., and Pearce, D. (2001). Applied morphological processing of english. *Natural Language Engineering*, 7(3):207–223.

Natt och Dag, J., Regnell, B., Carlshamre, P., Andersson, M., and Karlsson, J. (2002). A feasibility study of automated natural language requirements analysis in market-driven development. *Requirements Engineering*, 7(1):20–33.

Novorita, R. J. and Grube, G. (1996). Benefits of structured requirements methods for market-based enterprises. In *Proceedings of the Sixth Annual International Symposium on Systems Engineering (INCOSE'96)*, Boston, MA.

Osborne, M. and MacNish, C. K. (1996). Processing natural language software requirements specifications. In *Proceedings of the 2nd international Conference on Requirements Engineering (ICRE'96)*, pages 229–236, Colorado Springs, CO.

Park, S., Kim, H., Ko, Y., and Seo, J. (2000). Implementation of an efficient requirements-analysis supporting system using similaity measure techniques. *Information and Software Technology*, 42(6):429–438.

Rolland, C. and Proix, C. (1992). A natural language approach for requirements engineering. In *Proceedings of the Fourth International Conference on Advanced Information Systems Engineering (CAISE'92)*, pages 257–277, Manchester, UK.

Sawyer, P., Sommerville, I., and Kotonya, G. (1999). Improving market-driven RE processes. In *Proceedings of International Conference on Product Focused Software Process Improvement (PROFES'99)*, pages 222–236, Oulu, Finland.

Siegel, S. and Castellan, Jr, N. J. (1988). *Nonparametric Statistics for the Behavioral Sciences*. McGraw-Hill, Singapore, 2nd international edition.

PAPER IV

A Linguistic Engineering Approach to Large-Scale Requirements Management

Johan Natt och Dag, Vincenzo Gervasi, Sjaak Brinkkemper, Björn Regnell

IEEE Software, 22(1), 32–39, 2005

IV

Abstract

The development of large, complex software products aimed for a broad market involves a continuous, massive inflow of customers' wishes (collected from the market) and product requirements (generated inside the developing organization). The interrelationships between these two sources of requirements should be identified and maintained to enable well-founded development decisions. Unfortunately, the manual linkage that is routinely performed today is cumbersome, time-consuming, and error-prone.

This paper presents a pragmatic approach based on linguistic engineering to support the between customer wishes and product requirements. An evaluation with real requirements from industry is presented, showing that such automatic support could make linkage faster. Based on these data, we estimate that considerable time savings are possible. The results, together with the identified enhancement, are promising for improving software quality and saving time in industrial requirements engineering.

1 Introduction

For large software companies, the sheer number of textual requirements presents specific challenges. To find market opportunities, organizations must continuously elicit new requirements and reevaluate old ones as market needs evolve. Even so, you can implement only a subset of these requirements for the next release. Using linguistic- engineering techniques early on in the requirements management process could make this ongoing process less of a hindrance.

2 Market-driven requirements management

In a product company, requirements management bridges market interaction (existing customers, prospects, and analysts) with product development planning (content, resource planning, and timing). We can therefore distinguish between two major groups of requirements: *customer wishes* and *product requirements*.

Customer wishes are expressions of the perceived market need. These are naturally subject to frequent change – as the product evolves, the market need changes accordingly. Customer wishes make up a vital and valuable information source for decision-making. Also, because the wishes are written using the customer's own perspective, they enable better communication with each customer.

Product requirements are those that the developing company finds worthwhile to pursue (stated from the developing company's perspective). Companies also use these as a basis for product release planning as well as for conducting feasibility studies and, if selected for implementation, starting actual software development.

Customer wishes and product requirements often emerge independently of one another, and, for several reasons, it's essential to keep them separated. For example, customers might express their wishes slightly differently from one another and without referring to the software or business architecture. However, a product requirement addressing all the differently stated wishes might include additional information that's required for decision-making and development but that shouldn't be communicated back to the customers (for example, references to potential technical solutions, either the company's own or from competitive analysis).

Naturally, the two requirements groups share multiple associations. Organizations must find and maintain these links, because they constitute a significant piece of information for requirements prioritization and release planning.

Unfortunately, the linkage process is cumbersome. Each time a new customer wish arrives, the task of determining whether it's related to the wide variety of product requirements is time consuming. Organizations often accomplish this task using simple search facilities, which takes effort and can result in missing links. A hierarchical requirements organization that's well thought out might help (for example, based on the software architecture), but as the product gets more complex, requirements won't always fit nicely into such a structure. Moreover, as the architecture, product, and company focus evolve, the requirements hierarchy deteriorates.

3 A linguistic-engineering approach

Industrial experience shows the need for automated support in the requirements management area (Kaindl et al., 2002; Höst et al., 2001). Modeling several thousand requirements, even incrementally, to be able to efficiently select only a small subset for implementation is simply not financially beneficial. Any automated support must rely purely on the original form of requirements, or unrefined natural language.

Approaches using natural language processing (NLP) techniques to model, validate, and help understand requirements are available but aren't directly applicable here (Natt och Dag and Gervasi, 2005). These approaches present interesting opportunities but can't effectively cope with the large amount of requirements we're considering(Natt och Dag et al., 2002; Park et al., 2000). We have to choose a more pragmatic angle.

A link between a customer wish and a product requirement indicates that they refer to the same software functionality. Two requirements should be linked if they have the same meaning, even if expressed in a different style and vocabulary. Unfortunately, there's still no method for representing meaning in a way that automated systems can use successfully. We therefore choose to recast the challenge into suggesting semantic similarity on the basis of lexical features. We assume that customer wishes and product requirements refer to the same functionality if they use the same terminology. In a requirements-engineering context this is a reasonable assumption, because the language tends to be more precise than in literary text, and, moreover, both customer wishes and product requirements refer to the same domain.

When we submit the requirements to an automated process for establishing proper links, an imagined support system first performs several internal preprocessing steps (see the "Preprocessing" sidebar).

Preprocessing

The system first flattens each requirement by merging the label and description fields and discarding other administrative information (spelling errors are italicized for clarity).

Next, it transforms each requirement into a sequence of tokens after removal of capitals, punctuation, brackets, and so on. This stage is called tokenization.

The system then applies stemming to each token to remove affixes and other lexical components. For example, after this step, both "managed" and "managing" are transformed into "manage", thus simplifying further processing.

Finally, the system then removes common terms that are unlikely to contribute to an appropriate similarity measure (stop words). Articles, prepositions, and a few other words are discarded in this step.

For example, part of the market requirement in Table 1 (see main text) is reduced as shown below:

Stage 1: Flattened	**Stage 2: Tokenized**	**Stage 3: Stemmed**	**Stage 4: Stop words removed**
Pricing and Containerization Specifically what I am interested in is containerization and pricing. For a prospect I am working with (pretty much a distributor of *electonic* components) I need *pricng* by type of package by *cusotmer* type (wholesale or retail).	pricing and containerization specifically what i am interested in is containerization and pricing for a prospect i am working with pretty much a distributor of *electonic* components i need *pricng* by type of package by *cusotmer* type wholesale or retail	price and containerization specifically what i be interest in be containerization and price for a prospect i be work with pretty much a distributor of *electonic* component i need *pricng* by type of package by *cusotmer* type wholesale or retail	price containerization specifically containerization price prospect pretty distributor *electonic* component *pricng* type package *cusotmer* type wholesale retail

THE VECTOR-SPACE MODEL AND THE COSINE MEASURE

The vector-space model is a standard way of representing texts through the words they comprise. Each text is represented as a vector in the high-dimensional space corresponding to the vocabulary used, where each dimension represents a word.

Parts of the market and business requirements in Table 1 would be represented by the word space and corresponding vectors shown in Table A (where the values represent the number of occurrences of each word).

The Cosine measure then takes the two vectors as input and returns a similarity value between 0 and 1, corresponding to the cosine of the angle between the vectors:

$$\sigma(r_m, r_b) = \frac{\vec{r}_m \cdot \vec{r}_b}{|\vec{r}_m||\vec{r}_b|}$$

The $\vec{r}_m \cdot \vec{r}_b$ denotes the dot product of r_m and r_b, which is calculated by multiplying the corresponding frequencies of each word and then adding them together. However, as the number of times a word occurs is relevant, its relevance decreases as the number gets larger. One common approach is therefore to weight the term frequencies using the formula $1 + log_2(term\ frequency)$. Thus, for the business and market requirements in our example, the similarity becomes

$$\sigma(r_m, r_b) = \frac{\sum_i [1 + log_2\ r_m(i)] \cdot [1 + log_2\ r_b(i)]}{\sqrt{\sum_i [1 + log_2\ r_m(i)]^2} \cdot \sqrt{\sum_i [1 + log_2\ r_m(i)]^2}} \approx 0.32$$

	container	*containerization*	*item*	*level*	*main*	*package*	*price*	*print*	*process*	*purchase*	*sale*	*sequence*	*statistics*	*type*
$r_m =$	1	2	2	0	0	1	3	0	7	0	0	0	0	2
$r_b =$	6	0	5	4	2	0	1	2	0	1	1	0	8	1

The system will then internally represent each requirement using a vector of terms, according to the vector-space model (see the "The Vector-Space Model and the Cosine Measure" sidebar). From the vectors, the system can derive how many terms the requirements have in common; we can use this as an intuitive starting point for a similarity measure.

However, better measures exist that consider both the requirements' length and the number of times the shared terms occur. One common measure the literature suggests is the Cosine measure, which calculates the angle between the vectors in the high-dimensional space (see the "The Vector-Space Model and the Cosine Measure" sidebar).

Once we define the similarity measure, suggesting potential links for an incoming requirement is a matter of sorting preexisting requirements according to their similarity to the new one and offering the most similar requirements to the user as candidates for establishing links.

4 Experiment: The Baan requirements set

From 1998 through 2002, Brinkkemper introduced a new requirements management process at Baan (now part of SSA Global). As Figure 1 shows, requirements management is part of the overall release development process, which also consists of *development management* to create the new releases and *delivery management* to control the software component delivery to customers (not shown in the figure).

The concepts this process introduces are

Market requirement (MR) A customer wish related to future products, defined in the customer's perspective and context.

Business requirement (BR) A product requirement to be covered by Baan products, described in Baan's perspective and context.

Release initiation (RI) A formal document that triggers a release project in Baan (containing criteria for selecting BRs)

Version definition (VD) A document listing the new release's BRs with the needed personnel resources.

Conceptual solution (CS) A document explaining the business solution preferably for one BR.

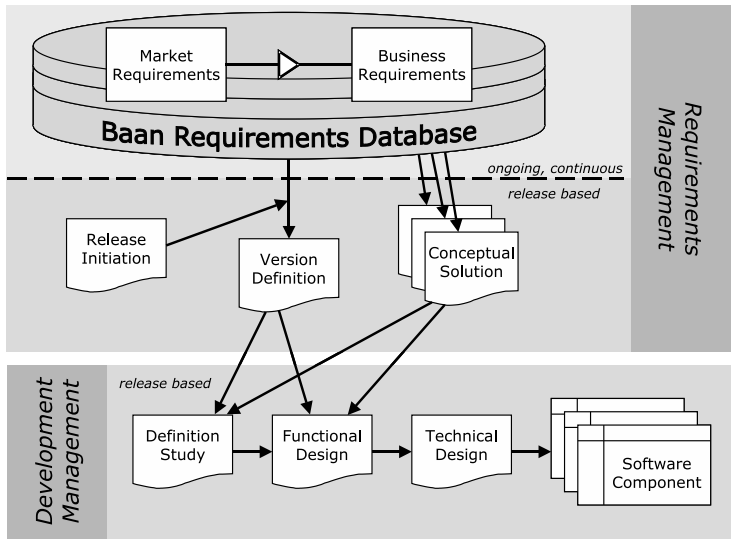


Figure 1: The Baan Requirements Management Process.

Continuously and as soon as possible after their receipt or creation, the new MRs and BRs are inserted into the Baan Requirements Database (BRD). Only after company management decides to start a new release project, an RI document triggers the writing of the corresponding VD and CS. These are then input for the development processes, which include writing design documents and actual coding.

Copying MRs into the BRD occurs without altering the original text as specified by the customer. So, if several customers suggest the same functional extensions, these are each recorded in separate MRs. Providing timely feedback helps maintain a good relationship with the customer, who receives an informative message after input review and also after the release is complete.

Product managers responsible for (a part of) the product create BRs, which should reflect a coherent, well-defined extension of the product. It might be that an MR is very large and therefore linked to many different BRs. A non-coherent MR dealing with dispersed functional areas is also linked to different BRs. Hence, the MR-BR relationship is of many-tomany cardinality, making proper MR-to-BR linking an essential process.

Linking MRs to BRs, and vice versa, is a daily routine for the product managers. Each time a new MR arrives into the BRD, they first check it by searching to find out whether one or more BRs already include the specified functionality. This process is quite time consuming, as current tools allow

only text search in the requirement description.

Similarly, when a new BR is created, the corresponding MRs have to be found in the BRD, since the objective is to satisfy as many customers as possible. Finding all MRs that the BR at hand covers is virtually impossible because of the numerous MRs and the time-consuming process of trying to understand MR content.

For an idea of the high volume and complexity in the requirements management process, consider these statistics. By the end of 2000, the Baan software framework consisted of 250 modules and 10,000 components, comprising about 4.5 millions of lines of code. From 1996 through 2002, Baan elicited 3,800 business and 8,300 market requirements. Each month, 100 new market requirements arrive, of which 20 are handled for the coming release. Over the years, 2,400 market requirements have been linked to 1,100 business requirements.

4.1 Example requirements

Table 1 features representative examples of an MR and a linked BR. In the label and description fields, we find the principal information that constitutes the requirement. The contents in these fields are written in English, Baan's corporate language, and might very well contain spelling errors (italicized for clarity in the examples), acronyms, code snippets, and so on.

Currently, product managers would look for candidate MRs for the BR in Table 1 by searching for specific terms. For example, the term container gives hits in the label field of 37 requirements and in the description field of 318 requirements. In our case, five MRs were linked by experts, of which four were found through the label field. The last link was found by searching for statistics (giving 40 label and 99 description hits).

4.2 Evaluation results

For a particular requirement of a given type, we're interested in the other type's list of candidate requirements. We therefore construct a top list for each market requirement by sorting the business requirements by similarity.

To evaluate how well the approach performs when it comes to presenting the correct links, we use the product manager's manually identified links as the "presumably correct" answer. We can then calculate the recall rate as a function of the top list size (the ratio between the number of correct links found in the top list and the total number of correct links).

Table 1: Example market and business requirements.

Field	Example
Id	MR10739 *[Market requirement]*
Example	*[Request raiser's company. Proprietary information]*
Request Person	*[Request raiser. Proprietary information]*
Date	1996-05-29
Label	Pricing and Containerization
Description	Specifically what I am interested in is containerization and pricing. For a prospect I am working with (pretty much a distributor of *electonic* components) I need *pricng* by type of package by *cusotmer* type (wholesale or retail). I think pricing by container solves this problem, but I understand to use this feature the item must be a process item and I don't know if this is good or bad. If I must use process what do I gain or lose, like do I have to run a *seperate* MRP etc. Do I have to have one process company and one non-process company. They have mainly an assembly operation with no process involved. If process would be to cumbersome how *difficut* a mod would it be to disconnect *containerzation* from process.
Keywords	Pricing, order planning
Priority	Medium
Type	Functionality
Status	Closed/Completed
User name	*[Requirement submitter. Proprietary information]*
Comments	020699: functionality is available in BaanERP in the Pricing module
Agreement	None

Id	BR10025 *[Business requirement]*
Date	1998-01-27
Label	Statistics and containers
Description	1. Container (end item) in statistics Purchase and sales statistics used to be maintained only at main item level. But now it has also become possible to build statistics at container level. There are two aspects: printing statistics in the number of containers for a main item selecting and/or printing statistics at container level 2. Displays in statistics Displays are compositions of end items (for example, an attractive display of different types of cake). The statistics will be updated at both the levels of display item and container (which is part of the display). Prevention of duplicate counting, and correct pricing must be arranged in a procedural manner.
Keywords	Process industries
Type	Usability
Status	Assigned
User name	*[Requirement submitter. Proprietary information]*
Comments	Warehousing only

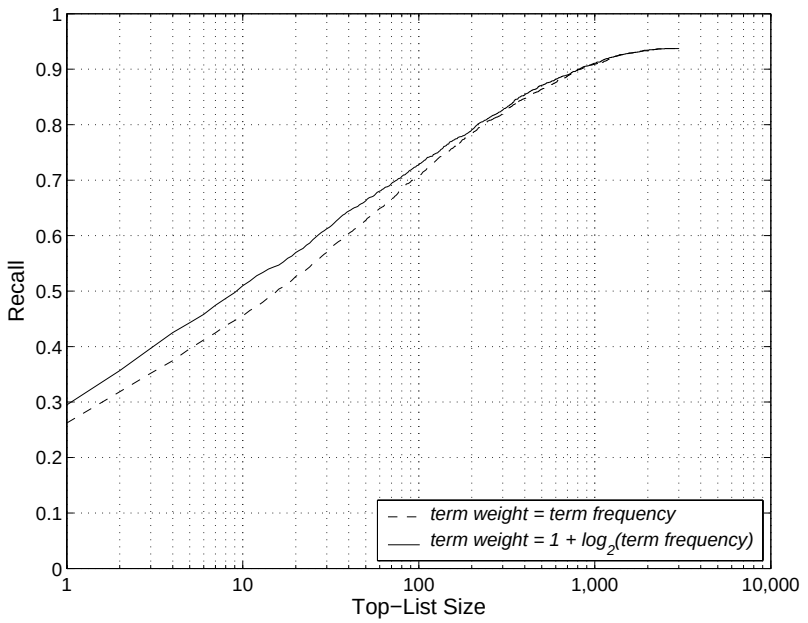


Figure 2: Recall for linking an MR to BRs.

A top list size of 1 isn't necessary, or wanted. For example, a top-10 list lets us quickly spot one or more correctly related requirements, while taking into account that we aren't able to reach 100 percent recall.

The results from our evaluation show the recall curve for the top lists of suggested BRs for each MR (see Figure 2). The solid line represents the recall curve for calculating similarity using $1 + log_2(termfrequency)$, and the dashed line for calculating it using just the term frequency.

As you can see, we never reach 100 percent recall. This is because some links couldn't be identified at all – that is, some related requirements have no terms in common. We couldn't identify 204 links, which resulted in a maximum recall of approximately 94 percent – that is (3,259 - 204)/3,259 – but to reach the maximum recall, we would require a top list of 3,000 requirements, which is quite unreasonable.

For a reasonable top list size of 10, as the figure shows, we reach a recall of 51 percent. This is good considering the pragmatic approach we took and the impact on the time that could be saved in industry.

4.3 Saving time

To indicate the amount of time the process could save, we make a rough estimate on the basis of these statistics and another measure reflecting how many requirements could be completely linked just by browsing a top-10 list. We found that for 690 of the BRs, the recall rate would be 100 percent using a top list size of 10. This means that every related MR for each of the BRs would be found within a top-10 list. The 690 BRs link to 1,279 MRs, giving an average of 1.85 MRs per BR. But to not exaggerate the gain, we assume that, in the manual case, one search term is enough to find all the links for one requirement.

We further assume that a manual search would return approximately 30 hits (based on the previous search examples). Thus, the worstcase scenario would be to browse 30 requirements. With a top list size of 10, the worst-case scenario with automated support would be to browse 10 requirements. Consequently, the process could save up to 66-percent effort.

If we assume that it takes about 30 seconds to accept or reject a requirement as a link, we find that the gain is $(690\dot{3}0\dot{0}.5) - (690\dot{1}0\dot{0}.5)$ minutes, which is 6,900 minutes, or 115 hours.

The critical reader might say that in a real setting it's impossible to know the stop criterion, or how to know if a presented top-10 list comprises all the possible links for an arbitrary requirement. Although that's true, the same applies to the manual case: searching for more keywords could yield more links. Nevertheless, the calculations just given show that a similar coverage level can be reached more efficiently (that is, with less effort) by applying lexical similarity compared to keywords search. If so desired, the time saved can be spent in increasing the level of coverage, by examining more candidates, or devoted to other RE activities if the coverage attained is deemed acceptable.

5 The ReqSimile tool

Considering the need for automated support in the described linkage process and also to demonstrate our approach, we've implemented an open source tool in Java called ReqSimile (see Figure 3; http://reqsimile.sourceforge.net). The tool operates on arbitrary requirements sources, which are accessed through a standard interface (Java Database Connectivity). ReqSimile can therefore integrate well with existing requirements databases (provided a database driver is present). All the involved database tables and fields can be specified from within the tool.

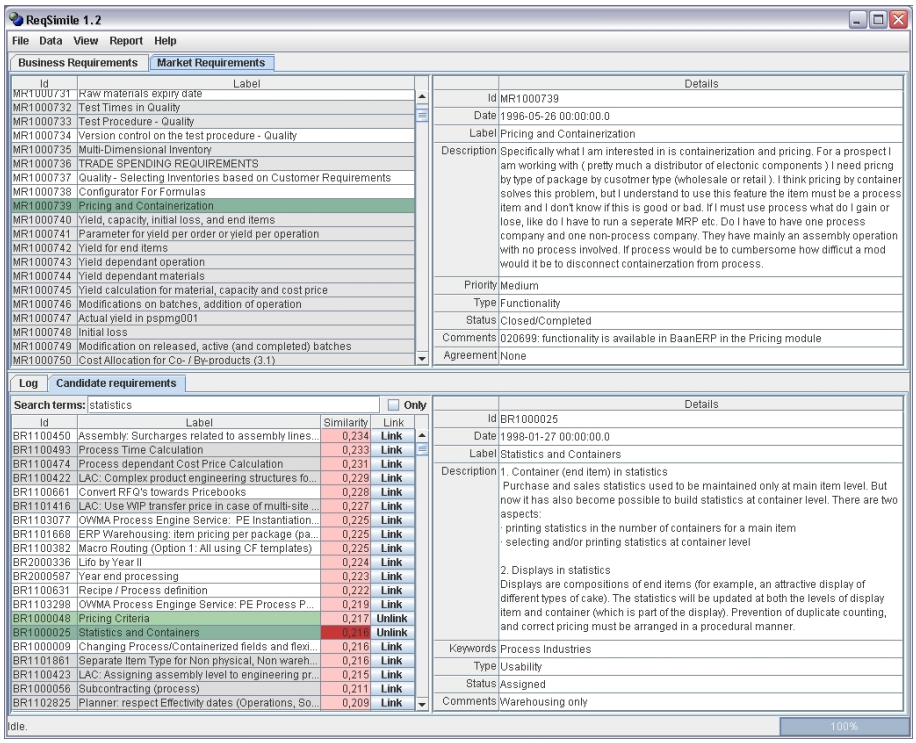


Figure 3: ReqSimile, an open source tool in Java.

The left side in the top pane of the window presents a brief list of requirements. Selecting a requirement makes the requirement's details show on the right. Double-clicking a requirement makes ReqSimile calculate the requirement's similarity to all the requirements in the other set.

A list of candidate requirements, sorted by similarity, then shows in the bottom pane. For flexibility, the user can affect the ranking by adding more search terms in a separate text box.

Already linked requirements are highlighted, and each requirement's details are shown on the right when the requirement is selected. Through the buttons next to each requirement, the user can remove or add links between the selected requirements, and the program will update the requirements database accordingly.

For research purposes, the program also calculates and reports different measures, such as statistics on the requirements sets and recall rates based on the currently linked set.

For practical use, a company can integrate the technique we describe into

its existing requirements management solution, or it can use ReqSimile as an external tool supporting the linking step. In the latter case, the company obtains integration with the existing requirements management practices by configuring ReqSimile to interface with the organization's requirements database.

6 Further work

The ReqSimile tool and the underlying techniques do a good job of supporting the linkage process. Our approach's statistical nature makes it resilient to unintentional errors in the text (for example, spelling errors). It's not worth trying to correct such errors automatically as part of the linking process. In fact, automatic correction is likely to introduce more errors. Furthermore, occasional typos have negligible impact on the recall. However, we've identified other issues that might be interesting to pursue (previous work provides a more elaborate review of potential improvement Natt och Dag et al. (2002, 2004)).

One worthwhile approach would be to incorporate and aggregate several different similarity calculation techniques.6 Different linguistic models might together contribute to better precision.

Another promising approach for improving the precision in future suggestion lists would be to reuse the information in previously linked requirements. A support tool could use the information to improve similarity measures, to leave out already linked requirements, or as a learning set to add relevant terms not originally in the requirement.

A third issue is to incorporate semantics to catch more distant similarities. We expect the handling of synonyms, hypernyms (more general terms, such as vehicle), and hyponyms (more specific terms, such as bike) to provide marginal improvement over the results reported.

Implementing all the extensions we've mentioned would likely bring further improvements over our results, making the approach even more effective.

Software engineers have yet to fully exploit linguistic-engineering techniques to support large-scale software product development (Mich et al., 2004). One reason for this is researchers' limited access to industrial requirements collections. These information sources, together with the requirements activities currently performed in industry, will reveal new opportunities for applicable linguistic-engineering research. The challenge is to consider all the criteria to yield acceptance: usability, cost-benefit, flexibility, robustness, and efficiency, to mention a fewGarigliano (1995). The approach we present is a promising

step toward well-engineered systems to aid large-scale requirements management in companies that rely on communication in natural language.

Acknowledgments. The authors wish to thank Pierre Breuls and Wim van Rijswijk at Baan in Barneveld for kindly providing the requirements database. Thanks to Per Runeson and Lena Karlsson for critical reviews. Thanks to Ernhold Lundström Foundation for covering travel expenses.

References

Garigliano, R. (1995). JNLE Editorial. *Natural Language Engineering*, 1(1):1–7.

Höst, M., Regnell, B., Natt och Dag, J., Nedstam, J., and Nyberg, C. (2001). Exploring bottlenecks in market-driven requirements managament processes with discrete event simulation. *The Journal of Systems and Software*, 59:323–332.

Kaindl, H., Brinkkemper, S., Bubenko jr, J. A., Farbey, B., Greenspan, S. J., Heitmeyer, C. L., Leite, J. C. S. d. P., Mead, N. R., Mylopoulos, J., and Siddiqi, J. (2002). Requirements engineering and technology transfer: Obstacles and incentives. *Requirements Engineering Journal*, 7.

Mich, L., Franch, M., and Novi Inverardi, P. L. (2004). Market research for requirements analysis using linguistic tools. *Requirements Engineering*, 9(1):40–56.

Natt och Dag, J. and Gervasi, V. (2005). Managing large repositories of natural language requirements. In Aurum, A. and Wohlin, C., editors, *Engineering and Managing Software Requirements*. Springer-Verlag.

Natt och Dag, J., Gervasi, V., Brinkkemper, S., and Regnell, B. (2004). Speeding up requirements management in a product software company: Linking customer whishes to product requirements through linguistic engineering. In *Proceedings of the International Requirements Engineering Conference (RE2004)*, pages 283–294, Kyoto, Japan. IEEE CS.

Natt och Dag, J., Regnell, B., Carlshamre, P., Andersson, M., and Karlsson, J. (2002). A feasibility study of automated natural language requirements analysis in market-driven development. *Requirements Engineering*, 7(1):20–33.

Park, S., Kim, H., Ko, Y., and Seo, J. (2000). Implementation of an efficient requirements-analysis supporting system using similaity measure techniques. *Information and Software Technology*, 42(6):429–438.

PAPER V

An Experiment on Linguistic Tool Support for Consolidation of Requirements from Multiple Sources in Market-Driven Product Development

Johan Natt och Dag, Thomas Thelin, Björn Regnell

Submitted to *Empirical Software Engineering*

Abstract

This paper presents an experiment with a linguistic support tool for consolidation of requirements sets. The experiment is designed based on the requirements management process at a large market-driven software development company that develops generic solution to satisfy many different customers. New requirements and requests for information are continuously issued, which must be analyzed and responded to. However, new requirements should first be consolidation with the old to avoid reanalysis of previously elicited requirements and to enable informed decisions on which requirements are the most rewarding to implement.

In the presented experiment, a new open-source tool is evaluated in a laboratory setting. The tool calculates similarities between requirements and presents a ranked list of suggested similar requirements, between which links may be assigned. It is hypothesized that the proposed technique for finding and linking similar requirements makes the consolidation more efficient.

The results show that subjects that are given the support provided by the tool are significantly more efficient and more correct in consolidating two requirements sets, than are subjects that do not get the support. The results suggest that the proposed techniques may give valuable support and save time in an industrial consolidation process.

1 Introduction

Requirements engineering is generally regarded as an important success factor in software development (Hofmann and Lehner, 2001). One of the activities within requirements engineering, requirements management, deals with requirements storage, change management, and traceability issues (Sommerville, 2001). These activities are generally challenging. In market- and technology-driven software development they are exceptionally difficult due to the particular characteristics of the development situation (Lubars et al., 1993; Novorita and Grube, 1996; Potts, 1995; Regnell et al., 1998; Sawyer et al., 1999; Yeh, 1992). Fundamental organizational issues, such as the primary goal, the success measurements, and the product life cycle, present challenges not found in bespoke, or contractual, development. Unfortunately, many industrial requirements management activities are left untouched by the research world and there is an urgent need for a better understanding and improved support (Brinkkemper, 2004).

This paper presents an experiment with a new open-source tool, which incorporates linguistic engineering techniques to support a specific requirements management activity: consolidation of large amounts of requirements that are elicited either from different stakeholders or from the same stakeholder at different points of time. The general purpose of the consolidation process is to find the overlap of two requirement sets with respect to the functionality they convey. The general background to this activity is further explained in Section 2. This is followed in Section 3 by a detailed description of a case in industry, where requirements consolidation is particularly challenging and where much work could be saved if consolidation was better supported.

The experiment was designed to capture the consolidation process in the industrial case. The purpose has been to investigate if the proposed tool and the underlying techniques may give adequate support in that process. The two subjects group were asked to consolidate two requirements sets by finding and linking requirements that address the same underlying functionality. The groups were given two different version of the support tool. One version comprised linguistic engineering techniques aimed at giving automatic assistance in finding similar requirements. The other version offered the possibility to submit search terms for finding similar requirements.

The main results show that the subjects in the group using the tool with automatic assistance performs, in average, significantly better. Not only are they able to consolidate requirements faster, but they are also doing it more correctly - assigning more correct links and missing less.

The paper is structured as follows. In the next section, the general background to the consolidation process is given. This is followed by a problem description captured from our industrial partner. The experimental conception, preparation, planning, and operation are then presented in Sections 4 through 7. The analysis of the experimental results are presented in Section 8. A short discussion can be found in Section 9 followed by Section 10, which summarizes and concludes the paper.

2 Background

Market- and technology-driven software development companies essentially concentrate on three parameters in order to succeed (Novorita and Grube, 1996; Sawyer et al., 1999):

- The point of time when new releases shall reach the market, i.e. how well the market window is targeted.
- The release content, i.e. the quality and functionality of the product release.
- The cost of development, comprising the full life-cycle of the release.

To increase market share and decrease risks, companies concentrate on serving many needs within one market (i.e. *market specialization* (Kotler, 2002)). To fulfill specific needs within the market, customization and customerization has been adopted. *Customization* allows end-users to buy individually differentiated products within certain prescribed aspects (e.g. by individually specify the components to be included in a new computer). *Customerization* goes one step further and involves the interaction with each customer, allowing each customer to request customization on a certain product.

To optimally enable market specialization and product customization in software development, software engineering has embraced the concept of product lines (Clements and Northrop, 2002). A *software product line* is a family of software-intense products that share a common set of features. Each individual product is then developed from a common set of assets in a controlled manner. The objective of a software product line is to optimize software engineering effectiveness and efficiency and to gain improvements in productivity, cost, and quality (Krueger, 2003; Clements and Northrop, 2002).

For software product line development in a market-driven company, the requirements engineering activities differ compared to those for single systems in the following ways (Clements and Northrop, 2002; Potts, 1995):

Requirements elicitation A larger set of stakeholders, such as marketing, support, development, testing, usability evaluations, and technology forecasting. To remain competitive on the market, requirements collection is insufficient and new requirements must also be invented within the organization based on foreseen end-user need. For individual customer satisfaction, possible and anticipated variations points must explicitly be captured.

Requirements analysis A focus on identifying commonalities and variations in order to identify reuse. Close contact with potential customers enables feedback and negotiation of opportunities for saving money by using more common features in favor of unique ones.

Requirements specification Two levels: product-line-wide requirements and product-specific requirements. The product-line-wide specification must indicate where variations points are possible. The product-specific requirements complete or extends the product-wide specification.

Requirements verification Occurs in two stages according to the two levels of specification.

Requirements management Must acknowledge the two levels of specification and the two stages of verification. Impact of changes to the product line must be systematically assessed with respect to all the products in the product line. Links between the core assets and the product line requirements must be identified and maintained.

These activities are conducted continuously and in parallel, making it increasingly important to maintain an effective and efficient requirements engineering process. When a company that has embraced the product line approach is growing, more product variations and customization will be part of the product line. Each individual product has its own life-cycle and so does the whole product line. Thus, new product lines may be started in parallel with already existing, reusing parts of the soon to be phased-out product line.

To effectively manage commonalities and variations points, requirements are stored in a central repository. Changes are made to the requirements in the

repository and project requirements specification are extracted on a needs basis. With the possibility to assign links between requirements to indicate interrelationships, e.g. commonalities, requirements from different customers may be kept separated. This enables better communication with each customer.

New requests arriving from customers and new requirements invented within the organization must therefore be *consolidated* with the requirements already in the repository. Generally speaking, the consolidation process is the process of finding the differences between the requirements in the repository and the newly arrived requirements, in terms of new and changed requirements.

When the product line approach is scaled up, the number of stakeholders increases and the complexity of both the product and development escalates. As a consequence, requirements arrive at a higher pace and in larger volumes, which require more time for the identification of commonalities between stakeholders. Meanwhile, the time-to-market constraint remains.

Eventually, companies face the challenge of dealing with huge information flows that may overwhelm their management and analysis capabilities. Due to the increasing difficulty of identifying and maintaining requirements interrelationships, the consolidation process becomes overloaded, resulting in requirements repository deterioration. The effects can be serious, impeding proper decision-making and customer satisfaction.

3 Industrial problem description

The general problem described in the previous section has been identified through collaboration with our different industrial research partners. One of our partners, Sony Ericsson Mobile Communications AB (SEMC), has found it particular important to resolve their instance of the problem. Our collaboration has comprised investigations into their situation and a preliminary evaluation of the techniques that are used in a proposed support tool (more on the tool is found in Section 5.2). This section provides a description of our industrial partner's development situation and their consolidation problem in the requirements management process.

SEMC develops mobile phones for a global market. As a market- and technology-driven company with many stakeholders, SEMC must handle requirements from a number of sources. As shown in Figure 1, the stakeholders situation presents a more complex flow of requirements than in typical single-systems software projects.

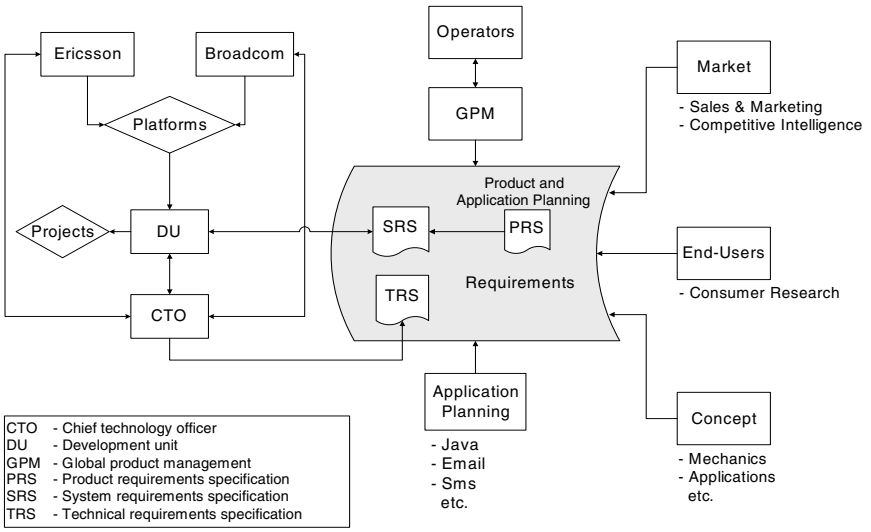


Figure 1: Requirements sources

The arrows in the figure indicate the flow of requirements. In the top left part of the figure, the hardware platform vendors are shown, represented by the two companies Ericsson Mobile Platforms and Broadcom. SEMC's technical requirements on the platform are stored in a Technical Requirements Specification. This document is managed by the Chief Technology Officer, who is also responsible for negotiation with the platform vendors.

In the right part of the figure, inputs from the market, end users, and concept studies are shown. Information from the market is gathered from sales and marketing, and through business intelligence and competitive analysis. Although SEMC does not sell their phones directly to end-users, requirements stemming from their own consumer research programs comprise a valuable source of information for future undertakings. Ideas that may provide competitive advantage are evaluated through concept studies and may generate requirements on both hardware and software.

Application planning, shown at the bottom of the figure, are responsible for the different application areas (e.g. Games, Java, etc.). They act as internal customers who have their own set of requirements on the phone, which must be fulfilled in order for the applications to integrate nicely in the phone.

Finally, at the top of the figure, SEMC's primary customers, the operators, are shown. The operators are responsible for selling the phones to the end user, either directly or through a third party. Through the close contact

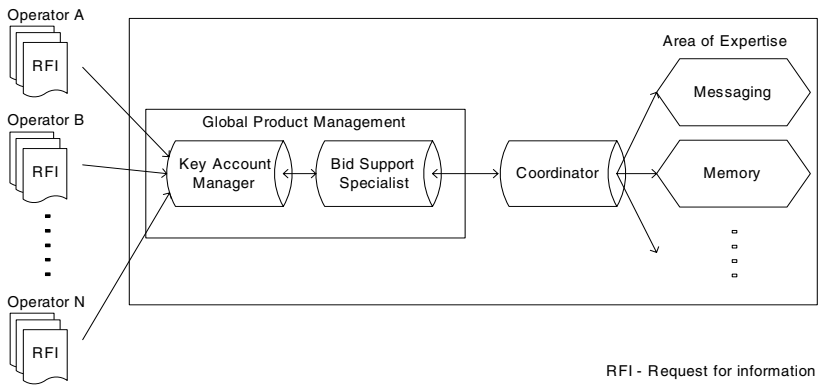


Figure 2: The Request for Information Process at Sony Ericsson

of Global Product Management with the operators, SEMC provides mobile phone customization. In order to optimize software engineering effectiveness and efficiency, this is managed through a product line approach. Requirements are gathered and negotiated through a process that is experiencing severe overload. This process is described in Section 3.1.

A mobile phone project is initiated by a pre-study on the Product Requirements Specification, essentially comprising high-level requirements stemming from the operators. Further input in the initial phase are the concept study reports and consumer research reports. The output from the pre-study is an initial version of the System Requirements Specification, which include both hardware and software requirements. The responsibility for the System Requirements Specification is assigned to a development unit, DU, which are the developers of a project.

3.1 The RFI process

In order for the operators to acquire knowledge in the technical capabilities of SEMC's phones, Requests for Information (RFI) are submitted to SEMC by the operators. Two kinds of RFI's can be identified: general information requests and requests for statement of compliance. Statements of compliance, which are the most common ones, comprise specific requirements and are replied upon using simple standardized statements on whether or not a stated requirement is fulfilled by the product.

The RFI process is depicted in Figure 2. Each year each operator submits a couple of RFIs. The RFIs arrive to the Key Account Managers, one for

each major operator, in different document formats (PDF, Excel, MSWord, etc) and at different times. The main specification technique for the RFI requirements is feature style, i.e. function specification in natural language (Lauesen, 2002).

The Key Account Manager passes the RFI on to a Bid Support Specialist, who reviews the RFI from a market point of view and decides which products shall be considered when dealing with the RFI. The Bid Support Specialist then passes the documents on to the Coordinator, who analyzes the RFI and the accompanying instruction and then distributes relevant parts of the RFI to Areas of Expertise.

An Area of Expertise consists of a Function Group and a Technical Work Group. The Technical Work Group works with road maps (i.e. future functions) and the Function Group works with implementation and testing. When the Areas of Expertise have stated the compliance to each requirement, they send the RFI reply back to the coordinator. The coordinator reviews the answers and sends the replies on to the Bid Support Specialist, who also checks the answers.

If the RFI originates from a major operator, a meeting is held with the Global Product Management, the coordinator and experts from the Areas of Expertise in order to discuss the answers which are to be submitted back to the operator. The RFI reply is then sent back to the operator by the Key Account Manager.

The RFIs play an important role in the operator's strategic planning. The RFIs also provide SEMC with vital business intelligence information as the features prioritized by the operators may be used as a guideline when developing future phones. The operators thus have a great deal of influence on the final requirements for a product and a good relationship with the operators, based on timely and correct replies to the RFIs, is therefore of utmost concern.

The efficiency of the RFI process, in which requirement are analyzed and checked against product features, is however severely impeded. The Areas of Expertise are concerned with their primary assignment in development and testing and have trouble finding the time required to analyze the RFIs. Furthermore, they get particularly frustrated as they have to state the compliance to the same or very similar requirements over and over again. Large parts of the new versions of RFIs arriving from the same operator are typically the same as previous versions. Furthermore, it is often the case that the same and very similar requirements appear in the RFIs from different operators.

Unfortunately, the requirements identifiers for the same requirements in

two consecutive version of an RFI from the same operator may differ and can therefore not be used for simple consolidation. In general, the revision history of the RFIs cannot be relied upon as the operators' requirements processes are not suited or adapted to the activities that SEMC must conduct (unfortunately, there are not enough financial incentives for the operators to make the required changes to their processes).

Consequently, there is much unnecessary redundant work required by the Areas of Expertise and SEMC considers it an important goal to resolve the process bottlenecks. The following sections present the experiment that was designed capture and investigate a potential solution to support the described work situation.

4 Experimental conception

SEMC and many other market-driven companies (Karlsson et al., 2002; Mich et al., 2004) specify their requirements in natural language. Previous research has shown that linguistic engineering may give valuable support in other large-scale requirements management activities, such as duplicate identification and linkage between customer and business requirements (Natt och Dag et al., 2002, 2004). That research, together with the problem description in the preceding section, gave incentives for conducting a controlled experiment for investigating if there is any significant gain from using linguistic engineering techniques to support requirements consolidation.

A laboratory experiment was design to capture the essence of the consolidation problem at SEMC and for evaluating a potential solution that could support the activity. A tool was developed for the purpose of giving support in the consolidation process. The tool takes two requirements sets as input and presents suggestions on similar requirements by calculation the fraction of words the requirements have in common (more information about the tool can be found in the next section).

Figure 3 depicts the conceptual solution of the consolidation activity. To the left in the figure, two requirement sets, A and B, are shown. Suppose that the two sets represent two consecutive submissions of RFIs from the same operator. The earlier RFI, lets assume it is A, would already have been analyzed and the result from the analysis should be available in the central requirements database. The coordinator then uses the support tool to find requirements in B that are already analyzed in A and to mark them by assigning a link between them. The output of the process is shown to the right in the figure.

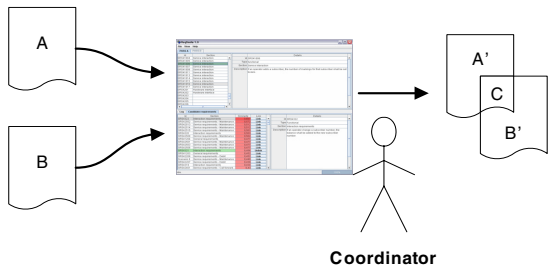


Figure 3: The process of using the support tool for requirements consolidation

The subset A' comprises all requirements that were previously analyzed but are no longer requested by the operator (remind that the overlap can neither be provided by the operator nor easily be found through requirements identifiers. See Section 3.1). The subset B' represents all new requirements that have not previously been analyzed. Finally, there is the subset C, which comprises all requirements in the new RFI that previously have been analyzed (these are the interlinked requirements). The coordinator would then send the requirements in set B' to the Areas of Expertise for analysis. The Areas of Expertise are thus relieved from the burden of re-analyzing the requirements in subset C.

5 Experimental preparation

To minimize costs and spare resources at SEMC, the first level of experimentation was chosen, i.e., experimentation in a laboratory environment (Juristo and Moreno, 2001). Positive results from a first experiment may justify further experimentation in real projects. The rest of this section describes the preparation needed to conduct the experiment and the subjects acting in the experiment.

5.1 Subjects

The experiment was conducted in an academic environment and the experiment subjects were students taking a course in requirements engineering[1]. The course is given in the last year of a 3-year bachelor-level education pro-

[1]The course code is ETS671 and course information is available at http://serg.telecom.lth.se/education/

gram in software engineering. The students are thus soon to become professional practitioners in software engineering. The course gives 5 credit points corresponding to a quarter of a semester and the duration of the course is 8 weeks. The course includes the following parts:

- lectures based on the textbook by Lauesen (2002),
- 6 open-ended home exercises, each handed in by a group of students followed by classroom discussions session regarding the assessment and alternative solutions,
- a practical project in cooperation with local industry involving real RE problems with real stakeholders,
- and two hands-on lab session; one on prioritization using a commercially available RE tool and, this time, one on the requirements consolidation problem described in this paper.

The experiment operation and data collection was planned as a part of the lab session, which was scheduled in the 4th week of the course. Before the data collection, the students had been taught requirements engineering terminology and have gained experience from the practical project. This is believed to be enough knowledge and experience in order to understand and carry out the experiment tasks.

The student population were between 21 and 30 years old with an average age of 23 years and there were 1 female and 22 male students. Further characteristics of the population, based on a pre-test, is given in Section 6.3.

5.2 Tools

Two adapted versions of the ReqSimile tool (Natt och Dag et al., 2005), developed by the first author, were used in the experiment. The user interfaces of the two versions of the tool, ReqSimileA and ReqSimileM, are shown in Figure 4 and Figure 5 respectively.

The left side in the top pane of the window presents a brief list of requirements. Selecting a requirement (1) makes the requirement's details show on the right (2) and a list of similar requirements in the other set appear in the bottom pane, sorted on the similarity value (3). The similarity value is calculated based on the words that the requirements have in common (preprocessed to remove inflections, etc.). The calculation procedure is described in more detail in Natt och Dag et al. (2004).

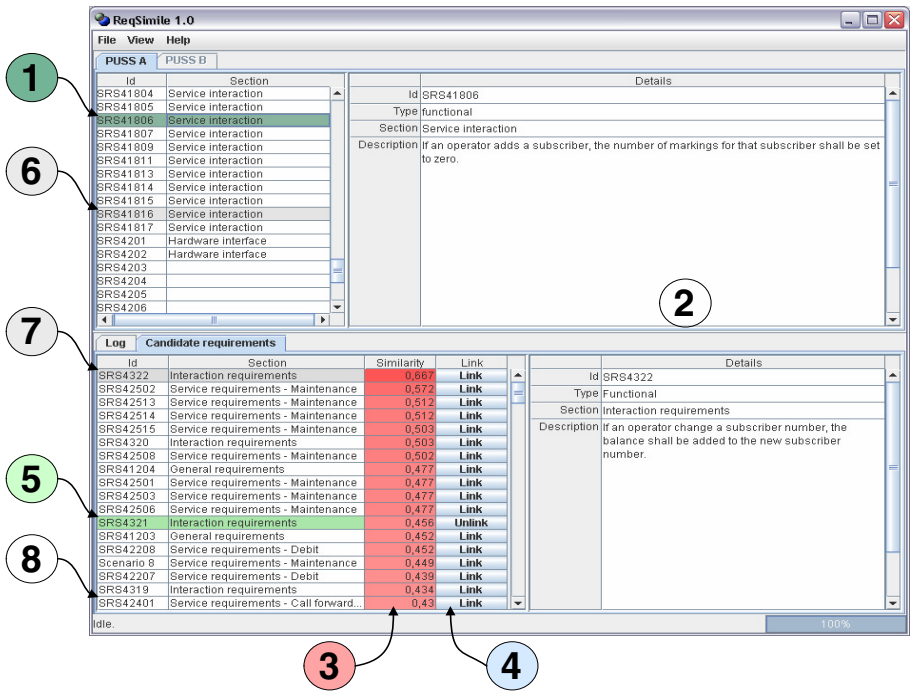


Figure 4: ReqSimileA - Providing automatic support

By clicking any of the Link buttons (4), a link is assigned between the selected requirement (2) and the requirement with the associated link button. The link is stored in an external database. A link may be removed by clicking the Unlink button, and the program will update the database accordingly. Linked requirements are highlighted according to the color scheme in the figure (5,6,7). All other requirements that are unlinked are uncolored (8).

ReqSimileM was modified so it did not provide an automatically calculated list of similar requirements. Instead, it provides the possibility to enter search terms (9), which are used to retrieve similar requirements. The person using ReqSimileM must consequently come up with relevant search terms by him/herself. The keyword search support was provided to mimic the search facilities available in requirements management tools[2], while, for experimentation purposes, keeping the differences between the version as small as pos-

[2]There is actually one tool, Focal Point (http://www.focalpointus.com), which do provide functionality to find similar requirements based on the research presented in (Natt och Dag et al., 2002))

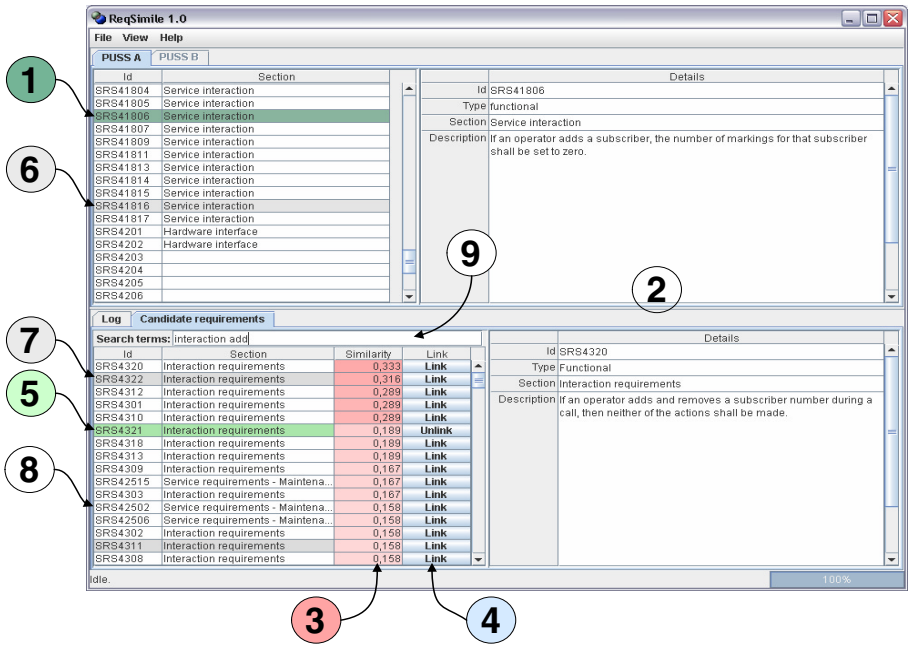


Figure 5: ReqSimileM - Providing manual key word search support

(1) **SELECTED REQUIREMENT** Currently selected requirement for which details are shown on the right and candidate requirements are shown on the bottom.

(2) On the right hand side the details are shown for a requirement that has been selected on the left.

(3) The requirements are sorted by similarity to the above *selected requirement*. The fields are colored with a shade between red and white, representing the similarity. Red, or 1, represents an exact match.

(4) This column has buttons for linking. **Link** is pressed to link and **Unlink** to unlink.

(5) **LINKED REQUIREMENT 1** This requirement has been linked to the currently selected requirement above.

(6) **LINKED REQUIREMENT 2a** This requirement has been linked. If you select it, the links may be reviewed using the bottom pane.

(7) **LINKED REQUIREMENT 2b** This requirement has been linked, but not to the currently selected!

(8) **UNLINKED REQUIREMENTS** White background means unlinked requirements. To link a requirement to the currently selected requirement, press the link button.

(9) **SEARCH TERMS** Search terms are entered to fetch requirements that match the search terms. Multiple terms are separated by space. If no search terms are entered, all requirements are shown.

sible. Actually, ReqSimileM provides better support than available requirements management tools as it sorts the resulting list of requirements based on the occurrence of keywords. Furthermore, the keywords are first preprocessed to remove inflections, etc., in the same manner as is done with the requirements in ReqSimileA.

Both versions of the tool were preset to use the requirements prepared as described in the next section. The original tool allows a user to select a requirement from either of the two sets (by clicking one of the tabs in the top half of the window named 'PUSS A' and 'PUSS B') in order to be presented with a list of similar requirements from the other set. To assure that the two experiment groups conducted the consolidation task in the same manner, both version of the tool were locked to enable selection from only one of the sets.

5.3 Requirements

For the consolidation experiment we used requirements specifications produced by students from the Master's degree program. The requirements specification had been produced as part of a course "Software Development for Large Systems"[3]. The course comprise a full development project including requirements specification, test specification, high-level design, implementation, test, informal and formal reviews, and acceptance test. At the end of the course the students deliver a first release of the controller software for an Ericsson switch board.

The requirements specifications, SRS A and SRS B, were chosen with respect to the experiment participants' knowledge in the domain. They had themselves taken the course one year earlier and were thus familiar with the requirements on such a system. Giving the course for 12 years we have seen that although the students are developing the same system and are guided to write their requirements in one particular way, they naturally write their requirements differently. Furthermore, in order to make the projects slightly different (partly in order to avoid the possibility to cheat) the projects are requested to develop different sets of services (e.g., redial, call forwarding, and take call). This results in different sets of requirements on two levels. First, the requirements required for each service and, secondly, the requirements resulting from resolving interaction issues between the services (e.g. in what way it is possible to take a call that has been forwarded to another subscriber).

[3]The course code is ETS032 and course information is available at http://serg.telecom.lth.se/education/

Two specifications were randomly selected from the course given in the year 2002 and 2003. Note that the specifications were taken from the Master's degree program and that the subjects were following the Bachelor's degree program (the programs are given in different cities, about 55 km apart). The requirements had been specified in the use case style or the feature style (Lauesen, 2002) and thus all requirements were written using natural language. In order not to burden the students (who produced the specification) with domain terminology challenges in both Swedish and English, they had been recommended to write their specifications in Swedish (they have challenges enough to handle in the project). The two specifications, respectively, comprised 139 and 160 unique requirements and scenarios (see also Section 6.3 about the requirements used in the experiment).

As our experimentation tool expects the requirements to be written in English (the underlying linguistic engineering techniques are adapted for the English language) the requirements specifications were independently translated to English by the first two authors. No discussion was allowed between the translators during the translation in order to reduce internal validity threats. In particular, we wanted to preserve the possibilities of using different wordings and injecting spelling errors. However, no errors or differences between the requirements sets were deliberately injected.

The experimentation tool expects the requirements to reside in a database so the requirements documents' structure was transferred to fields in a database table. The resulting structure and example requirements can be found in Tables 1 and 2. The two tables show requirements from the requirements specifications that are semantically similar. The examples also show the varying linguistic quality and the presence of differences between similar requirements. The two scenarios refer to the same underlying functionality. Note the error in the scenario in Table 1, where it says that no subscriber initially is unhooked (compare to the other scenario). Also note the left-out steps in the scenario in Table 2. Requirements SRS41606 in Table 1 and SRS42403 in Table 2 are similar but are written differently. Requirement SRS41804 in Table 1 refer to the same functionality as requirements SRS4306 and SRS4311 in Table 2 (together, i.e. a split requirement). Notice the alternate spelling of *canceled*.

In this experiment, the content from the *section* and the *description* fields were used to calculated the similarity between requirements. For more information about the underlying calculation procedure see (Natt och Dag et al., 2004).

Table 1: Example requirements from specification comprising 139 requirements

Key	Id	Type	Section	Description
3	Scenario 13	functional	Service: Regular call	Regular call - busy Actors: A:Calling subsriber, B:Called subscriber, S:System Prerequisites: Both A and B are connected to the system and are not unhooked. Step 13.1. A unhooks. Step 13.2. S starts giving dial tone to A Step 13.3. A dials the first digit in B's subscriber number Step 13.4. S stops giving dial tone to A. Step 13.5. A dials the remaining three digits in B's subscriber number Step 13.8. S starts giving busy tone to A Step 13.9. A hangs up Step 13.10. S stops giving busy tone to A
80	SRS41606	functional	Service: Call forwarding	Activation of call forwarding to a subscriber that has activated call forwarding shall be ignored by the system. This is regarded as an erroneous activation, and an error tone is given to the subscriber. (Motivaion: Together with SR41607, avoids call forwarding in closed loops)
111	SRS41804	functional	Service interaction	The service call forwarding shall be deactivated if an operator removes either the subscriber from which calls are forwarded or the subscriber to which calls are forwarded.

Table 2: Example requirements from specification comprising 160 requirements

Key	Id	Type	Section	Description
41	Scenario 13	Functional	Service requirements - Normal call	"Normal call - busy" Actors: A Calling subscriber. B: Subscriber which are called to. S: System. Condition: Both A and B are connected. A's receiver is on hook. B's receiver is off hook. Step 13.1-13.5. [The same as 11.1-11.5] Step 13.6. S gives busy tone to A. Step 13.7. A on hook. Step 13.8. S stops busy tone to A.
91	SRS42403	Functional	Service requirements - call forwarding	If call forwarding is done to a subscriber number which in its turn is forwarded, an error tone shall be used.
125	SRS4306	Functional	Interaction requirements	If a subscriber number is forwarded to another subscriber number and the later is removed, then the call forwarding shall be cancelled.
130	SRS4311	Functional	Interaction requirements	If a forwarded subscriber is removed, then the call forwarding shall be cancelled.

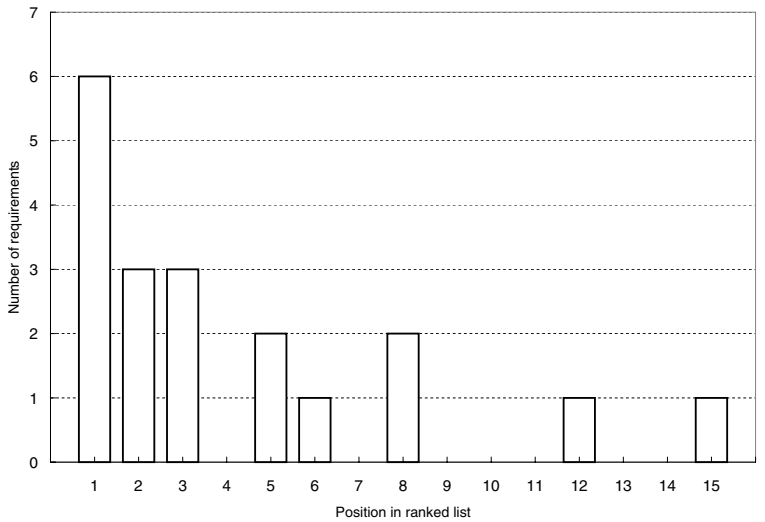


Figure 6: Histogram of the positions of similar requirements in the automatically produced ranked list of similar requirements

5.4 Correct consolidation

To enable measurement of the subjects' accuracy of linking requirements that are semantically similar, the first author created a key where the correct links had been assigned. The author have been teaching in the course for several years, taking different roles in the course, such as requirements expert, test expert, and expert reviewer, as well as holding exercises and laboratory sessions. It is therefore justifiable to regard the key as one provided by an expert in the domain. The key was created prior to any analysis of the subjects' assigned links in order to reduce any related validity threats.

A way of measuring the tool's ability of calculating similarities between requirements is too see at which position the correctly similar requirements end up in the ranked list produced by the tool. The requirements that the expert considers to be similar should preferably end up at the top of the list. Of the 30 requirements that the subjects were asked to link (see Section 6.3), 20 requirements were considered similar by our expert analysis.

In Figure 6, a histogram is found showing the distribution of the positions at which the correctly similar requirements end up in the ranked lists produced by the tool (see Section 5.2). As shown in the figure, almost all (17

out of the 20) ended up at position 8 or better in the ranked list and should therefore quickly be spotted. For presentation purposes, one data point is absent in the figure. One similar requirement was found at position 64 in the produced top list. We found two reasons for this. Firstly, vital information was available in another requirement (i.e. the requirement was split). Secondly, the varying lengths of the requirements seems to affect the similarity calculation adversely.

6 Experimental planning

In this section, the variables, the hypotheses and design of the experiment are described. In addition, the threats to the validity are discussed.

6.1 Variables

The variables in the experiment are described below, divided into independent, controlled, and dependent.

- The independent variable is the methods used in the experiment. The two methods compared are manual and assisted.
- The controlled variable is the experience of the participants. A questionnaire was used prior to the experiment to analyze the individual experience of the subjects.
- The dependent variables are:

  T time used for the consolidation

  N the number of analyzed requirements

  N_{cl} number of correct links

  N_{il} number of incorrect links

  N_{cu} number of correctly not linked

  N_{iu} number of missed links (incorrectly not linked)

  These dependent variables are used to analyze the hypotheses. The number of analyzed requirements are used in case the subjects are not able to analyze all requirements, which will affect N_{iu} and N_{cu}.

6.2 Hypotheses

The hypotheses of the experiment were defined to investigate whether the requirements engineers can be aided by using the assisted method. Six null hypotheses were defined, which are presented below. The *assisted method* refers to the method used by subject group that used the ReqSimileA tool. The *manual method* refers to the method used by the subject group that used the ReqSimileM tool (see Section 5.2 for a description of the tools).

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

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

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

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

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

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

In this experiment we are only interested to find out if the assisted method outperforms the manual. The reason for this is that it is of no practical importance if the assisted method is less effective than the manual method. So, we only want to reject the null hypotheses when the difference is in the expected direction. Therefore, we specify the following one-sided alternative hypotheses:

H_1^1 The assisted method results in a greater number of *requirements analyzed per minute*, N/T, compared to the manual method.

H_1^2 The assisted method results in a larger share of *correctly liked requirements*, $N_{cl}/(N_{cl} + N_{iu})$, compared to the manual method.

H_1^3 The assisted method results in a smaller share of *missed requirements links*, $N_{iu}/(N_{cl} + N_{iu})$, compare to the manual method.

H_1^4 The assisted method results in a smaller share of *incorrectly linked requirements*, N_{il}/N, compared to the manual method.

H_1^5 The assisted method is more *precise*, $N_{cl}/(N_{cl} + N_{il})$, than the manual method.

H_1^6 The assisted method is more *accurate*, $(N_{cl} + N_{cu})/(N_{cl} + N_{il} + N_{cu} + N_{iu})$, than the manual method.

6.3 Design

As a preparation to the experiment, a pilot experiment was performed to evaluate the instrumentation and design. Four colleagues participated and used the methods evaluated in the experiment. After the pilot experiment, it was concluded that the number of requirements should be reduced in order to enable the subject to consolidate an expected number of requirements within a lab session. Therefore, 30 requirements were randomly selected from the set of 139 requirements (see Section 5.3 about the requirements). The same set of 30 requirements were given to each subject during the experiment.

Before the main experiment, a questionnaire (pre-test) comprising 6 questions was handed out to the students to explore their experience from industrial software development and course participation, and skills in English reading and writing. The questionnaire showed that they had equal industrial experience (none) and equal experience from earlier courses. The differences in reading and writing skills were small, but could be used to randomly assign the participants to two experiment groups. This resulted in 11 students in the subject group that were given the ReqSimileM tool (groupM) and 12 students in the subject group that were given the ReqSimileA tool (groupA). One student in groupM and two students in groupA were removed from the analysis, since they did not perform all parts of the experiment. The experiment was carried out during two hours, which included a short introduction to the task, the experiment, and a post-test.

The experiment data have been analyzed with descriptive analysis and statistical tests (Montgomery, 2001). The data were checked for normal distribution (normal probability plots), and the conclusion was that parametric hypothesis tests could be used. The two-sample one-tailed t-test is used to investigate the hypotheses (Montgomery, 2001). The significance value of

rejecting the null hypotheses is set to 0.05 for all tests.

6.4 Threats to Validity

An important analysis when performing experiments is the validity of the results. The result of an experiment should be interpreted in the light of the threats to the validity, in order to help future replications, and generalization of the result. In this section, the threats are analyzed related to four groups of threats: Conclusion validity, internal validity, construct validity and external validity (Wohlin et al., 2000).

Conclusion validity concerns the relation between the treatments and the outcome of the experiment. The threats related to the statistical tests used in the experiment are considered being under control as normal probability plots have been used to check that parametric tests can be used. These have greater power that non-parametric, which is advantageous since only 23 subjects participated in the experiment. Threats with respect to the subjects are also limited since the subject groups are rather homogeneous, the subjects have attended the same education program during 2.5 years.

Internal validity of the experiment concerns the question whether the effect is caused by the independent variables or by other factors. The experiment has limited threats due to history, maturation, mortality, etc., since it is applied during a 2 hour period. The social threats are limited since the subjects had nothing to gain from the actual outcome of the experiment. The grading of the courses was only based on their participation in the experiment, not on their performance. The threat of selection is also under control, as the experiment is a mandatory part of a course. However, three subjects were considered as outliers since they did not make their best effort during the experiment session, see Section 8.

A potentially more problematic threat is that the subjects had to analyze and link requirements written in English when they had themselves used only Swedish to specify their own requirements in the domain. This threat were minimized through the pre-test were we asked about their ability to read and write common and technical English. Although the questions were answered through self-assessment, the students' answers that they generally can read and write both common English and technical English quite easily or fluently, is credible. Swedish students generally speak and write English well and a major part of their course material is in English. Furthermore, our own experience from translating the specifications suggests that it is easier to understand the Swedish concept based on the English written one, than to translate to the

correct English term from Swedish.

Construct validity concerns the ability to generalize from the experiment result to the concept behind the experiment. There are two potential threats, experimenter expectations and interaction of testing and treatment. The main experimenter developed the tool, which is the basis of the persons research. Hence, he expected the tool to be better than the manual method. To reduce this threat, two other peopled were included as experimenters (planning, operation and analysis), who are not directly involved in the research of the main experimenter. Furthermore, since two measures in the experiment are the number of correct and faulty links, the subjects may have been more aware of their errors. Also, when the subjects know that the time is measured, it is possible that they get more aware of the time they spend, and thus the time-consumption is affected. Subject's awareness cannot be controlled, but the analysis do not point in that direction.

The experiment would need another set of requirements in order to enable us to discover whether the results are the same, or if the set of requirements have affected the results.

External validity concerns the ability to generalize the results to industry practice. The largest threat is that only a few requirements have been used in the experiment. With larger sets, the automatic similarity calculation may not result in high ranking of actually similar requirements. Earlier research investigates this with larger sets (Natt och Dag et al., 2002, 2004) and shows that the techniques are good enough to give valuable support. Nevertheless, further experimentation is motivated.

Another large threat is that students are used as subjects. However, the students are in their third year of software engineering studies and thus close to start working in industry. The participants are familiar with the application domain, which is industry-like, as they participate in a requirements engineering course. Furthermore, as most experimental conditions, the time is an important factor. In order to reduce the fatigue effect, the number of prioritized requirements are fewer than in most real cases. Thus, it is difficult to judge whether extending the number of requirements would lead to the same result.

In summary, the threats are not considered large in this experiment. The main threats are that fewer requirements were used than in a real case and that students were used as subjects. Hence, future replications and case studies have to be performed in order to reduce these threats.

7 Experimental operation

The experiment was run in one two-hours lab session in fall 2004. The first 15 minutes of the 90 minutes lab session was dedicated to a presentation of the general problem, the industrial applicability, and the goal of the lab session. All students were given the same presentation, which also included an overview of the generic aspects of the two versions of the tool. The general differences between the tools were communicated without favoring one over the other. To avoid biasing, no hypotheses were revealed and it was made very clear that we did not know whether one of the approaches was in any way superior over the other.

After the presentation, the students were separated into two groups depending on which version of the tool they were going to use during the experiment. The two groups were directed to separate rooms and instructed to prepare for and begin the task as written in the material. Each group was accompanied by a supervisor. The students were not allowed to discuss their work with other students during the experiment. However, they were allowed to ask questions to the supervisor if they experienced any problems. Some students asked questions, which were mainly about how two requirements could be regarded as similar or not. Direct answers to these questions were not given. Instead we referred to the definitions given in the material and rephrased or clarified these if needed.

The material used in the experiment comprised:

- The ReqSimile package. This had been put into a zip file, which was downloadable from the course home page. Depending on the group the students were asked to download one of the packages and run the application. The package included:
  - The ReqSimile application, either with automated support or for manually specifying search terms.
  - MS Access database including
    - 30 randomly selected requirements from SRS A. These could be browsed and was to be linked to requirements in SRS B.
    - All 160 requirements from SRS B. These could be browsed through the suggestion list which was, depending on the tool version, presented upon selection of a requirements from SRS A or upon entering search terms in the search text box.

    * Preprocessed requirements. Separate tables to speed up the similarity calculation. For the full set of requirements this stage only took only 3 seconds but was removed to minimize the complexity of the task.
    * Empty link table in which the links set by the students would be stored.

- A document including:
  - Industrial scenario describing the actual challenge
  - A general task description
  - Detailed tasks with room for note down start and end times.
  - A short FAQ with general questions and answers about the requirements
  - A screen shot of the tool user interface with descriptions of the different parts.

The instructions to the students were:

- Walk through as many of the requirements as possible of the 30 shown in the tool. For each requirements, investigate if there are any requirements in the other set that can be considered identical or very similar (only differing in some detail) with respect to intended functionality.
- Assign a link between requirements that are considered identical or very similar.
- Log the times for start and finish
- When finished, notify the supervisor.

The subjects were initially given 60 minutes for the consolidation task, but due to operating system difficulties (promptly resolved by the subjects) the subject had about 45 minutes to consolidate the requirements.

15 minutes before the end of the lab session, the students that had not analyzed all 30 requirements were ask to stop working. All students were then given a post-test document. In the post-test document they were asked to note down the finishing time and the number of requirements they had been analyzing.

Table 3: The collected experimental data

Subject	T (min)	N	N_{cl}	N_{cu}	N_{il}	N_{iu}
A1	47	27	11	7	18	6
A2	44	17	7	5	12	3
A3	46	25	9	8	12	7
A4	46	17	8	5	12	2
A5	40	30	8	11	1	12
A6	37	17	6	4	8	4
A7	29	3	1	0	6	0
A8	30	3	1	0	3	0
A9	45	13	5	3	8	3
A10	47	22	12	5	9	1
A11	45	21	10	5	11	2
A12	35	30	14	7	11	6
M2	43	19	7	6	6	5
M3	42	19	6	5	6	6
M4	43	15	2	3	10	7
M5	45	15	6	2	10	3
M6	45	20	4	6	6	8
M7	45	13	4	3	17	4
M8	45	15	4	5	2	5
M9	45	16	5	5	3	4
M10	45	22	9	6	11	4
M11	40	15	3	6	1	6
M12	60	4	0	3	2	1

8 Analysis

This section presents the data collected during the experiment and the results from the statistical analyses used to investigate if any of the null hypotheses could be rejected. Each of the different hypotheses will be addressed separately and conjectures will be made iteratively based on the accumulated results from the hypothesis tests. All the measurements of the dependent variables are found in Table 3. It should be noted that one analyzed requirements can be linked to several others. Therefore, no functional relationship can be found between the number of analyzed requirements and the other measures.

8.1 Performance

The background for the experiment and the industrial case shows that it is a time-consuming task to consolidate requirements. Therefore, one of the first interesting collected data is the number of requirements the subjects were able to analyze during the time of the experiment.

The subject did not use the same amount of time so we calculated the number of requirements they were able to analyze per minute. This would be an appropriate measure of their performance. Figure 7 shows the results and it is clear that there is a significant difference between the manual method and the assisted method. GroupA analyzed in average 0.35 requirements per minute (2.8 minutes per requirement), while groupB analyzed in average 0.22 requirements per minute (4.5 minutes per requirement). In this case, the means and the medians coincide. The notches of the box plot gives a graphical representation of the significance of the difference between two means. In all the presented box plots the means of the data sets are significantly different at approximately the 0.05 level if the box plot's notches do not overlap.

In this particular case it is possible to reject the null hypothesis H_0^1 at a level of significance below 0.004. Thus, using the assisted method provides a significantly greater performance than using the manual method. This is a welcomed result, as we are interested in saving time in industry. However, further analysis is required before we may draw any final conclusions about the approach.

8.2 Quality

Naturally, nothing can be said whether the assisted method is actually better than the manual if the quality of the subjects performance is not evaluated. Three hypotheses have been formulated to decide if the assisted method enables the subjects to assign more correct, fewer missed, and fewer incorrect links (Section 6.2).

To test the null hypothesis H_0^2, we compared each subject's assigned links with the correct answers produced from our expert analysis. The result is shown in Figure 8. Again, it clearly shows a difference between the manual and assisted method. GroupA correctly assigned in average 68% of the links that the expert assigned (the median is just 0.6% less), while groupB correctly assigned in average 48% of the links (the median is, as can be seen in the figure, 50%).

The t-test shows that we can reject the null hypothesis at a significance

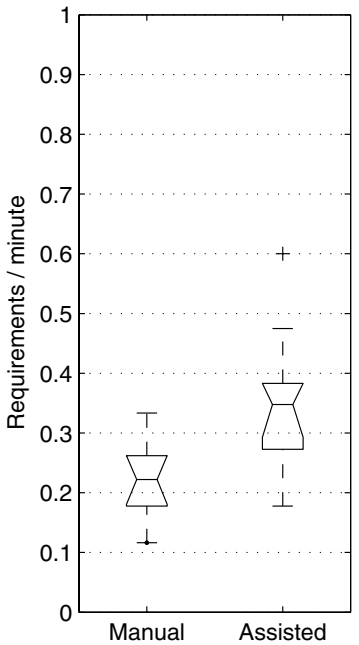


Figure 7: Analyzed requirements per minute.

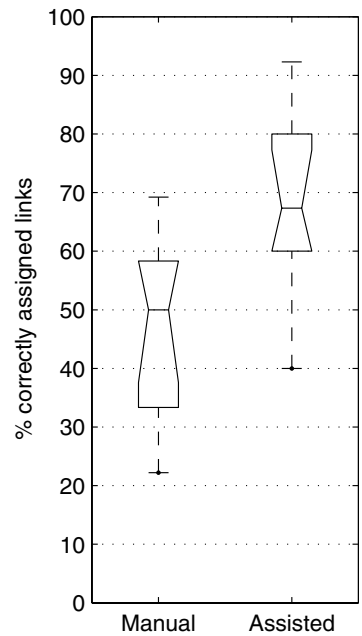


Figure 8: Correctly linked requirements.

level below 0.005. We may thus conclude that the assisted method enables the subjects to correctly link a significantly larger number of requirements than if they had used the manual method. Together with the performance statistics, this suggests that the assisted method is superior to the manual. However, further quality aspects exist.

For hypothesis H_0^3, we looked at each subject's analyzed requirements and counted how many links that should have been assigned, but were not. Since the subjects did not analyze the same number of requirements, it is important to only consider the requirements they had been able to analyze. The subjects had stated in the post-test how many requirements they had analyzed and they had worked through the requirements one after the other. This enabled us to correctly count the number of missed links compared to the our expert analysis. Figure 9 shows that there is a significant difference. GroupA missed in average 32% of the links that the experts assigned (median is about 0.5% higher), while groupB missed in average 52% of the links (median is 50%).

The t-test also reveals that we may reject the null hypothesis on a significance level below 0.005. This enable us to make the conjecture that the

assisted method helps the subjects to miss significantly fewer requirements links. This only makes the case stronger; that the assisted method is to prefer over the manual.

A conclusive quality measure would be the number of links that have been incorrectly assigned between two requirements. In Figure 10, the number of incorrectly assigned requirement for the two subjects groups are shown. It can be noted from the figure that the means are not significantly different (In average 51% for groupA and 53% for groupB, with medians 52% and 57% respectively). A t-test also acknowledge that we are not able to reject the null hypothesis H_0^4. Although the figure indicates that the subjects using the assisted method actually assign more incorrect links, there is no statistically significant difference.

How does the last statistic affect our case? In the consolidation process it would be very unfavorable to say that two requirements are similar when they are not, because those requirements might not be further analyzed and the mistake would be discovered too late. The error is human and must be addressed in other ways. One way is to raise the threshold for linking two requirements, meaning that if the human analysts are only the slightest hesitant of the similarity between two requirements, they should let it be. The requirement will the be sent to the Area of Expertise who makes a proper analysis.

8.3 Precision and accuracy

There is another way to assess the quality of the subjects performance by using the concepts of precision and accuracy. Precision is a measure that takes into account both the correctly and incorrectly linked requirements. Using an analogy, it is a measure of how much weed we have in the harvest. Accuracy, then, is a measure of how well the result conform to the truth. The truth in this case would be to assign all links correctly, not miss any, and not link anything that should not have been linked.

Unfortunately, both precision and measure incorporates the number of incorrectly linked requirements, which has a great impact on the measures. The number of incorrectly linked requirements are comparatively large in relation to the other measures (see Table 3). Therefore, it is not likely that we will find any significant differences between the two groups. The statistics of the incorrectly linked requirements propagates to the statistics of these measures. And quite so, our analysis of precision and accuracy of the two subject group shows now statistical differences. Thus, we are unable to reject both

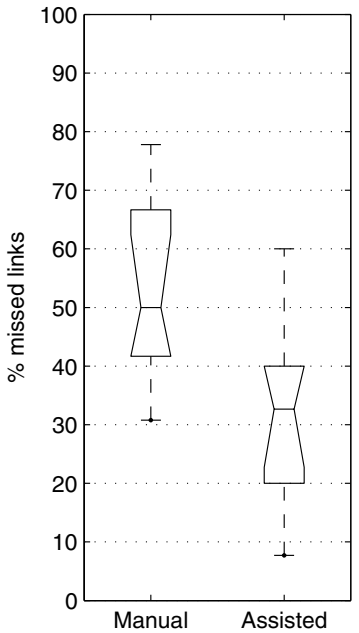


Figure 9: Missed requirements links.

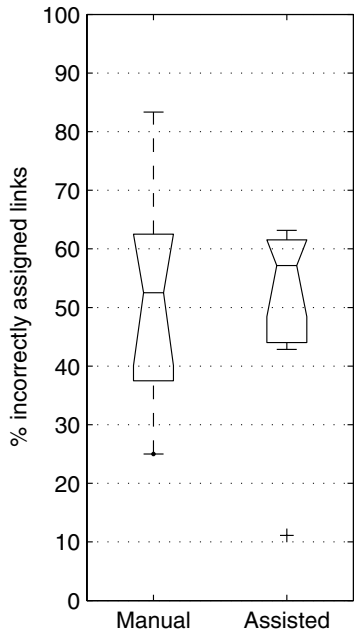


Figure 10: Incorrectly linked requirements.

hypothesis H_0^5 and H_0^6.

9 Discussion

The analysis of the experiment data allows for the rejection of three out of six null hypotheses (Table 4). Although it can not be statistically determined if there are any differences in precision and accuracy, the other results point in the same direction: that the assisted method is superior to the manual.

Previous investigations by Natt och Dag et al. (2002, 2004) suggests that linguistic engineering techniques may give support in certain requirements management activities. These activities (duplicate identification and linkage of customer wishes and business requirements) rely on an underlying activity which they share with the requirements consolidation process: that of finding and matching similar requirements. Previous research suggests that the linguistics engineering techniques may support this activity and that time could be saved in requirements management. The results from this experiment gives further evidence to this claim.

The results from this experiment are also supported by initial evaluations

Table 4: Summary of the results of the hypotheses

Hypothesis	p-value
H_0^1, Speed	0.0034 (significant)
H_0^2, Correct links	0.0047 (significant)
H_0^3, Missed links	0.0047 (significant)
H_0^4, Incorrect links	0.39
H_0^5, Precision	0.39
H_0^6, Accuracy	0.15

made at SEMC. In parallel with the operation and analysis of this experiment, two students have worked on the ReqSimile tool, adapting it to the specific requirements management process at SEMC and to the requirements analysts' needs. The initial evaluation with experts is very promising, indicating time-savings up to 30-40% in the activities of consolidating and analyzing RFIs. The experts have also provided their subjective feedback on the tool, which is very positive. Further studies are motivated, but that even initial results from a real industrial setting point in the same direction as the results from the presented experiment, makes the case somewhat stronger.

As always, when conducting experiments to increase the body of knowledge, the experiment should be replicated in different contexts. This experiment has been conducted in a laboratory setting and it would now be interesting to use experts as subjects in a real project.

Improvements to the tool and the underlying techniques are of course possible. Different models for calculating similarity are possible. This is elaborated further in related work (Natt och Dag et al., 2002, 2004, 2005; Natt och Dag and Gervasi, 2005).

10 Conclusions

In this paper we have presented an experiment with a tool that aims at giving support in a specific large-scale requirements management activities, namely requirements consolidation. A background and an industrial case has been presented to explain the origin of the requirements consolidation activity and to motivate the experiment. Experiences from initial evaluation of the tool in a real industrial environment have also been reported.

Two treatments were compared. One *assisted*, in which the subjects were presented with an automatically calculated ranked list of similar requirements,

and one *manual* in which the subject had to come up with their own search terms to find similar requirements. The task for the subjects was to assign links between requirements that could be considered similar.

The main results from the experiment is that the subjects using the assisted method are able to assign more correct links and also do this faster. Furthermore, they miss fewer links than the subjects using the manual method. The important results from the experiment are:

Speed The subjects using the assisted method are able to analyze significantly more requirements than the subjects using the manual method.

Correctness The subjects using the assisted method assign significantly more correct links than the subjects using the manual method.

Missed links The subjects using the assisted method miss significantly fewer links than the subjects using the manual method.

The presented experiment is an experiment at the first level of experimentation, i.e. conducted in a laboratory environment (Juristo and Moreno, 2001). The results from the experiment are promising for proceeding with both replications and with new experiments in a real industrial environment.

Acknowledgments. The authors would like to thank Daniel Karlström, Per Runeson, Martin Höst, and Lena Karlsson for participating in the pilot experiment. The authors would also like to thank the students for participating in this investigation and Lena Karlsson for acting as a stand-in experimenter during the experiment session with the students.

References

Brinkkemper, S. (2004). Requirements engineering research the industry is (and is not) wating for. In Regnell, B., Kamsties, E., and Gervasi, V., editors, *Proceedings of the 10th Anniversary International Workshop on Requirements Engineering: Foundations for Software Quality*, pages 267–284, Essen, Germany. Essener Informatik Beiträge.

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

Hofmann, H. F. and Lehner, F. (2001). Requirements engineering as a success factor in software projects. *IEEE Software*, 18(4):58–66.

Juristo, N. and Moreno, A. M. (2001). *Basics of Software Engineering Experimentation*. Kluwer Academic Publishers, Boston, MA.

Karlsson, L., Dahlstedt, A. G., Natt och Dag, J., Regnell, B., and Persson, A. (2002). Challenges in market-driven requirements engineering - an industrial interview study. In Saliensi, C., Regnell, B., and Pohl, K., editors, *Proceedings of the Eigth International Workshop on Requirements Engineering: Foundation for Software Quality*, pages 37–49, Essen, Germany. Essener Informatik Beiträge.

Kotler, P. (2002). *Marketing Management*. Prentice Hall, Upper Saddle River, NJ.

Krueger, C. W. (2003). Towards a taxonomy for software product lines. In *Proceedings of the 5th International Workshop on Product Family Engineering*, Siena, Italy.

Lauesen, S. (2002). *Software Requirements: Styles and Techniques*. Addison-Wasley, London, UK.

Lubars, M., Potts, C., and Richter, C. (1993). A review of the state of the practice in requirements modeling. In *Proceedings of IEEE International Symposium on Requirements Engineering*, pages 2–14, Los Alamitos, CA. IEEE CS.

Mich, L., Franch, M., and Novi Inverardi, P. L. (2004). Market research for requirements analysis using linguistic tools. *Requirements Engineering*, 9(1):40–56.

Montgomery, D. C. (2001). *Design and Analysis of Experiments*. John Wiley & Sons, USA.

Natt och Dag, J. and Gervasi, V. (2005). Managing large repositories of natural language requirements. In Aurum, A. and Wohlin, C., editors, *Engineering and Managing Software Requirements*. Springer-Verlag.

Natt och Dag, J., Gervasi, V., Brinkkemper, S., and Regnell, B. (2004). Speeding up requirements management in a product software company: Linking customer whishes to product requirements through linguistic engineer-

ing. In *Proceedings of the International Requirements Engineering Conference (RE2004)*, pages 283–294, Kyoto, Japan. IEEE CS.

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

Natt och Dag, J., Regnell, B., Carlshamre, P., Andersson, M., and Karlsson, J. (2002). A feasibility study of automated natural language requirements analysis in market-driven development. *Requirements Engineering*, 7(1):20–33.

Novorita, R. J. and Grube, G. (1996). Benefits of structured requirements methods for market-based enterprises. In *Proceedings of the Sixth Annual International Symposium on Systems Engineering (INCOSE'96)*, Boston, MA.

Potts, C. (1995). Invented requirements and imagined customers: Requirements engineering for off-the-shelf software. In *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, pages 128–130, Los Alamitos, CA. IEEE CS.

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

Sawyer, P., Sommerville, I., and Kotonya, G. (1999). Improving market-driven RE processes. In *Proceedings of International Conference on Product Focused Software Process Improvement (PROFES'99)*, pages 222–236, Oulu, Finland.

Sommerville, I. (2001). *Software Engineering*. Pearson Education, Harlow, UK, 6th edition.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2000). *Experimentation in Software Engineering - An Introduction*. Kluwer, Norwell, MA.

Yeh, A. (1992). Requirements engineering support technique (REQUEST) - a market driven requirements management process. In *Proceedings of the Second Symposium on Assessment of Quality Software Development Tools*, pages 211–223, Los Alamitos, CA. IEEE CS.

APPENDIX A

ReqSimile Technical Architecture

Johan Natt och Dag

Technical Report, CODEN:LUTEDX(TETS-7206)/1-28/(2005)&local1,
Department of Communication Systems, Lund University, Sweden

A

1 Change history

Date	Version	Description	Author
2005-01-06	1.0	First version of the software architecture for ReqSimile version 1.2	Johan Natt och Dag

2 Introduction

This document provides an overview of the technical architecture of the open-source tool ReqSimile. The primary purpose of ReqSimile is to provide semi-automatic support to requirements management activities that rely on one common activity: finding requirements that are semantically similar, i.e., refer to the same underlying functionality.

This document is intended to capture and convey the significant architectural decisions which have been made in designing and building the system. The purpose is to support future development and enable other researchers and developers to relatively easy adapt the tool to suit their particular needs.

ReqSimile may be used for any purposes of matching two arbitrary input sets, but is specifically aimed at providing a way to link requirements between very large input sets. It aims at saving time in the management of large amounts of requirements.

3 Background and related work

The ReqSimile tool was originally developed by Johan Natt och Dag for providing the means for investigating the benefits of using linguistic engineering techniques to support requirements management activities. Another motive was to enable and speed-up transfer of applied research to industry. It was developed based on the research conducted primarily by the following researchers:

- Johan Natt och Dag at the Department of Communication Systems, Lund University, Sweden.
- Dr. Björn Regnell at the Department of Communication Systems, Lund University, Sweden.
- Dr. Vincenzo Gervasi at the Computer Science Department, University of Pisa, Italy.
- Prof. Sjaak Brinkkemper at the Institute of Information and Computing Sciences, Utrecht University, The Netherlands.

References to the related research may be found in the bibliography at the end of this document.

4 Definitions

5 Architectural representation

The modeling, implementation, and documentation of a software system is made using different views. This documents uses the views specified by the Software Engineering Institute (SEI) (Clements et al., 2002):

Module view (Mod) Describes the structure of the system with respect to the static, logical units.

Component & Connector view (C&C) Describes the run-time behavior and the interaction between different elements in the system.

Allocation view (Alloc) Describes how the architecture relates to the environment, e.g., how the code is organized.

The architecture of ReqSimile is represented using the specific views listed in Table 1. For further documentation of the methods in the classes, JavaDoc documentation is available together with each release of the ReqSimile tool (Natt och Dag, 2004).

Table 1: Architectural views

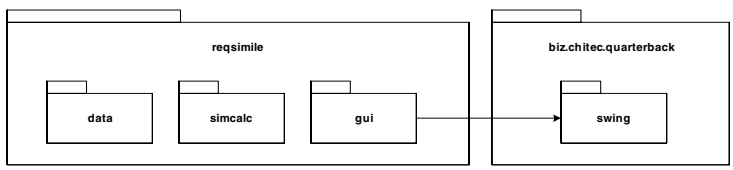
View	Type	Description
ReqSimile Decomposition	Mod	The packages of the system
ReqSimile Dependencies	Mod	The dependencies between the packages
ReqSimile Classes	Mod	The classes and interfaces in the 'reqsimile' package
ReqSimile Data Classes	Mod	The classes and interfaces in the 'reqsimile.data' package
ReqSimile SimCalc Classes	Mod	The classes and interfaces in the 'reqsimile.simcalc' package
ReqSimile GUI Classes	Mod	The classes and interfaces in the 'reqsimile.gui' package
ReqSimile Startup	C&C	Activity diagram showing execution flow at startup.
ReqSimile Preprocess requirements	C&C	Activity diagram showing execution flow for preprocessing requirements.
ReqSimile Fetch requirements	C&C	Activity diagram showing execution flow when a selecting a requirement or submitting additional search terms.
ReqSimile Shutdown	C&C	Activity diagram showing execution flow at shutdown.

6 Architectural views

6.1 View: ReqSimile Decomposition

Primary presentation

ReqSimile is organized according to the Java package structure below.

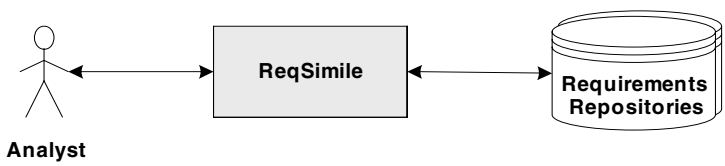


Element catalog

Class/interface	Description
reqsimile	Comprise all application-generic classes
data	Comprise classes for all database operations, including fetching requirements, and storing preprocessed requirements
simcalc	Comprise classes for preprocessing requirements and calculating similarity between requirements
gui	Comprise classes for the graphical user interface
swing	External class collection comprising 3rd-party Swing utilities

Context diagram

The view represents the whole system, represented by the shaded box below.



Variability guide

No open questions.

Architecture background

The early decision to select Java as the language for implementation has affected the architecture from start (the main reason for choosing Java is its portability and wide-spread use). However, the architecture is not made dependent on the particular language.

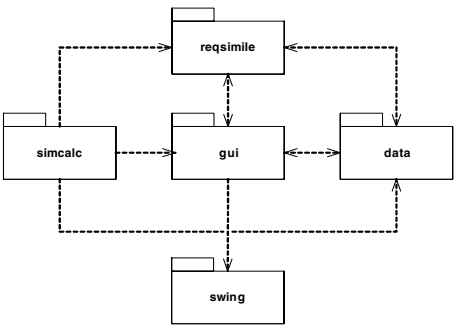
The external Java QuarterBack class collection was chosen for its open source availability also licensed according to GPL (GNU public license). Currently (2005-01-07), only one class is used: TableCellSizeAdjustor. Implementation in another language than Java would just make the dependency on this class obsolete.

The package structure was initially selected for its straight-forwardness. Refactoring has been made continually in order to comply to the initial design and to avoid architectural deterioration.

6.2 View: ReqSimile Dependencies

Primary presentation

Dependencies between the Java packages are shown below.



Element catalog

Class/interface	Description
reqsimile	Comprise all application-generic classes
data	Comprise classes for all database operations, including fetching requirements, and storing preprocessed requirements
simcalc	Comprise classes for preprocessing requirements and calculating similarity between requirements
gui	Comprise classes for the graphical user interface
swing	External class collection comprising 3rd-party Swing utilities

Context diagram

Not applicable

Variability guide

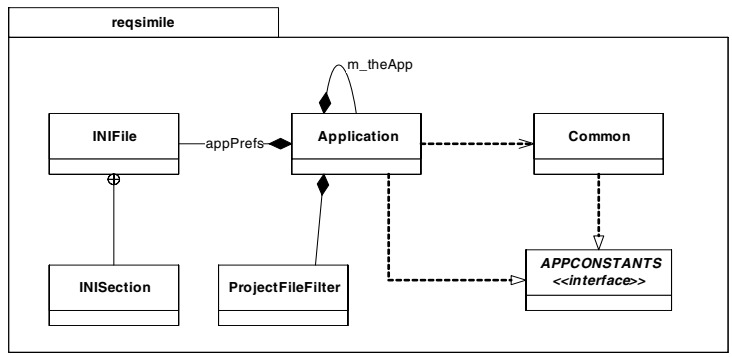
No open questions.

Architecture background

The system is not very large, but the almost complete dependencies and communication between packages, may call for an overview of the current architecture and potential changes.

6.3 View: ReqSimile Classes

Primary presentation



Element catalog

Class/Interface	Description
APPCONSTANTS	Contains all application-wide constants
Application	Application is the starting point of the application, which creates the central classes and provides information about these (such as the application itself, the gui, database connections, etc.).
Common	Contains generic functions that are used throughout the application.
INIFile	INIFile class provides methods for manipulating (Read/Write) files in the INI format (Microsoft).
INISection	Private class representing the INI Section.
ProjectFileFilter	Provides the project extension filter to the file dialog.

Context diagram

Not applicable

Variability guide

Project file handling (reading and writing) could be put in a separate class.

Architecture background

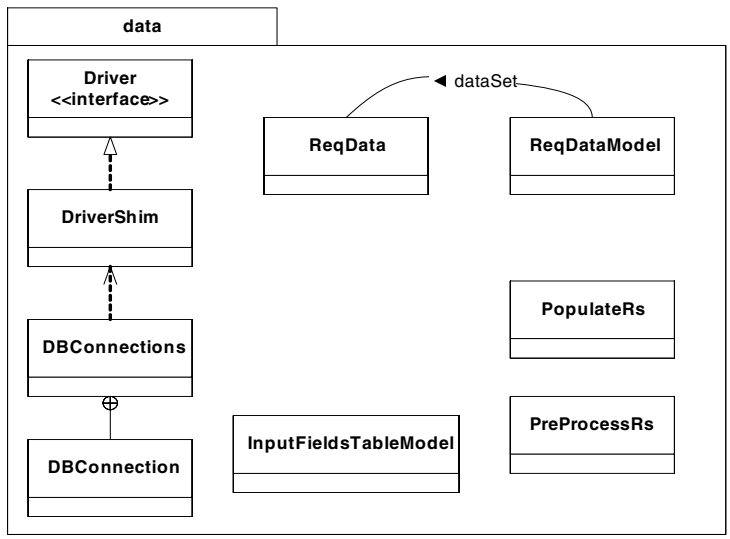
The `APPCONSTANTS` interface is implemented by virtually every class. This design is to some regarded as a violation of good style. The reason for imple-

menting it as an interface is that access to the constants becomes much easier. Instead of writing `APPCONSTANT.CONSTANT_NAME` to access the constant, it is possible to write just `CONSTANT_NAME`. To switch to good style, which is recommended, the `APPCONSTANTS` interface should be implemented as a static class.

The `INIFile` class is based on the code in the article by Prasad P. Khandekar (found at http://www.codeproject.com/useritems/INIFile.asp). Since it did not measure up to what was needed, several major enhancements are made, which are found in the change log in the source file.

6.4 View: ReqSimile Data Classes

Primary presentation



Element catalog

Class/Interface	Description
ReqDataModel	The middle layer between the actual data and a table in the user interface. It provides a JTable with the data to present in the table and fetches that data from an associated data set instantiated from ReqData. (ReqData) => (ReqDataModel) => (JTable).
ReqData	Holds all the requirements data used. It is used by ReqDataModel to present all or a subset of the available information.
PopulateRs	Reads all the data from the database and populates the data sets from which the tables get their data to display. Any field as specified to be included in the summary view, detail view or the calculation are fecthed from the database. The field for calculation are used in the preprocessing step (PreProcessRs) This class should not be used if data is to beread on a needs basis. The current approach is not memory efficient as it reads all data into memory. It is done for speed purposes and because the program currently can not answer incrementally to any database changes. The preprocessing is made upon request and on the snapshot of the database, read by the run method in this class.

Class/Interface	Description
PreProcessRs	Preprocesses a requirement according to these steps: 1. Tokenization, stemming and stop word removal of each requirement. 2. Updates the table that holds all distinct tokens (and token ids). 3. Updates the frequency table that keeps the representation of which tokens occur in the requirements, and their corresponding frequency.
InputFieldsTableModel	The table model for the tables in the settings dialog used for selecting which fields that are to be displayed in the summary and detail views, and be used for calculation.
DriverShim	A shim class used as a workaround to enable dynamic loading of JDBC driver at runtime. The workaround makes it possible to use a driver that is not in the classpath, nor in the jar.
DBConnections	Creates and saves all database connections. This is done so that database connections may be reused.
DCConnection	Holds one connection to a database.

The following classes and interfaces are part of the Java2 language.

Class/Interface	Description
Driver	Interface that every driver class must implement.

Context diagram

Not applicable.

Variability guide

No open questions.

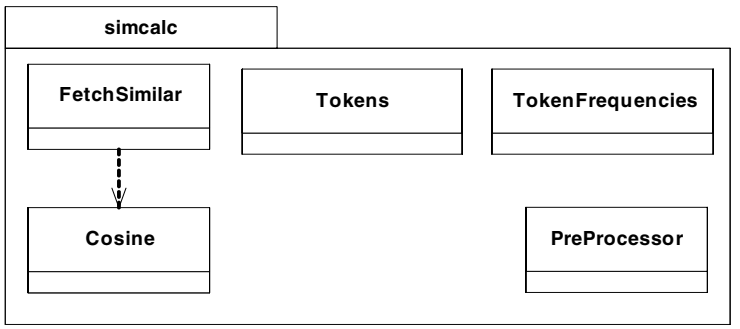
Architecture background

The ReqDataModel class is designed to provide the data for all table views, both the summary view (list of requirements) and the detail view (requirements details). This has made several methods rather complicated and it would be a good idea to split it up for easier maintenance.

DBConnections will accept connection using the JDBC interface for highest portability (several native drivers and bridges exist for various platform). It could be possible to also support common file formats, but tabular data in virtually any type of file may be accessed via JDBC as long as there is a driver.

6.5 View: ReqSimile SimCalc Classes

Primary presentation



Element catalog

Class/Interface	Description
FetchSimilar	FetchSimilar fetches all similar requirements to a specified requirements (e.g. a requirements selected in a table). It is run every time a new list of requirements shall be fetched. For performance reasons this should be kept simple and fast.
Cosine	Cosine contains helper methods to calculate the Cosine angle between two requirements in the word space. In the current implementation, the Cosine is calculated in a two-step process: • Preprocessing where weights and square sums calculated • SQL statement in which the final calculation is made This class contains only helper functions. For further information on how the measure is calculated see PreProcessRs and FetchSimilar.
Tokens	Tokens temporarily holds all the tokens/words used in the requirements sets. It is used in the preprocessing step to rebuild the token table, which is stored in an external database table.
TokenFrequencies	TokenFrequencies temporarily holds the token frequencies for one single requirement. It is used in the preprocessing stage to rebuild the complete token frequency table (for a whole requirements set), which is stored in an external database table.
PreProcessor	This class is a scanner generated by JFlex 1.4 from the flex specification file (preprocessor.flex). The scanner handles tokenization, stemming and stop word removal at once, implemented through a finite state automata (generated by JFlex).

Context diagram

Not applicable.

Variability guide

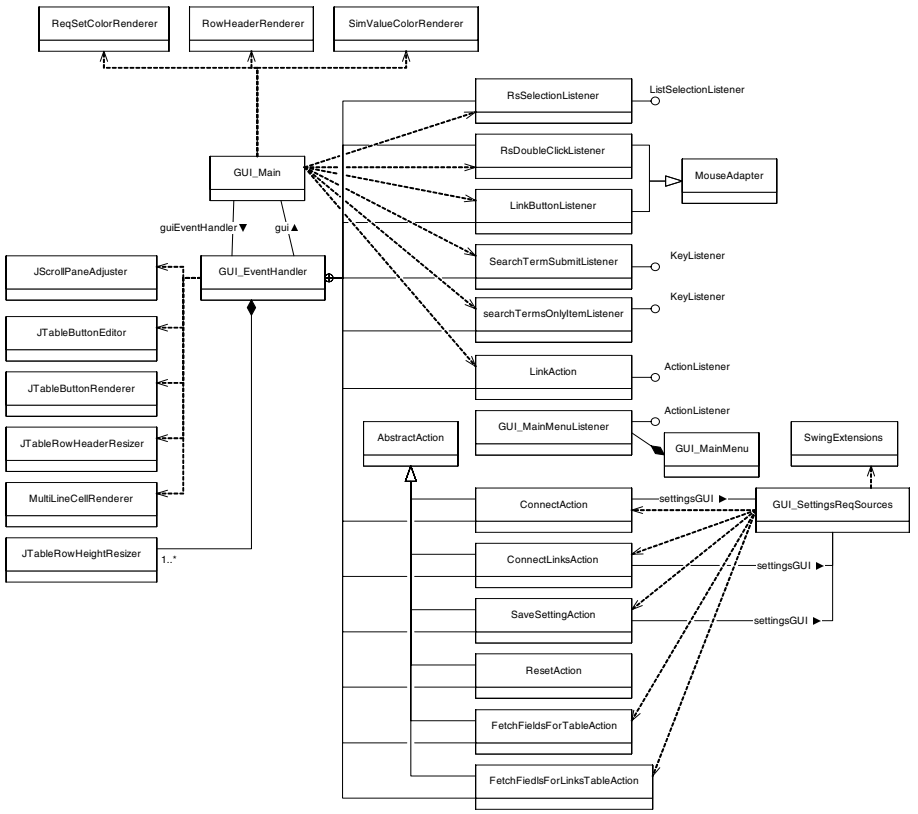
No open questions.

Architecture background

For performance reasons, all requirements will be preprocessed before similar requirements are calculated (i.e. during the main operative mode of the tool). The preprocessing step is made using the PreProcessor scanner, which is generated from a flex grammar. The size of the generated source is huge, but the run-time imprint is small and performance is high. The preprocessing takes less than 0.03 seconds per requirement (based on average calculated from preprocessing 11,723 requirements of various sizes on a 1.7 GHz Pentium II processor).

6.6 View: ReqSimile GUI Classes

Primary presentation



Element catalog

Class/Interface	Description
GUI_EventHandler	GUI_Eventhandler handles all events resulting from interaction with the gui by the user. This includes all buttons, table selections but not menu actions. Menu actions are handled by GUI_MainMenuListener. This class contains all classes derived from the associated actions or listeners.
GUI_Main	This class comprises the main GUI of the application.
GUI_MainMenu	GUI_MainMenu defines the main window menu.

Class/Interface	Description
GUI_MainMenuListener	GUI_MainMenuListener handles the actions to be taken upon selection of menu items in the application menu.
GUI_SettingsReqSources	Defines the settings dialog window (for setting data sources).
JScrollPaneAdjuster	Java bug-fix class. Workaround for synchronization of scrolling in table header and viewport. JScrollPane's support for synchronization between row/column headers and the main view is asymmetric. While it adjusts the headers if the position in the view changes, it doesn't do so if the headers' positions changed. This means that even a simple scrollRectToVisible call on a row/column header component will scroll the header, but not properly adjust the main view. This happens for example if you use a JTable as row header and the user selects by keyboard or by dragging, as then implicit scrolling happens. This is a known bug (#4202002).
JTableButtonEditor	Decides what happens when the Link/Unlink button is pressed in a table (according to general Java Swing guidelines).
JTableButtonRenderer	Represents the visual appearance of the Link/Unlink button (according to general Java Swing guidelines).
JTableRowHeaderResizer	Support class to enable resizing of a table row header.
JTableRowHeightResizer	Support class to enable resizing of the height of a table row.
MultiLineCellRenderer	Extends JTextArea to display a long line in a table as a multiline row using line wrap.
ReqSetColorRenderer	Provides coloring of table cells based on the requirements status (e.g. selected, linked, etc.)
RowHeaderRenderer	Define the look/content for a cell in the row header.
SimValueColorRenderer	Provides coloring of table cells that show the similarity value. Color is based on both the similarity value and the requirements status (e.g. selected, linked, etc.)
SwingExtensions	Provides common methods for GUI manipulation
RsSelectionListener	Handles the action of selecting a row in a table.
RsDoubleClickListener	Handles the action for the Link button in the result table.
LinkButtonListener	Used to synchronize the selected row in the result table with the pressed Link/Unlink button.

Class/Interface	Description
SearchTermSubmitListener	Handles the action of pressing Enter in the search term box above the result table.
searchTermsOnlyItemListener	Handles the action of changing the state of the check box used to indicate whether only the specified search terms shall be used to find similar requirements or if both the selected requirement and the search terms shall be used.
LinkAction	Handles the action for the Link/Unlink buttons.
ConnectAction	Handles the action for the Connect button in the requirements sets tabs in the settings dialog.
ConnectLinksAction	Handles the action for the connect button in the links tab in the settings dialog.
SaveSettingsAction	Handles the action for the Save button in the settings dialog.
ResetAction	Handles the action for the Reset button in the settings dialog.
FetchFieldsForTableAction	Handles the action when a table is selected in the combo box in the settings dialog.
FetchFieldsForLinksTableAction	Handles the action when a table is selected in the combo box in the settings dialog's links tab.

The following classes and interfaces are part of the Java2 language and description of their usage may be found in official Java documentation.

Class/Interface
ListSelectionListener
MouseAdapter
KeyListener
ActionListener
AbstractAction

Context diagram

Not applicable.

Variability guide

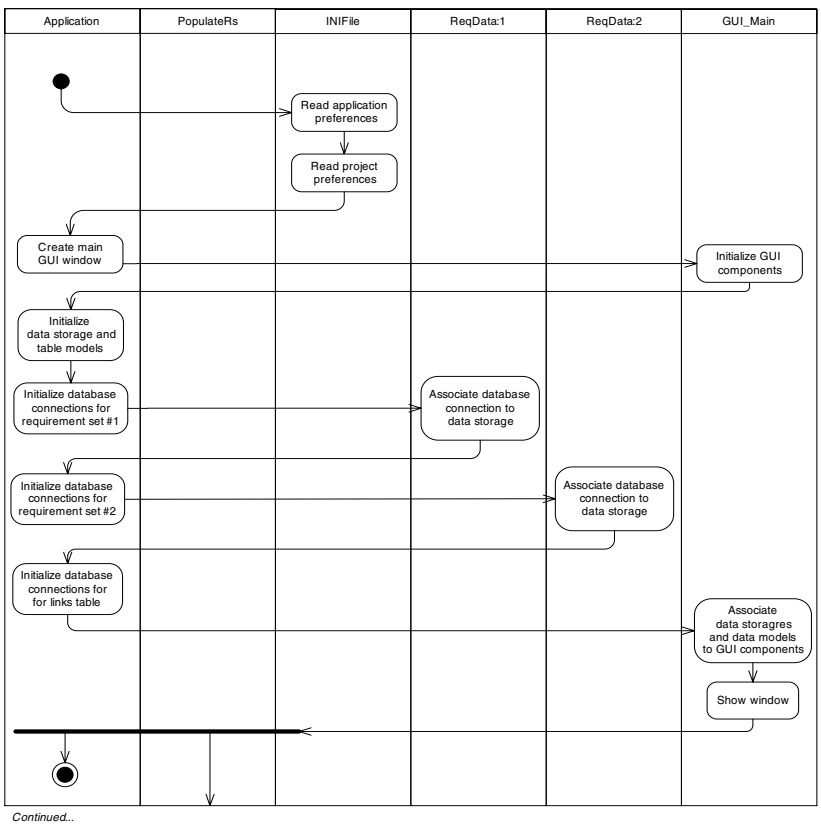
It would be a nice if it was possible to design a middle layer where generic logic for setting and reading GUI component properties are put. The lowest GUI layer would then "simply" be a specification file of the user interface, read by the middle layer. The feasibility, benefits, and drawbacks of such an architectural change must be further investigated.

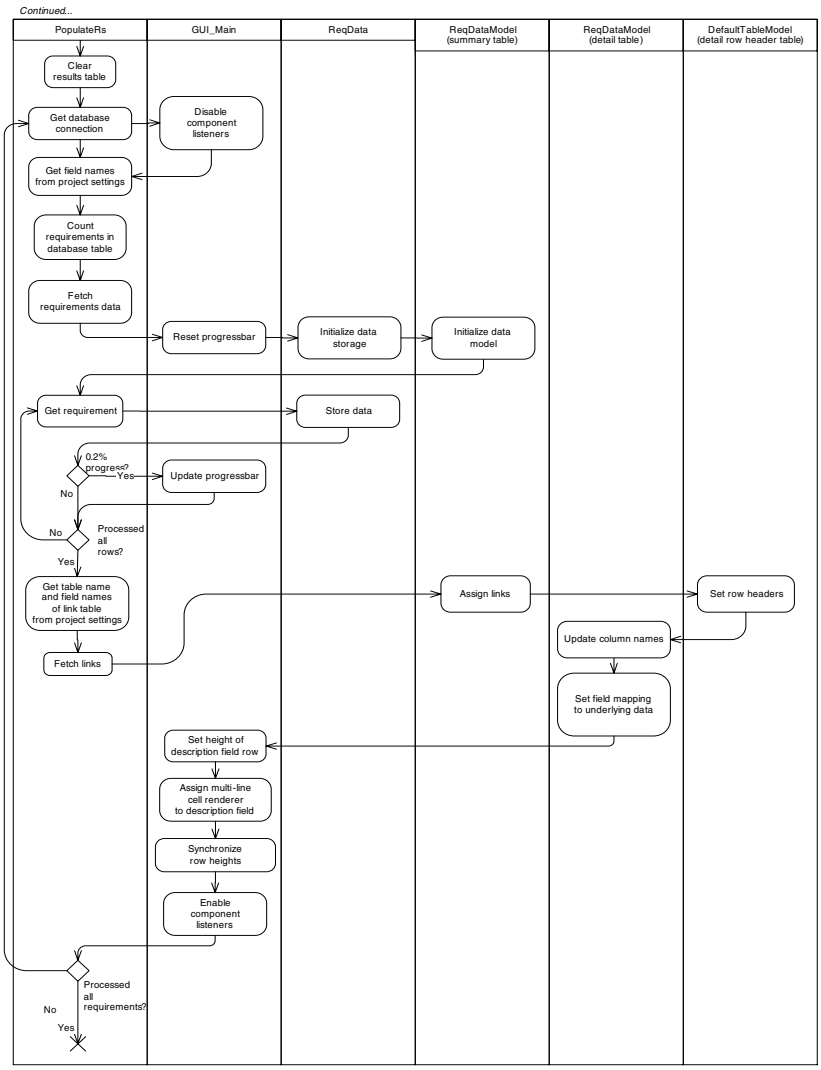
Architecture background

In general, GUI architectures easily become complex. This class has gone through several refinements and been subject for many additions during the analysis and implementation stage. The model has been updated continuously and its current form may call for a division into smaller, less complex classes.

6.7 View: ReqSimile Startup

The following activity diagram shows the execution behavior at startup. In short, the settings file is read and if a project file is specified the specified data sources are read and the user interface tables are populated.





Element catalog

The elements in this catalog are a selection of the classes described in the following model views.

Class/Interface	Model view
Application	Section 6.3.
PopulateRs	Section 6.4.
INIFile	Section 6.3.

Class/Interface	Model view
ReqData	Section 6.4.
ReqDataModel	Section 6.4.
GUI_Main	Section 6.6.
DefaultTableModel	Standard Java Swing class.

Context diagram

Not applicable.

Variability guide

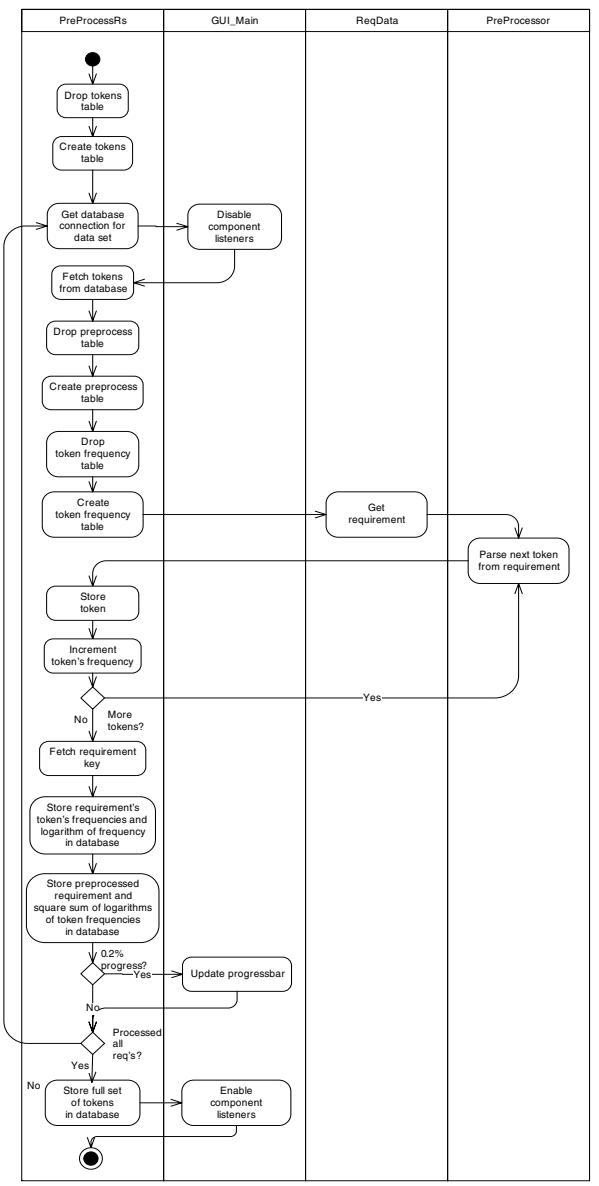
No open questions.

Architecture background

The decisions for the activity flow has emerged during implementation. The overall package structuring has guided the separation of logic to different parts of the program.

6.8 View: ReqSimile Preprocess requirements

The following activity diagram shows the execution behavior for preprocessing requirements (that previously have been fetched from from an external database). In short, all requirements are preprocessed (tokenization, stemming, and stop word removal) are stored in an external database together with tokens, token frequencies and pre-calculations for the cosine measure.



Element catalog

The elements in this catalog are a selection of the classes described in the following model views.

Class/Interface	Model view
GUI_Main	Section 6.6.
ReqData	Section 6.4.
PreProcessor	Section 6.4.
PreprocessRs	Section 6.4.

Context diagram

Not applicable.

Variability guide

ReqSimile is a stand-alone application. It is an open question to what extent it would be possible to fully integrate it with requirements management tools in order to perform continuous preprocessing when requirements are change. This would require the possibility to receive and send events between ReqSimile and the requirements management tool.

Anyhow, if the techniques are implemented into a requirements management tool altogether, it could be possible to have a background process to keep similarity values between requirements up-to-date. The background process would for example be notified by changes of relevant contents of the requirements.

Architecture background

The preprocessing procedure was based on an early design decision for speeding up similarity calculations during the main activity (of finding similar requirements by the click of a button).

The similarity calculations are separated into two stages: the preprocessing stage and the final calculation stage (the latter which is based on the currently selected requirement).

The procedure is balanced so that each stage takes minimal time. Calculation of the full similarity matrix would speed up the fetching of similar

requirements tremendously, but the preprocessing would then take extremely long time.

The preprocessing step produces tables that are used to quickly fetch similar requirements (see Section 6.9). The tables and their structures are as follows:

rsPreProcessedA (A is the requirements set)

Field	Description
ReqKey	Unique numeric identifier.
Tokens	Textual representation of the preprocessed requirement.
SqSum	Square sum of requirement tokens' weights (this is the term in the denominator of the formula for the Cosine similarity measure).

rsTokens

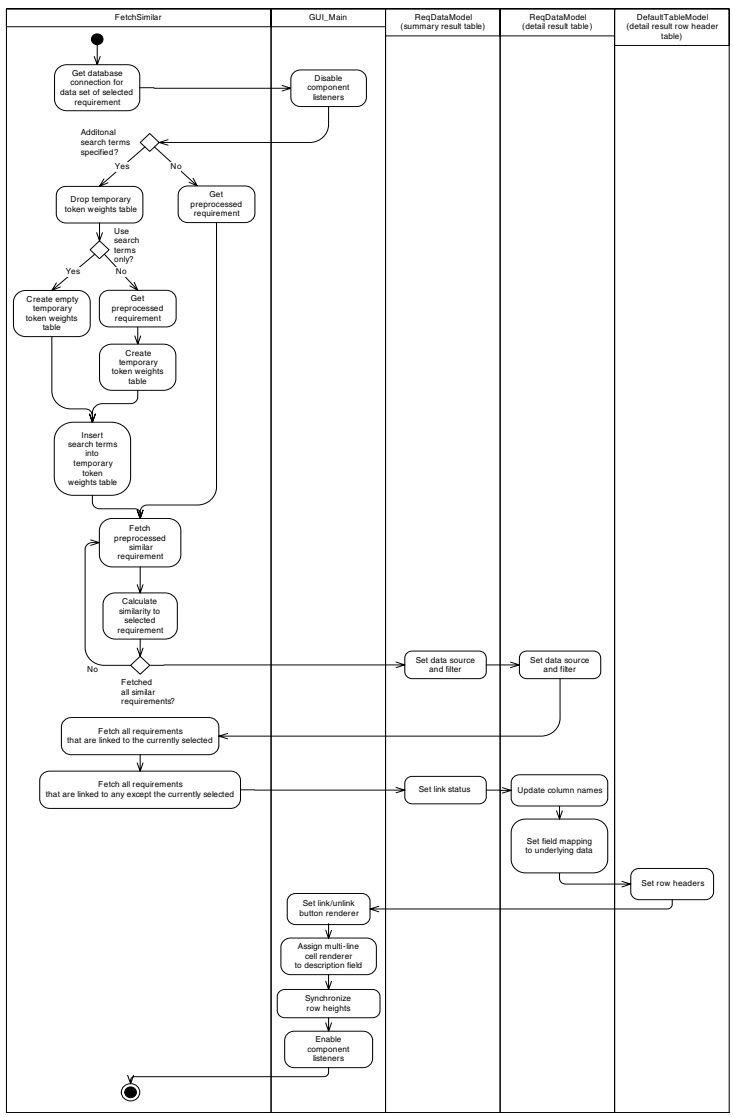
Field	Description
TokenKey	Unique numeric identifier of token.
Token	Textual representation of the preprocessed token.

rsTokenWeightsA (A is the requirements set)

Field	Description
ReqKey	Unique numeric identifier of requirement.
TokenKey	Unique numeric identifier of token.
Weight	The weight (e.g. $1 + log_2(frequency)$) of the token 'TokenKey' in the requirement 'ReqKey'.

6.9 View: ReqSimile Fetch requirements

The following activity diagram shows the execution behavior when the user selects a requirement or when the user submits search terms. In short, the preprocessed selected requirement (if any) is fetched and the search terms are preprocessed. The result is used to fetch all similar requirements and calculate the similarity value. The currently set links are then fetch from the database to be able to reflect the link status in the graphical user interface.



Element catalog

The elements in this catalog are a selection of the classes described in the following model views.

Class/Interface	Model view
FetchSimilar	Section 6.4.
ReqDataModel	Section 6.4.
GUI_Main	Section 6.6.

Context diagram

Not applicable.

Variability guide

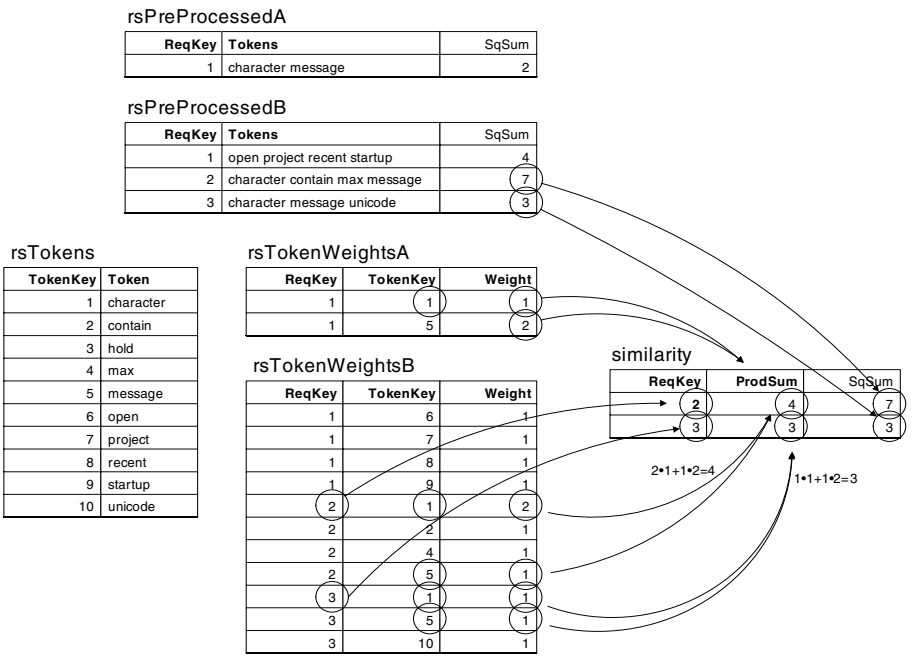
No open questions.

Architecture background

The fetch procedure was based on the same early design decisions as described in the architecture background for the preprocessing procedure (See Section 6.8.

The calculation of the similarity value between the requirements is partly left to the database engine used for storing the preprocessing tables. The fetching of similar requirement is made simultaneously with the calculation of the product sum in the nominator of the formula for the Cosine similarity measure (Natt och Dag et al., 2004). In a pre-study it was found that this can be performed through one single database query statement.

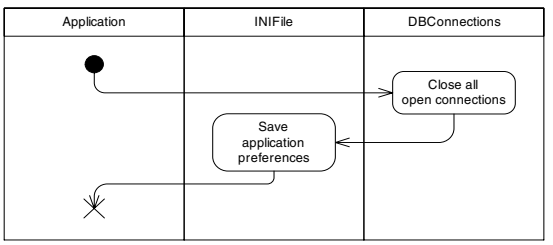
The example below illustrates the procedure. It is assumed that the user has selected requirement #1 from requirements set A (normally, there would many more requirements in set A). The similarity is calculated by matching the TokenKeys. If there are no TokenKeys in common, then the requirement will not end up in the resulting table. The product sum is calculated by multiplying the token weights for the tokens the requirements have in common, and adding the products together.



The last calculation step, for reaching a final similarity, is done by looping through the resulting table. This is done internally, and the result is not stored in any database.

6.10 View: ReqSimile Shutdown

The following activity diagram shows the execution behavior when ReqSimile is closed down. In short, database connections are closed and the applications preferences are saved.



Element catalog

The elements in this catalog are a selection of the classes described in the following model views.

Class/Interface	Model view
Application	Section 6.3.
INIFile	Section 6.3.
DBConnections	Section 6.4.

Context diagram

Not applicable.

Variability guide

To preserve application settings if system crashes, applications preferences could be saved each time they are changed, instead of when the application is closed.

Architecture background

None.

References

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., and Stafford, J. (2002). *Documenting Software Architectures: Views and Beyond*. The SEI Series in Software Engineering. Addison-Wesley, Boston, MA.

Natt och Dag, J. (2004). Reqsimile home page. http://reqsimile. sourceforge. org.

Natt och Dag, J. and Gervasi, V. (2005). Managing large repositories of natural language requirements. In Aurum, A. and Wohlin, C., editors, *Engineering and Managing Software Requirements*. Springer-Verlag.

Natt och Dag, J., Gervasi, V., Brinkkemper, S., and Regnell, B. (2004). Speeding up requirements management in a product software company: Linking customer whishes to product requirements through linguistic engineering. In *Proceedings of the International Requirements Engineering Conference (RE2004)*, pages 283–294, Kyoto, Japan. IEEE CS.

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

Natt och Dag, J., Regnell, B., Carlshamre, P., Andersson, M., and Karlsson, J. (2002). A feasibility study of automated natural language requirements analysis in market-driven development. *Requirements Engineering*, 7(1):20–33.

Reports on Communication Systems

101. **On Overload Control of SPC-systems**
Ulf Körner, Bengt Wallström, and Christian Nyberg, 1989.
CODEN: LUTEDX/TETS- -7133- -SE+80P

102. **Two Short Papers on Overload Control of Switching Nodes**
Christian Nyberg, Ulf Körner, and Bengt Wallström, 1990.
ISRN LUTEDX/TETS- -1010- -SE+32P

103. **Priorities in Circuit Switched Networks**
Åke Arvidsson, Ph.D. thesis, 1990.
ISRN LUTEDX/TETS- -1011- -SE+282P

104. **Estimations of Software Fault Content for Telecommunication Systems**
Bo Lennselius, Lic. thesis, 1990.
ISRN LUTEDX/TETS- -1012- -SE+76P

105. **Reusability of Software in Telecommunication Systems**
Anders Sixtensson, Lic. thesis, 1990.
ISRN LUTEDX/TETS- -1013- -SE+90P

106. **Software Reliability and Performance Modelling for Telecommunication Systems**
Claes Wohlin, Ph.D. thesis, 1991.
ISRN LUTEDX/TETS- -1014- -SE+288P

107. **Service Protection and Overflow in Circuit Switched Networks**
Lars Reneby, Ph.D. thesis, 1991.
ISRN LUTEDX/TETS- -1015- -SE+200P

108. **Queueing Models of the Window Flow Control Mechanism**
Lars Falk, Lic. thesis, 1991.
ISRN LUTEDX/TETS- -1016- -SE+78P

109. **On Efficiency and Optimality in Overload Control of SPC Systems**
Tobias Rydén, Lic. thesis, 1991.
ISRN LUTEDX/TETS- -1017- -SE+48P

110. **Enhancements of Communication Resources**
Johan M Karlsson, Ph.D. thesis, 1992.
ISRN LUTEDX/TETS- -1018- -SE+132P

111. **On Overload Control in Telecommunication Systems**
Christian Nyberg, Ph.D. thesis, 1992.
ISRN LUTEDX/TETS- -1019- -SE+140P

112. **Black Box Specification Language for Software Systems**
Henrik Cosmo, Lic. thesis, 1994.
ISRN LUTEDX/TETS- -1020- -SE+104P

113. **Queueing Models of Window Flow Control and DQDB Analysis**
Lars Falk, Ph.D. thesis, 1995.
ISRN LUTEDX/TETS- -1021- -SE+145P

114. **End to End Transport Protocols over ATM**
Thomas Holmström, Lic. thesis, 1995.
ISRN LUTEDX/TETS- -1022- -SE+76P

115. **An Efficient Analysis of Service Interactions in Telecommunications**
Kristoffer Kimbler, Lic. thesis, 1995.
ISRN LUTEDX/TETS- -1023- -SE+90P

116. **Usage Specifications for Certification of Software Reliability**
Per Runeson, Lic. thesis, May 1996.
ISRN LUTEDX/TETS- -1024- -SE+136P

117. **Achieving an Early Software Reliability Estimate**
Anders Wesslén, Lic. thesis, May 1996.
ISRN LUTEDX/TETS- -1025- -SE+142P

118. **On Overload Control in Intelligent Networks**
Maria Kihl, Lic. thesis, June 1996.
ISRN LUTEDX/TETS- -1026- -SE+80P

119. **Overload Control in Distributed-Memory Systems**
Ulf Ahlfors, Lic. thesis, June 1996.
ISRN LUTEDX/TETS- -1027- -SE+120P

120. **Hierarchical Use Case Modelling for Requirements Engineering**
Björn Regnell, Lic. thesis, September 1996.
ISRN LUTEDX/TETS- -1028- -SE+178P

121. **Performance Analysis and Optimization via Simulation**
Anders Svensson, Ph.D. thesis, September 1996.
ISRN LUTEDX/TETS- -1029- -SE+96P

122. **On Network Oriented Overload Control in Intelligent Networks**
Lars Angelin, Lic. thesis, October 1996.
ISRN LUTEDX/TETS- -1030- -SE+130P

123. **Network Oriented Load Control in Intelligent Networks Based on Optimal Decisions**
Stefan Pettersson, Lic. thesis, October 1996.
ISRN LUTEDX/TETS- -1031- -SE+128P

124. **Impact Analysis in Software Process Improvement**
Martin Höst, Lic. thesis, December 1996.
ISRN LUTEDX/TETS- -1032- -SE+140P

125. **Towards Local Certifiability in Software Design**
Peter Molin, Lic. thesis, February 1997.
ISRN LUTEDX/TETS- -1033- -SE+132P

126. **Models for Estimation of Software Faults and Failures in Inspection and Test**
Per Runeson, Ph.D. thesis, January 1998.
ISRN LUTEDX/TETS- -1034- -SE+222P

127. **Reactive Congestion Control in ATM Networks**
Per Johansson, Lic. thesis, January 1998.
ISRN LUTEDX/TETS- -1035- -SE+138P

128. **Switch Performance and Mobility Aspects in ATM Networks**
Daniel Søbirk, Lic. thesis, June 1998.
ISRN LUTEDX/TETS- -1036- -SE+91P

129. **VPC Management in ATM Networks**
Sven-Olof Larsson, Lic. thesis, June 1998.
ISRN LUTEDX/TETS- -1037- -SE+65P

130. **On TCP/IP Traffic Modeling**
Pär Karlsson, Lic. thesis, February 1999.
ISRN LUTEDX/TETS- -1038- -SE+94P

131. **Overload Control Strategies for Distributed Communication Networks**
Maria Kihl, Ph.D. thesis, March 1999.
ISRN LUTEDX/TETS- -1039- -SE+158P

132. **Requirements Engineering with Use Cases - a Basis for Software Development**
Björn Regnell, Ph.D. thesis, April 1999.
ISRN LUTEDX/TETS- -1040- -SE+225P

133. **Utilisation of Historical Data for Controlling and Improving Software Development**
Magnus C. Ohlsson, Lic. thesis, May 1999.
ISRN LUTEDX/TETS- -1041- -SE+146P

134. **Early Evaluation of Software Process Change Proposals**
Martin Höst, Ph.D. thesis, June 1999.
ISRN LUTEDX/TETS- -1042- -SE+193P

135. **Improving Software Quality through Understanding and Early Estimations**
Anders Wesslén, Ph.D. thesis, June 1999.

ISRN LUTEDX/TETS- -1043- -SE+242P

136. **Performance Analysis of Bluetooth**
Niklas Johansson, Lic. thesis, March 2000.
ISRN LUTEDX/TETS- -1044- -SE+76P

137. **Controlling Software Quality through Inspections and Fault Content Estimations**
Thomas Thelin, Lic. thesis, May 2000
ISRN LUTEDX/TETS- -1045- -SE+146P

138. **On Fault Content Estimations Applied to Software Inspections and Testing**
Håkan Petersson, Lic. thesis, May 2000.
ISRN LUTEDX/TETS- -1046- -SE+144P

139. **Modeling and Evaluation of Internet Applications**
Ajit K. Jena, Lic. thesis, June 2000.
ISRN LUTEDX/TETS- -1047- -SE+121P

140. **Dynamic traffic Control in Multiservice Networks - Applications of Decision Models**
Ulf Ahlfors, Ph.D. thesis, October 2000.
ISRN LUTEDX/TETS- -1048- -SE+183P

141. **ATM Networks Performance - Charging and Wireless Protocols**
Torgny Holmberg, Lic. thesis, October 2000.
ISRN LUTEDX/TETS- -1049- -SE+104P

142. **Improving Product Quality through Effective Validation Methods**
Tomas Berling, Lic. thesis, December 2000.
ISRN LUTEDX/TETS- -1050- -SE+136P

143. **Controlling Fault-Prone Components for Software Evolution**
Magnus C. Ohlsson, Ph.D. thesis, June 2001.
ISRN LUTEDX/TETS- -1051- -SE+218P

144. **Performance of Distributed Information Systems**
Niklas Widell, Lic. thesis, February 2002.
ISRN LUTEDX/TETS- -1052- -SE+78P

145. **Quality Improvement in Software Platform Development**
Enrico Johansson, Lic. thesis, April 2002.
ISRN LUTEDX/TETS- -1053- -SE+112P

146. **Elicitation and Management of User Requirements in Market-Driven Software Development**
Johan Natt och Dag, Lic. thesis, June 2002.
ISRN LUTEDX/TETS- -1054- -SE+158P

147. **Supporting Software Inspections through Fault Content Estimation and Effectiveness Analysis**
Håkan Petersson, Ph.D. thesis, September 2002.
ISRN LUTEDX/TETS- -1055- -SE+237P

148. **Empirical Evaluations of Usage-Based Reading and Fault Content Estimation for Software Inspections**
Thomas Thelin, Ph.D. thesis, September 2002.
ISRN LUTEDX/TETS- -1056- -SE+210P

149. **Software Information Management in Requirements and Test Documentation**
Thomas Olsson, Lic. thesis, October 2002.
ISRN LUTEDX/TETS- -1057- -SE+122P

150. **Increasing Involvement and Acceptance in Software Process Improvement**
Daniel Karlström, Lic. thesis, November 2002.
ISRN LUTEDX/TETS- -1058- -SE+125P

151. **Changes to Processes and Architectures; Suggested, Implemented and Analyzed from a Project viewpoint**
Josef Nedstam, Lic. thesis, November 2002.
ISRN LUTEDX/TETS- -1059- -SE+124P

152. **Resource Management in Cellular Networks -Handover Prioritization and Load Balancing Procedures**
Roland Zander, Lic. thesis, March 2003.
ISRN LUTEDX/TETS- -1060- -SE+120P

153. **On Optimisation of Fair and Robust Backbone Networks**
Pål Nilsson, Lic. thesis, October 2003.
ISRN LUTEDX/TETS- -1061- -SE+116P

154. **Exploring the Software Verification and Validation Process with Focus on Efficient Fault Detection**
Carina Andersson, Lic. thesis, November 2003.
ISRN LUTEDX/TETS- -1062- -SE+134P

155. **Improving Requirements Selection Quality in Market-Driven Software Development**
Lena Karlsson, Lic. thesis, November 2003.
ISRN LUTEDX/TETS- -1063- -SE+132P

156. **Fair Scheduling and Resource Allocation in Packet Based Radio Access Networks**
Torgny Holmberg, Ph.D. thesis, November 2003.
ISRN LUTEDX/TETS- -1064- -SE+187P

157. **Increasing Product Quality by Verification and Validation Improvements in an Industrial Setting**
Tomas Berling, Ph.D. thesis, December 2003.
ISRN LUTEDX/TETS- -1065- -SE+208P

158. **Some Topics in Web Performance Analysis**
Jianhua Cao, Lic. thesis, June 2004.
ISRN LUTEDX/TETS- -1066- -SE+99P

159. **Overload Control and Performance Evaluation in a Parlay/OSA Environment**
Jens K. Andersson, Lic. thesis, August 2004.
ISRN LUTEDX/TETS- -1067- -SE+100P

160. **Performance Modeling and Control of Web Servers**
Mikael Andersson, Lic. thesis, September 2004.
ISRN LUTEDX/TETS- -1068- -SE+105P

161. **Integrating Management and Engineering Processes**
Daniel Karlström, PhD. thesis, December 2004.
ISRN LUTEDX/TETS- -1069- -SE+230P

162. **Managing natural language requirements in large-scale software development**
Johan Natt och Dag, PhD. thesis, February 2005.
ISRN LUTEDX/TETS- -1070- -SE+200P