Blekinge Institute of Technology
Licentiate Series No. 2005:04
ISBN 91-7295-059-5
ISSN 1650-2140

2005-04-21

Impact Analysis

Organisational Views and Support Techniques

Per Jönsson

Department of Systems and Software Engineering
School of Engineering
Blekinge Institute of Technology
Sweden

Blekinge Institute of Technology
Licentiate Series No. 2005:04

ISBN 91-7295-059-5
ISSN 1650-2140

Printed in Sweden
Kaserntryckeriet AB, Karlskrona 2005

To Madeleine

This thesis is submitted to the Faculty of Technology at Blekinge Institute of Technology, in partial fulfillment of the requirements for the degree of Licentiate of Technology in Software Engineering.

Contact Information

Per Jönsson
Department of Systems and Software Engineering
School of Engineering
Blekinge Institute of Technology
PO Box 520
SE-372 25 Ronneby
SWEDEN

E-mail: per.jonsson@bth.se
Web: http://www.ipd.bth.se/pjn

Abstract

Change is unavoidable in software development. During the entire lifecycle of a product, from concept to retirement, the environment changes; the needs of customers or the market change and grow, and with them the requirements on the system being developed. Under these conditions, it is crucial to have strong change control in order to be able to manage change in an orderly fashion. Unmanaged change may lead to fault-prone software, thereby increasing test, support and maintenance costs.

Impact analysis is the activity of analysing a change and assessing the consequences it may have, including necessary modifications to development artefacts. Thus, it serves as a very important change control tool. Furthermore, as the consequences may include aspects of time, resources, market and technology, impact analysis has the potential to be a valuable product and project management tool.

To this date, impact analysis research has mainly been conducted in the software maintenance field. However, as impact analysis clearly has a wide field of application, it is relevant to study it in other contexts as well. This thesis looks at impact analysis from a requirements engineering perspective, with particular focus on organisational aspects related to different roles and organisational levels. The results show that impact analysis indeed has a diverse nature with respect to these aspects. Furthermore, the thesis includes the proposal and evaluation of a semi-automatic method for performing impact analysis. Finally, it provides a thorough evaluation of a technique for reconstructing missing data in surveys.

Acknowledgements

First and foremost, I am greatly indebted to my main advisor *Claes Wohlin*, for endless support and numerous hours of discussion about all those problems that seemed so simple from the beginning. I am also grateful to my secondary advisor *Michael Mattsson*, for making me consider Ph.D. studies in the first place and for asking the difficult, philosophical questions.

The research environment at Blekinge Institute of Technology is both inspiring and challenging. I wish to thank colleagues in the BESQ project and in the SERL research group for creating this environment. In particular, I want to thank *Patrik Berander* for contributing good ideas and reviewing papers (including this thesis).

A special thanks goes to *Mikael Lindvall*, Fraunhofer USA, for fruitful discussions and the cooperation in writing Chapter 2 of this thesis.

I want to thank all the people at *Ericsson*, my industrial research partner, who have participated in my studies and thereby contributed to my research. The cooperation with Ericsson has resulted in many interesting research ideas and results. I especially appreciate the help from *Anna Eriksson*, *Lars Angelin*, *Johan Häggman*, *Bengt Romberg* and *Helena Olá*.

Finally, I would like to thank my family for continuous support through the years. My mother *Barbro* and my father *Jan*, for teaching me to pose questions and to seek answers. My brother *Ola*, for always being there for me. Last, *Madeleine*, for understanding and for putting up with my odd working hours. You're the best!

This work was partly funded by the *Knowledge Foundation* in Sweden under a research grant for the project "Blekinge – Engineering Software Qualities (BESQ)" (http://www.bth.se/besq).

Table of Contents

List of Figures ix

List of Tables xi

Chapter 1 1
Introduction

1.1 Research Setting 3
1.2 Impact Analysis. 4
1.2.1 Definitions 4
1.2.2 Perspective 5
1.2.3 Organisational vs. Technical 7
1.2.4 Uses 8
1.2.5 In Industry 13
1.3 Thesis Papers 14
1.3.1 Paper Abstracts 15
1.3.2 Thesis Outline 18
1.4 Research Methodology. 19
1.4.1 Research 19
1.4.2 Empirical Research 20
1.4.3 Research Taxonomies 21
1.4.4 Classification of Thesis Studies. 24
1.4.5 Research Validity 26
1.5 Contribution 29
1.5.1 Thesis 29
1.5.2 Chapters 30
1.6 Future Work 31
1.6.1 Semi-Automatic Impact Analysis 32
1.6.2 Technical Support for Organisational IA. 32
1.6.3 Practitioners' Impact Analysis Ability. 32
1.7 Summary 33

Chapter 2 35
Impact Analysis

2.1 Background. 37
2.1.1 Concepts and Terms 37
2.1.2 Software Change and Impact Analysis 39
2.1.3 History and Trends 42
2.2 Strategies for Impact Analysis 44

2.2.1 Automatable Strategies 45
2.2.2 Manual Strategies 49
2.3 Non-Functional Requirements 52
2.4 Impact Analysis Metrics. 53
2.4.1 Metrics for Quantifying Change Impact 54
2.4.2 Metrics for Evaluation of Impact Analysis 55
2.5 Tool Support 60
2.6 Future of Impact Analysis 62
2.7 Summary. 63

Chapter 3. 65
Understanding the Importance of Roles in Architecture-Related Process Improvement – A Case Study

3.1 Background and Research Setting. 66
3.1.1 Architecture-Related Process Improvement 67
3.2 Related Work 68
3.3 Design. 69
3.3.1 Questionnaire Design 69
3.3.2 Sampling and Response Rate 70
3.3.3 Treatment of Missing Data. 71
3.3.4 Roles 72
3.3.5 Validity Threats 73
3.4 Results. 75
3.4.1 Infrastructure Questions. 75
3.4.2 Improvement Question 77
3.5 Analysis. 78
3.5.1 Infrastructure Questions. 78
3.5.2 Improvement Question 79
3.6 Discussion 79
3.7 Conclusions 81

Chapter 4. 83
Understanding Impact Analysis: An Empirical Study to Capture Knowledge on Different Organisational Levels

4.1 Related Work 84
4.2 Method 85
4.2.1 Research Setting 85
4.2.2 Organisational Levels 86
4.2.3 Interview Design. 86
4.2.4 Results Triangulation and Filtering 87
4.2.5 Prioritisation 87
4.3 Operation 88
4.3.1 Organisational Levels 88

4.3.2 Interviews 89
4.3.3 Prioritisation 90
4.4 Results 90
4.4.1 Threats to Validity 91
4.5 Analysis and Discussion 92
4.5.1 Qualitative Analysis 92
4.5.2 Quantitative Analysis 93
4.5.3 Discussion 94
4.6 Conclusions 95

Chapter 5 99
Semi-Automatic Impact Analysis through Keyword-Based Relationships – A Feasibility Study

5.1 Usage Scenarios 101
5.2 Related Work 103
5.3 Method 104
5.3.1 Natural vs. Technical Language 104
5.3.2 Fuzzy String Matching 105
5.3.3 Latent Semantic Indexing 106
5.3.4 Method Introduction 108
5.3.5 Method Steps 109
5.4 Evaluation 114
5.4.1 Evaluation Context 114
5.4.2 Tools Used 115
5.4.3 Step 1: Screen for Relevance 115
5.4.4 Step 2: Identify Keywords 115
5.4.5 Step 3: Identify Dependencies Using LSI 117
5.4.6 Step 4: Examine Results and Estimate Impact 117
5.5 Analysis 118
5.6 Discussion and Future Work 121
5.7 Summary 123

Chapter 6 125
Benchmarking k-Nearest Neighbour Imputation With Homogeneous Likert Data

6.1 Related Work 127
6.2 Research Data 129
6.2.1 Evaluation Data 129
6.2.2 Missing Data 131
6.3 Imputation Methods 131
6.3.1 Random Draw Substitution 132
6.3.2 Random Imputation 132
6.3.3 Median Imputation 133
6.3.4 Mode Imputation 133

6.3.5 k-Nearest Neighbour 134
6.4 Evaluation Process 136
6.4.1 Data Removal - Step 1 137
6.4.2 Imputation - Step 2 139
6.4.3 Evaluation - Step 3 140
6.5 Simulation 141
6.5.1 Parameters 141
6.5.2 Software 142
6.5.3 Process Reuse 143
6.6 Results 143
6.6.1 Incomplete Data Sets 144
6.6.2 Comparison of k-Values and Strategies 145
6.6.3 Comparison of Attribute Counts 148
6.6.4 Comparison of Percentages 150
6.6.5 Benchmarking 150
6.6.6 Summary and Interpretation of the Results 152
6.7 Validity and Future Work 155
6.7.1 Threats to Validity 155
6.7.2 Future Work 156
6.8 Conclusions 157

References 159

List of Figures

1.1 Change Process at Ericsson 3
1.2 Importance of Impact Analysis Uses 9
2.1 Software Lifecycle Objects (SLOs) Affected (*right*) Due to Requirements Changes in Different Phases (*left*) 37
2.2 Change Management Process Framework 41
2.3 Three Views of the Relationships Among SLOs 47
2.4 Measuring Impact Using Metrics 54
2.5 Tree of Impact Analysis Metrics 56
3.1 Infrastructure Questions – Answer Distribution (*y axis*) per Role (*x axis*) 76
4.1 Box Plots for Issues i4 and i19, Individual Perspective 95
5.1 Average Query-Document Similarities 119
6.1 Distribution of Response Options 130
6.2 Evaluation Process Outline. 137
6.3 Performance at 14.5% and 19.0% Missing Data, CC and IC 145
6.4 9.9% Missing Data, 12 and 18 Attributes, IC 149
6.5 Ability vs. Amount of Missing Data 150

List of Tables

1.1 Requirements Engineering Search Queries 6
1.2 Number of RE and SM Articles. 6
1.3 Classification of Thesis Studies 26
2.1 Impact Factors. 55
2.2 Predicted vs. Actual Changes 58
3.1 Most Important Improvement – Answer Distribution Among Roles 77
3.2 Kruskal-Wallis (*left*) and Chi-Square (*right*) Outcome, All Questions 78
3.3 Significant Role-Purpose Pairs 79
4.1 Top Five Placements for Issues and Uses 93
4.2 Top Five vs. Bottom Five Issues. 93
4.3 Kruskal-Wallis for Issues i4 and i19 94
4.1 All Issues 96
4.2 All Uses 97
5.1 Keyword Classes Used in the Evaluation. 116
5.2 Recall and Precision for the Evaluation 120
6.1 Example Incomplete Data Set 135
6.2 Overview of Incomplete Data Sets 144
6.3 Optimal k-Values with R, Q and MSE for the CC Strategy 147
6.4 Optimal k-Values with R, Q and MSE for the IC Strategy. 147
6.5 Optimal k vs. Calculated k 148
6.6 Results Overview. 153
6.7 Benchmarking Overview 153

CHAPTER 1

Introduction

Software development is a complex engineering activity with the ultimate goal of transforming stated and implied needs into a finished product. Needs of customers, users or the market are captured as requirements that subsequently govern development tasks such as architecture, design, implementation and testing. In an utopian view of software development, requirements are fully known and perfectly understood prior to analysis and design, and development tasks are simply a matter of doing the right things. In the real world, understanding, insights and general know-how evolve progressively as development proceeds and the product begins to shape. The underlying needs behind requirements can be problematic to grasp, time plans may be overly optimistic, and customers and users may change their minds about what they want. These factors create an environment in which requirements cannot be regarded as fixed entities. Instead, they may change unexpectedly at any time, thereby disturbing normal development work.

The ever-changing environment in which development takes place calls for strong change control. Uncontrolled changes will inevitably lead to software deterioration; if a change is not well-understood, it may introduce faults in the software and thereby destabilise the system. *Change impact analysis* denotes the activity of identifying and understanding the scope of modifications necessary in response to a change. Thus, it is an important tool to support change control and prevent software deterioration.

In software evolution, when a system is developed over several subsequent releases, new requirements can be placed on a level with changing requirements, in that they also result in changes being made to already implemented code. Thus, when a new release of a system is developed, it is crucial to perform impact analysis for both requirements changes and new requirements. If follows that the term “change impact analysis” is too narrow to capture this duality. Consequently, this thesis uses only “impact analysis” to reflect the wider application of the activity.

The outcome of impact analysis is typically a set of development artefacts affected by a proposed change or a new requirement. Affected artefacts are not only code modules, classes and functions, but also test cases, design models, user manuals and so on. Knowledge about affected artefacts of different kinds can be used to update time plans, re-allocate resources, revise product plans, modify test strategies and so on. Thus, impact analysis has, if used properly, the potential to be a very powerful project and product management tool, and deserves as such to be studied in a wide organisational context.

This thesis supports understanding of impact analysis from a requirements engineering perspective in an organisational context. It starts by providing an extensive look at the role of impact analysis in requirements engineering. It then goes on to present organisational views of impact analysis, both in software process improvement and with a more direct focus on associated uses and issues. Next, it describes an initial exploration of a semi-automatic method for supporting the impact analysis activity. Finally, as an exception to the main topic, the last chapter presents a study of one solution to a problem many researchers are faced with – missing data.

The remainder of this chapter is structured as follows. Section 1.1 begins by introducing the main research setting of the thesis studies. Next, Section 1.2 presents some key aspects of impact analysis as it is seen in the thesis. Section 1.3 lists the papers that are included in the thesis and relates them to the research setting. Section 1.4 describes the different research methodologies employed in the thesis, and classifies the included papers according to two research taxonomies. The contribution of the thesis and of the individual chapters is presented in Section 1.5. Finally, future work is discussed in Section 1.6, and a summary is given in Section 1.7.

1.1 Research Setting

Of the five studies presented in the thesis, all except the one described in Chapter 2 are directly or indirectly related to Ericsson AB, Sweden (hereafter referred to only as *Ericsson*). In the two studies described in Chapters 3 and 4, developers from Ericsson participated as study subjects, whereas the two studies described in Chapters 5 and 6 were performed in a laboratory context, but used data collected at Ericsson.

Ericsson develops, among other things, solutions for charging and positioning in mobile phone networks (for example, GSM networks). The company operates on a world market, and sells systems to many of the world's largest mobile phone operators. Due to the inherent complexity in the type of systems developed, projects are both long and large. A typical project length is between 12 and 18 months, and a project normally involves somewhere between 60 and 120 persons.

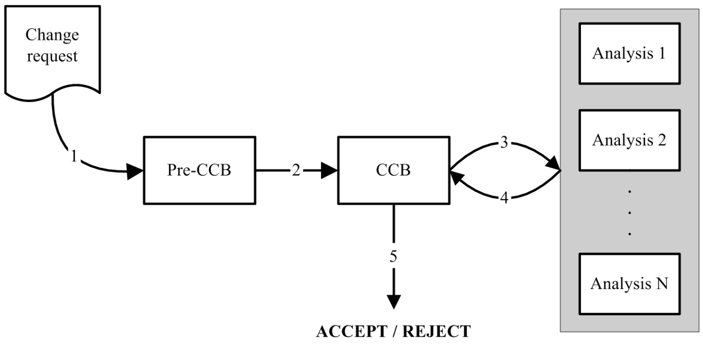


Figure 1.1 Change Process at Ericsson

Ericsson is very responsive when it comes to customers' demands, which manifests in a well-defined change process. A simplified view of the change process is shown in Figure 1.1 (the numbers in the figure show the flow described next). A change request is submitted from an internal source (such as a developer) or an external source (such as a customer forum). To alleviate the pressure in the Change Control Board (CCB), the change request is first screened in a pre-CCB, where a first technical analysis takes place. If the change request is unclear, it is communicated back to its source for clarification. The CCB subsequently decides who will analyse the change request and sets a suitable time frame. The change request is next subjected to a number of analyses, concerning everything from system impact to necessary adjustments of time plans and re-allocation of resources. Finally, the CCB considers all analyses and determines whether the change request should be accepted or rejected.

1.2 Impact Analysis

This section provides an introduction to impact analysis. Chapter 2 outlines the state-of-the-art of impact analysis within the requirements engineering field, while the character of this section is more of a general discourse of the various aspects of the subject. Nevertheless, there is a certain overlap between this section and Chapter 2.

1.2.1 Definitions

There are many definitions of impact analysis. In order to be clear about the context in which impact analysis is seen in this thesis, some definitions are presented and commented on below. The definitions are listed in chronological order.

Turver and Munro define impact analysis as "the assessment of a change, to the source code of a module, on the other modules of the system. It determines the scope of a change and provides a measure of its complexity." [117] A problem with this definition is that it has a very narrow scope; it is only concerned with changes to and effects on the source code. Consequently, it does not fit well in a requirements engineering context, where changes affect many different kinds of artefacts, not only source code. Even in a software maintenance context, there are typically other artefacts, such as designs and object models, that may be affected by a change.

Bohner and Arnold define impact analysis as "the activity of identifying the potential consequences, including side effects and ripple effects, of a change, or estimating what needs to be modified to accomplish a change before it has been made." [14] This definition is not geared towards any particular type of development artefact, and is as such appropriate to use in a requirements engineering context. The definitions of and distinction between ripple effect and side effect are clarified in Chapter 2.

Lindvall has an outspoken focus on impact analysis from a requirements perspective. He defines requirements-driven impact analysis as "the identification of the set of software entities that need to be changed to implement a new requirement in an existing system." [73] This definition is particularly interesting, as it concerns new requirements rather than changes to current requirements. As hinted in the definition, this type of impact analysis is useful when a system is developed incrementally over several releases.

Pfleeger defines impact analysis as "the evaluation of the many risks associated with the change, including estimates of effects on resources, effort and schedule." [89] This definition is also generic, but indicates, by explicitly mentioning risk and project control, a broad view of what is meant by "impact" and a strong focus on non-technical aspects of impact analysis.

Throughout this thesis, the definitions by Bohner and Arnold and Lindvall are used. This reflects the fact that both changing requirements and new requirements are seen as important causes for software change. Definitions of common terms associated with impact analysis can be found in Chapter 2.

1.2.2 Perspective

Impact analysis research typically belongs to the software maintenance field rather than the requirements engineering field. To illustrate this, a search was undertaken with the goal of counting the number of impact analysis articles in the respective field.

To find impact analysis articles, the phrase "impact analysis" was specified as a general search phrase to make it match against all possible article data (typically article title, abstract and keywords, but also full text if available).

To restrict the results to articles in the requirements engineering field, the phrase "requirements engineering" was specified as part of the publication title (i.e., journal, conference or equivalent). Similarly, to restrict results to articles in the software maintenance field, the phrase "software maintenance" was used. Of course, the title of the publication need not reflect the exact focus of an included article, but it should nevertheless reflect to some extent what the authors originally wished to communicate about the article.

Three databases were selected and used in the search: ACM Guide (a superset of the ACM Digital Library) [2], IEEE Xplore [57] and the combined Compendex & Inspec [39]. Table 1.1 shows the search queries for finding publications in the requirements engineering field for each database. In addition to specifying the queries, results were requested only from 1993 and forward, since 1993 was the year of the first International Symposium on Requirements Engineering; it would be an unfair comparison to consider earlier results (the first Conference on Software Maintenance, for example, was held already in 1985).

Table 1.1 Requirements Engineering Search Queries

Database	**Search query**
ACM Guide	+"impact analysis" +publication:"requirements engineering"
IEEE Xplore	"impact analysis" <and> ("requirements engineering" <in> jn)
Compendex & Inspec	{impact analysis} wn ALL AND {requirements engineering} wn CF

Table 1.2 shows the results of the search. The figures in the table do not represent the raw search matches, but rather the article counts after some irrelevant matches, such as entire conference proceedings, were discarded. It should be noted that the removed articles belonged only to the software maintenance field.

Table 1.2 Number of RE and SM Articles

Database	**RE**	**SM**
ACM Guide	2	36
IEEE Xplore	2	30
Compendex & Inspec	2	33
UNION	2	44

There is a large overlap among the three databases for both fields; the two requirements engineering articles (coming from one publication) are the same in all three databases, and the union of all software maintenance articles (coming from three publications) contains 44 articles. The large difference in article count between the fields indicates that software maintenance so far has been the predominant target field for impact analysis research. Given the initial argumentation for the relationship between impact analysis, software change and changing requirements (see also Chapter 2), it seems reasonable that impact analysis should be more visible in requirements engineering publications. There is a need both for exploring challenges for impact analysis in the requirements engineering field, and for investigating how current impact analysis methods and tools work in a requirements engineering context.

This emphasises the importance of the survey of impact analysis in requirements engineering presented in Chapter 2.

Note that the intention with this search was not to provide a complete overview of impact analysis articles within software maintenance and requirements engineering, nor was it to extract a statistically representative sample of said articles. As stated, it was merely to show that there to this date is a large imbalance between requirements engineering and software maintenance regarding impact analysis research.

It should also be mentioned that the search intentionally did not include variants of “impact analysis”, such as “analysis of impact” and “analyse change impact” (or, as used by Bratthall et al., “predict change impact“ [17]). It was argued that the choice of phrasing in an article would indicate whether the authors had a true impact analysis focus, or just mentioned it in passing.

1.2.3 Organisational vs. Technical

It is possible to look at impact analysis from two different perspectives: *organisational* and *technical*. The technical perspective, which by far is the most common one in current research (see, for example, [14]), concerns methods and tools for assessing the impact of changes on the system level. The organisational perspective, on the other hand, concerns studies with the goal of, for example, creating an understanding of how impact analysis is viewed and used by software practitioners.

A drawback with the purely technical perspective is that such research often assumes the presence of a detailed, structured and connected infrastructure of development artefacts (see, for example, [18] and [126]). However, this rules out impact analysis in early stages of development, for example when only a coarse architecture model exists. Furthermore, it is in contradiction with a problem commonly seen in industry: models, documentation and dependencies are not always created and updated unless there are mechanisms for doing it in simple and efficient ways (see also [25]).

The organisational perspective adds anchoring of impact analysis in the context where it ultimately has its place – software development organisations (consider, for example, Pfleeger’s definition with its clear organisational focus). Some reflections on what is necessary to pursue with respect to the organisational perspective are:

- A focus on understanding *why*, *what* and *when* about impact analysis in software development organisations. For example, what kind of impact analysis is performed? For what reasons are impact analysis performed? When is impact analysis performed? Early (project-wise) impact analysis is different from late impact analysis, as the available foundations (e.g., architecture, design models and source code) differ in both availability and maturity.
- The development of methods and tools that do not require a completely updated and accurate infrastructure of development artefacts. An excellent example is Egyed's method that addresses the problem of missing traceability by deducing certain dependencies automatically [36]. Also, Wiegers provides checklists for straightforward, although manual, impact analysis [120].
- The development of methods and tools that can be integrated with mainstream software development tools typically used in organisations. For example, von Knethen and Grund propose a traceability model and an associated tool environment that includes a requirements management tool and a CASE tool [61].
- A focus on performing empirical research in general (see also Section 1.2.5). This is important to better tie industry and academia in order to obtain a healthy exchange of problems and solutions.

1.2.4 Uses

As stated initially, impact analysis can potentially be a powerful project and product management tool. This section discusses a number of possible uses of impact analysis. These uses were ranked as the most important from an organisational perspective in the study about uses and issues associated with impact analysis, presented in Chapter 4 (see Table 4.1). The view provided here can be seen as a complement, as it provides some details that were not part of the original paper on which Chapter 4 is based. The study distinguished between three organisational levels as follows:

- Decisions made on the *strategic* level typically have large scope, large impact and long-term perspective.
- Decisions made on the *tactical* level concern planning of time and resources to reach strategic goals, and are often made by middle management.

- Decisions made on the *operative* level are made when realising the project according to the plan, and are often of technical nature.

The discussions below take these organisational levels into consideration, by commenting on the importance of each use on the different levels. Figure 1.2 shows the average relative importance of the uses for each organisational level. For example, on the strategic level, use u1 was considered to account for 14% of the total importance of all uses, while the corresponding figures for the tactical and operative levels were around 11% and 8%, respectively.

Only uses that were among the top five ranked for at least one of the levels are discussed. It should also be noted that, in the study, impact analysis was seen in the light of proposed changes to requirements, not new requirements. The identifiers of the uses correspond to the ones used in Chapter 4.

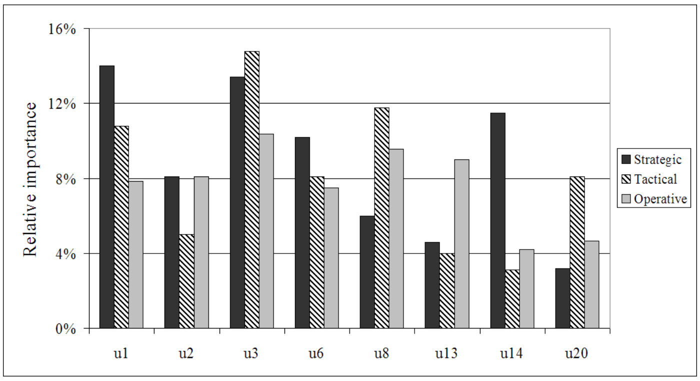


Figure 1.2 Importance of Impact Analysis Uses

Planning the project with respect to time and cost (u1)

This use is straightforward; the scope and technical complexity of a change directly feed into project planning. For example, if a change is extensive, it is likely that ordinary project activities will be delayed, which in turn will affect deadlines and deliveries. A technically complex change may require requesting additional expertise into the project, which naturally will increase project costs.

As can be seen in Figure 1.2, this use was regarded as more important on the strategic level than on the tactical level. This was unexpected, as the tactical level per definition is where decisions about planning and resources should take place.

Determining cost versus benefit (u2)

This use requires the analysis to include both an assessment of the cost of a change and the assessment of the perceived benefit for customers, for the market or even internally. Cost/benefit ratios can be used to, for example, prioritise between several concurrent changes. Furthermore, high-cost, high-benefit changes could indicate several interesting things, for example:

- The original requirements collection was not entirely successful. Thus, the system was designed based on partly incorrect assumptions.
- There is a large discrepancy between what the system currently is capable of doing and what it is expected to do. This is mostly an issue in a multi-release context, where changes are part of the incremental evolution of the system.
- The architecture (i.e., the basic structure) of the system is not very well designed in relation to expectations from customers (or the market).

This use was among the top five on both the operative level and the strategic level, but not on the tactical level. An explanation for this may be that people at the tactical level are not as close to neither customers (or the market) nor internal development as are the other two levels.

Deciding whether to accept or reject the change (u3)

This use could be seen as the ultimate goal of impact analysis of a proposed change, and is hierarchically related to other uses, such as determining cost versus benefit. The importance of this use is reflected by the fact that it was the top use on both the operative and the tactical level, and the second highest ranked use on the strategic level.

Understanding technical and market consequences of including or not including the change (u6)

This use is related to a number of other uses, and is an important precursor to deciding whether to accept or reject a change. Clearly, there is also a relation to using impact analysis to determine cost versus benefit. However, this use intuitively conveys a long-term perspective, including, for example, how the change will affect the system in comparison to competitors' systems. Similarly, technical consequences of a change include architectural issues of changeability, maintainability, interoperability and so on.

Figure 1.2 shows that this use was more important on the strategic level than on the other levels (although not very much). It was, however, not among the top five uses on the operative level. This was somewhat unexpected, as technical consequences should be important to consider on the operative level (given that decisions on this level typically are of technical nature).

Understanding the proposed change (u8)

An important aspect of impact analysis is to create and spread an understanding of the change. This is particularly important for changes that are technically complex or have large scope, as there otherwise is a risk that such changes are not implemented correctly, thereby potentially destabilising the system. However, there is also an aspect of education; it is crucial that all stakeholders are aware of changes made to the system and the actual functionality of the system, in particular when changes are frequent.

This use was ranked as the second most important on both the operative level and the tactical level, but was not among the top five uses on the strategic level. This may be because the tactical and operative levels are more concerned with the implementation of a change (from planning and technical points of view, respectively) than the strategic level.

Assessing system impact (u13)

This is a fundamental use, and traditionally one of the most prominent uses for impact analysis. For example, much effort has been put into devising and testing methods for determining impact based on dependencies in source code or between design entities (see, for example, [18], [66] and [68]). The definition of "system" can of course be discussed. For example, source code and design models can be said to be parts of the system, but does the same hold for requirements? One argument is that requirements constitute one view of the system, and should thus be considered as parts of it. Another argument is that requirements reflect the basic needs that the system is designed to satisfy, and are thus separate entities against which the system should be verified. Nevertheless, there are important differences between this use and other uses (such as planning the project with respect to time and cost).

This use was among the top five on the operative level, while it was not among the top five on the other levels. This confirms that the assessment of system impact is more important from a technical point of view, since it is typically technicians who are responsible for implementing changes in the system. Consequently, they are also the ones that have to correct faults introduced along with the changes.

Obtaining a new or changed requirements baseline (u14)

Leffingwell and Widrig define *baseline* to be "the itemised set of features, or requirements, intended to be delivered in a specific version of the application." [70] Of course, it is also possible to envision multiple baselines within one single project, for example due to internal milestones or deliveries. This use of impact analysis emphasises the importance of making sure that requirements are up-to-date and form a baseline for future changes. If the requirements baseline is out-of-date, it is not possible to accurately verify that the system functions as required at a delivery or milestone.

This use was among the top five only on the strategic level, which indicates that the focus on requirements is much higher there than on the other levels. This may be caused by a closer connection to customers (or the market), and consequently a need to make sure that the requirements at all times work as development drivers.

Revealing synergies and conflicts between change proposals (u20)

This is a particularly important use of impact analysis, as it potentially can alleviate the change effort through an understanding of how concurrent changes interact. Without such understanding, there is a risk that change activities become suboptimal (with respect to planning and resources), and that faults are introduced as a results of changes not working together.

This use was among the top five on the tactical level, which, given its potential, was not surprising. However, it was unexpected that it was not among the top five on the operative level, since people on this level should experience clear drawbacks from conflicting changes, for example.

The different uses presented above clearly show that there is more to impact analysis than technical details. Impact analysis is an activity that provides input to many different decision processes, and can thus be seen as a versatile requirements engineering and change management tool. Furthermore, the differences between the organisational levels demonstrate that the manifold nature of impact analysis is important to explore further.

1.2.5 In Industry

Empirical research about impact analysis exists, but is to date not very common, in particular not when it comes to research performed in cooperation with industry. One exception is the work by Lindvall and Sandahl, who have studied impact analysis from a requirements engineering perspective at Ericsson AB in Linköping, Sweden [73, 74]. Another is the work by Bratthall et al., who have studied the effect of using a design rationale when assessing impact, with senior industrial designers as some of the study participants [17].

In a study[1] about how well software practitioners estimate the impact of new requirements on an existing system, Lindvall and Sandahl found that the impact was underestimated by a factor of 3 [74]. This is one indication that industry needs better support for performing impact analysis. While the research community has constructed several tools (e.g., [21], [41], [61], [68] and [96]), and devised methods and techniques (e.g., [18], [26], [36] and [66]), impact analysis support must be built into existing tool sets and processes for industry to fully adopt it. An informal survey of a number of requirements management tools (see Chapter 2) revealed that only a third of the tools provided the ability to create relationships among all types of software development artefacts. This means in practise that many requirements management tools do not support even straightforward traceability-based impact analysis other than in a limited way.

The industrial study described in Chapter 4 provided some qualitative data, not mentioned in the chapter due to restrictive space constraints for the original paper. These data showed, among other things, that some of the major foundations for performing impact analysis were *experience*, *knowledge* and *gut feeling*. While these should

1. Some details and results from the study can be found in Chapter 2.

not be underestimated, they come with certain risks, as they put more focus on the subjective element of impact analysis than on the objective element. As can be seen in Chapter 4, some of the top issues associated with impact analysis were that impact is underestimated or missed, that affected parties are overlooked, and that the wrong persons perform the analyses. These three issues all confirm that the subjective element of impact analysis can be problematic.

Pour et al. discuss the gap between industry and academia for software engineering [91]. They argue that software engineering is still an immature field that needs to develop more. The remedy for bridging the gap between current practise and best practise in the field is to work closer between industry and academia. This clearly suggests that empirical research is one of the cornerstones of the success of the software engineering field. Thus, the same argument holds for impact analysis; more empirical research in the area is required in order to understand how it can be successfully adopted by industry and used to its full potential.

1.3 Thesis Papers

This section aims at introducing the reader to the papers included in the thesis, as well as discussing how they fit into the main thesis topic. A more detailed view of the research methodologies employed in the papers is given in Section 1.4, while the contributions of the papers and of the thesis in its entirety are outlined in Section 1.5.

The following papers are included in the thesis:

Chapter 2 Jönsson, P. and Lindvall, M. (2005). Impact Analysis. In A. Aurum and C. Wohlin (Eds.), *Engineering and Managing Software Requirements*. Springer-Verlag.

Chapter 3 Jönsson, P. and Wohlin, C. (2005). Understanding the Importance of Roles in Architecture-Related Process Improvement – A Case Study. In *Proceedings of the International Conference on Product Focused Software Process Improvement*, June 13-15, Oulu, Finland.

Chapter 4 Jönsson, P. and Wohlin, C. (2005). Understanding Impact Analysis: An Empirical Study to Capture Knowledge on Different Organisational Levels. Submitted to *International Conference on Software Engineer-*

ing and Knowledge Engineering, July 14-16, Taipei, Taiwan, Republic of China.

Chapter 5 Jönsson, P. and Wohlin, C. (n.d.). Semi-Automatic Impact Analysis through Keyword-Based Relationships – A Feasibility Study. To be submitted.

Chapter 6 Jönsson, P. and Wohlin, C. (2005). Benchmarking k-Nearest Neighbour Imputation with Homogeneous Likert Data. Submitted to *Journal of Empirical Software Engineering*.

The last paper was originally published at the 2004 International Metrics Symposium with the title "Evaluating k-Nearest Neighbour Imputation Using Likert Data". The thesis version is an extended version written and submitted in response to an invitation to a special issue of the Journal of Empirical Software Engineering.

The following paper is not included in the thesis, as it is not related to the thesis work:

Mårtensson, F., Jönsson, P., Bengtsson, PO., Grahn, H. and Mattsson, M. (2003). A Case Against Continuous Simulation for Software Architecture Evaluation. In *Proceedings of IASTED International Conference on Applied Simulation and Modelling*, September 3-5, Marbella, Spain, pp. 97-105.

1.3.1 Paper Abstracts

As an introduction to the chapters in the thesis, the abstracts of the original papers are given below.

Chapter 2 **Impact Analysis**

"Software changes are necessary and inevitable in software development, but may lead to software deterioration if not properly controlled. Impact analysis is the activity of identifying what needs to be modified in order to make a change, or to determine the consequences on the system if the change is implemented. Most research on impact analysis is presented and discussed in literature related to software maintenance. In this chapter, we take a different approach and discuss impact analysis from a requirements engineering perspective. We relate software change to impact analysis, outline the history of impact analysis and present common strategies for performing impact analysis. We also mention the application of impact analysis to non-functional requirements and discuss tool support for impact analysis. Finally, we outline what we see as the future of this essential change management tool.”

Chapter 3 **Understanding the Importance of Roles in Architecture-Related Process Improvement – A Case Study**

“In response to the increasingly challenging task of developing software, many companies turn to software process improvement (SPI). One of many factors that SPI depends on is user (staff) involvement, which is complicated by the fact that process users may differ in viewpoints and priorities. In this paper, we present a case study in which we performed a pre-SPI examination of process users’ viewpoints and priorities with respect to their roles. The study was conducted by the means of a questionnaire sent out to the process users. The analysis reveals differences among roles regarding priorities, in particular for product managers and designers, but not regarding viewpoints. This indicates that further research should investigate in which situations roles are likely to differ and in which they are likely to be similar. Moreover, since we initially expected both viewpoints and priorities to differ, it indicates that it is important to cover these aspects in SPI, and not only rely on expectations.”

Chapter 4 **Understanding Impact Analysis: An Empirical Study to Capture Knowledge on Different Organisational Levels**

“Change impact analysis is a crucial change management activity that previously has been studied much from a technical perspective. In this paper, we present a systematic interview-based study of a non-technical aspect of impact analysis. In the study, we have investigated how potential issues and uses of impact analysis are viewed by industrial experts at three organisational levels, based on Anthony’s decision-making model: operative, tactical and strategic. The results from our analyses show that on the whole, agreement on both issues and uses was large. There were, however, some differences among the levels in terms of issues. Thus, we conclude that it is both relevant and important to study impact analysis on different organisational levels.”

Chapter 5 **Semi-Automatic Impact Analysis through Keyword-Based Relationships – A Feasibility Study**

“Impact analysis plays an important role in requirements engineering as a tool to determine the changes necessary in order to implement a new requirement in an existing system. A common way of performing impact analysis is to examine dependencies among development artefacts, which is only possible if the dependencies are properly documented. This paper presents a feasibility study of a method for performing semi-automatic impact analysis based on lexicographic relationships between requirements and architecture component descriptions. Thus, it does not require explicitly documented dependencies. The method combines fuzzy string matching and Latent Semantic Indexing for identifying dependencies, but relies on manual expertise to estimate impact. An evaluation on an industrial software system with documented dependencies shows that the performance of the method is not as high as expected. However, the evaluation may have been misleading as the documented dependencies were very coarse. Thus, further evaluations will be performed to better assess the performance of the method.”

Chapter 6 **Benchmarking k-Nearest Neighbour Imputation With Homogeneous Likert Data**

“Missing data are common in surveys regardless of research field, undermining statistical analyses and biasing results. One solution is to use an imputation method, which recovers missing data by estimating replacement values. Previously, we have evaluated the hot-deck k-Nearest Neighbour (k-NN) method with Likert data in a software engineering context. In this paper, we extend the evaluation by benchmarking the method against four other imputation methods: Random Draw Substitution, Random Imputation, Median Imputation and Mode Imputation. By simulating both non-response and imputation, we obtain comparable performance measures for all methods. We discuss the performance of k-NN in the light of the other methods, but also for different values of *k*, different amounts of missing data, different neighbour selection strategies and different numbers of data attributes. Our results show that the k-NN method performs well, even when much data are missing, but has strong competition from both Median Imputation and Mode Imputation for our particular data. However, unlike these methods, k-NN has better performance with more data attributes. We suggest that a suitable value of *k* is approximately the square root of the number of complete cases, and that letting certain incomplete cases qualify as neighbours boosts the imputation ability of the method.”

1.3.2 Thesis Outline

This section presents the thesis chapters from the point of view of how they relate to each other and to the thesis as a whole. The chapters have been organised to reflect an important aspect of the work behind the thesis – to gain an understanding of impact analysis, both as a topic and as a tool in various contexts.

Chapter 2 serves as an introduction to impact analysis in addition to the one given in Section 1.2. It looks at the subject from a requirements engineering perspective, as opposed to the more common software maintenance perspective. Thus, it provides an understanding of how impact analysis works and can be used in the context of changing requirements. This chapter forms a foundation for the subsequent chapters.

Chapter 3 focuses on differences among roles in software process improvement, and thus does not relate directly to impact analysis. Instead, impact analysis stands out as an important improvement of software architecture documentation for most of the roles. This indicates the importance of understanding how impact analysis is seen from an organisational perspective, which is crucial when exploring further aspects of the topic. It was based on the study described in this chapter that impact analysis was selected as the target for subsequent research.

In Chapter 4, a study of issues and uses of impact analysis is presented. The intention of the chapter is to communicate an understanding of the views of impact analysis on different organisational levels. Compared to the previous chapter, the scope is more narrow, as the focus is shifted from software process improvement to organisational aspects of impact analysis. This chapter fills a gap in current impact analysis research, which mainly focuses on technical aspects (see Section 1.2), and is a natural continuation of the organisational track started in Chapter 3.

Next, Chapter 5 presents a method for semi-automatic impact analysis based on keyword-based lexicographic relationships between requirements and architecture component descriptions (i.e., existing written artefacts). The study described in Chapter 4 provided a number of important insights not discussed in the chapter, one of which was that impact analysis is often based on knowledge, experience and gut feeling (as opposed to models and documentation), which naturally entails certain risks. At the same time, however, knowledge, experience and gut feeling are essential components in any decision-making process (which impact analysis can be seen as). Thus, it is important to understand how the impact analysis activity can be supported in ways that help people make use of their expertise.

Finally, Chapter 6 explores a common problem of surveys encountered in the study presented in Chapter 3 – missing data. Missing data is a serious threat to research studies, in particular when it comes to statistical analyses, but can be circumvented through the use of imputation. In the Chapter 3 study, the k-Nearest Neighbour imputation method was used, based on recommendation, to reconstruct missing questionnaire responses. This prompted an evaluation of the method with Likert data, the kind of data used in the study. Thus, Chapter 6 is a sidetrack from the thesis topic, but provides a deeper understanding of how missing data in the Chapter 3 study were treated, and the various implications of this.

1.4 Research Methodology

The purpose of this section is to describe the various research methodologies used in the thesis chapters, and from that characterise the thesis as a whole. In order to accomplish this task, the thesis studies are positioned according to two taxonomies of software engineering research. First, though, a general discussion of the type of research that colours the thesis is given.

1.4.1 Research

Russ-Eft has recently (2004) surveyed a number of definitions of *research*, in order to be able to give an answer to the question: What is research anyway? [100] She identifies two main features, each of which has two contrasting dimensions. The first feature has to do with knowledge:

- Research extends present scientific knowledge and adds new insights, including constraints and limitations.
- Research confirms current knowledge, through replication of previous studies or by making current practises visible.

Clearly, research that presents new knowledge and introduces new concepts or methods is more “exciting” than research that tries to replicate earlier research to confirm (or reject) its findings. Nevertheless, the lack of replication within software engineering research poses a serious threat to the validity of research that bases assumptions on results from non-replicated studies. For example, Lindvall and Sandahl performed a study that is commonly cited in impact analysis papers as a proof that practitioners generally underestimate the impact of software changes [74]. However, without replication, this study provides only one data point, which not necessarily has the wide generalisability it so far has been attributed.

The second feature identified by Russ-Eft has to do with systematicness:

- Research is performed in a systematic and orderly fashion using disciplined processes.
- Research depends on the ability to recognise a solution to a problem even if it manifests haphazardly.

In the first of the two bullets above, research is depicted as something structured, whereas in the second bullet, there is an element of chance. Actually, many significant historical discoveries have been made by chance, the discovery of penicillin being an excellent example [19].

Russ-Eft concludes, in the light of the multidimensional features of research, that research by necessity must be clearly described and fully transparent for replications to be made and follow-up studies to be performed. This neatly reflects a basic intention with this thesis: to present the performed research in a clear and transparent way (although, for reasons of confidentiality, it is not possible to reveal all the raw data used in the studies).

1.4.2 Empirical Research

The main character of the research presented in this thesis is empiricism. Merriam-Webster Online defines *empirical* to be:

"(1) originating in or based on observation or experience; (2) relying on experience or observation alone often without due regard for system and theory; (3) capable of being verified or disproved by observation or experiment." [77]

In other words, relevant keywords for empirical research appear to be *observation* and *experience*, a view shared by others as well [98, 122]. Of the five studies presented in the thesis chapters, all can be seen as empirical. The study in Chapter 2 is much of a survey of relevant and current impact analysis literature, but is largely based on the experience of the authors and rests heavily on a number of empirical research publications. The studies in Chapters 3 to 5 are clearly empirical, as they are about identifying and observing phenomena in industry. Finally, the study in Chapter 6 can be seen as empirical as it is based on a simulation, which is one of the experimental research approaches listed by Zelkowitz and Wallace [125]. Furthermore, the data used in the simulation comes from the empirical study described in Chapter 3.

1.4.3 Research Taxonomies

To further establish the position of this thesis in relation to other software engineering research, the thesis studies are classified according to two different taxonomies:

- Zelkowitz and Wallace present a taxonomy for software engineering validation that contains 12 experimental approaches [125]. They argue that the taxonomy help create an understanding of experimentation within software engineering.
- Glass et al. present a taxonomy developed for the investigation of topics, research approaches, research methods, reference disciplines and levels of analysis of a great number of software engineering publications [48]. For brevity, the thesis studies are only classified with respect to research approach and research method.

The 12 experimental approaches in Zelkowitz and Wallace's taxonomy are divided into three main categories: *observational*, *historical* and *controlled*. In observational approaches, data is collected during the course of a project. There are four such approaches in the taxonomy:

- In *project monitoring*, existing data is collected in a non-intrusive way, exerting little or no pressure on the project.
- A *case study* is more intrusive, in that the researcher actively seeks data related to a certain goal. Additional work performed by the project members may affect their reactions towards the study.
- *Assertions* are experiments purposely designed to show the superiority of one technology, rather than investigating the differences between two technologies.

- Finally, *field studies* are like case studies, but less intrusive and with wider scope. In a field study, several projects are studied in parallel, which provides multiple comparable data points.

Historical approaches study historical data, for example from projects that are already completed. The taxonomy contains four historical approaches:

- In *literature search*, published work is passively analysed in order to, for example, confirm hypotheses or find related data.
- In *legacy data* approaches, existing quantitative data from a completed project is analysed in order to learn about trends or structures.
- *Lessons learned* approaches summarise qualitative findings from projects in order to communicate improvement potential and wisdom to future projects.
- Finally, *static analysis* is concerned with analysing the structure of a developed product. The difference between static analysis and studying legacy data is that the former focuses on the product, whereas the latter focuses on the process.

The third category, controlled approaches, contains approaches that allow multiple instances of an observations to be made. Also in this category, there are four approaches:

- In a *replicated experiment*, several projects undertake the same task, albeit with certain different parameters, for example programming language or test approach. Several data points allow the researchers to reach statistical validity.
- In *synthetic environment experiments*, researchers can study phenomena in artificial approximations of real environments. A problem with this kind of experiments is that the approximate nature of the environments restrict generalisation to real environments.
- *Dynamic analysis* is concerned with examining the characteristics approximate dynamic analysis.
- Last, *simulation* is a way to test a technology in a model of a real environment. A model allows certain variables to be disregarded, which is a clear advantage over dealing with a real environment.

The taxonomy developed by Glass et al. has the character of a framework for assessing software engineering research. They classify 369 journal papers according to the taxonomy in order to answer a number of question about software engineering research [48]. Because of the wide scope of the taxonomy, only two of its aspects are presented here: *research approaches* (overall approaches) and *research methods* (more detailed techniques).

Research approaches are divided into three main categories: *descriptive*, *evaluative* and *formulative*. The taxonomy contains three descriptive approaches:

- *Descriptive system*, capturing studies that focus on describing a system.
- *Descriptive other*, capturing studies that focus on describing something else than a system.
- *Review of literature*, capturing studies that mainly focus on reviewing literature.

Furthermore, the taxonomy contains four evaluative approaches[2]:

- *Evaluative-deductive*, capturing studies that focus on testing theories in order to increase predictive understanding. Such studies are typically of formal or quantitative nature, and often involve hypothesis testing and generalise findings from a sample to a population.
- *Evaluative-interpretive*, capturing studies that focus on understanding phenomena in their natural context and in a non-invasive way, from the perspective of the participants.
- *Evaluative-critical*, capturing studies that focus on critiquing well-established beliefs and assumptions that are taken for granted.
- *Evaluative-other*, capturing studies of evaluative character that do not fit into the previous three categories.

Last, the taxonomy contains six formulative approaches to cover studies that formulate entities of various kinds. The approaches are *formulative-framework*, *formulative-guidelines/standards*, *formulative-model*, *formulative-process/method/algorithm*, *formulative-classification/taxonomy* and *formulative-concept*.

The taxonomy contains 22 research methods in total, coded at the lowest possible level (i.e., not organised in a hierarchy). Many of the

2. These (apart from evaluative-other) are based on Orlikowski and Baroudi's classification of studies as *positivist*, *interpretive* and *critical*, respectively [87].

methods are very far from the actual methods used in the thesis studies, which is why the complete list is omitted here. The methods discussed below are: *case study*, *literature review/analysis*, *laboratory experiment (software)* and *simulation*.

1.4.4 Classification of Thesis Studies

Next, descriptions of the research methodologies of the thesis studies are given. Each study is classified according to the two taxonomies presented above.

Chapter 2 This chapter presents the state-of-the-art of impact analysis in requirements engineering. It gives a historical view of the subject and discusses current research with respect to methods, approaches, tools and metrics associated with impact analysis. Therefore, it must be categorised as a historical study according to the taxonomy by Zelkowitz and Wallace. More specifically, it can be said to be a literature search. Similarly, according to the taxonomy by Glass et al., the research approach employed is review of literature, and the research method used is literature review/analysis.

Chapter 3 This chapter describes a quantitative study of how viewpoints and priorities differ among software practitioners with different roles. The study was performed by means of a quantitative questionnaire sent to software practitioners systematically sampled from all employees at the studied company. In the chapter, both a rigorous statistical analysis of the results and a discussion of the practical implications of the results are presented.

The study fits nicely into the taxonomy by Zelkowitz and Wallace, according to which it could be classified as an observational study, more specifically a case study. The research approach of the study, according to the taxonomy by Glass et al., is evaluative-deductive, as it is concerned with the observation of the real world and inferential knowledge. With regard to research method, the study can be classified as a case study.

Chapter 4 This chapter describes a study that resembles the one in the previous chapter, but focuses on uses and issues associated with impact analysis on three organisational levels. Interviews containing both qualitative and quantitative questions were performed with a number of practitioners with different organisational roles. In a subsequent post-test, the interviewed persons were asked to prioritise both uses and issues mentioned during the interviews from different perspectives.

The title suggests that the chapter describes an "empirical study", but in terms of the taxonomies presented, the study can be classified as a case study (Zelkowitz and Wallace) and an evaluative-deductive case study (Glass et al.).

It should be noted that this chapter demonstrates a highly systematic approach for capturing knowledge on different organisational levels. Hence, it would be possible to consider it having a formulative-process/method/algorithm research approach. However, the systematic approach is not the main contribution of the chapter, and it is more an example of systematicness than a method.

Chapter 5 This chapter describes a feasibility study of a method for automatic impact analysis based on implicit lexicographic relationships between requirements and architecture component descriptions (and thus components). The proposed method makes use of Latent Semantic Indexing, a technique for information retrieval. The study includes an evaluation of the method using actual project data (including impact predictions made by developers) gathered from a historical project at Ericsson.

As opposed to the two previous chapters, this chapter does not present an observational study. Given the source of the data used for validation, the study could be classified as a historical study of legacy data, according to the taxonomy by Zelkowitz and Wallace. However, as the validation of the proposed method took place in a laboratory context, it would also be appropriate to classify the study as a synthetic environment experiment.

Looking at the taxonomy by Glass et al., the research approach of the study can be said to be formulative-process/method/algorithm, while the research method can be said to be laboratory experiment (software).

Chapter 6 The last chapter describes an evaluation of the k-Nearest Neighbour imputation method, a method for reconstructing missing data. The method is also benchmarked against a number of other imputation methods. Incomplete data sets artificially created based on a complete data set were fed to various imputation methods in a set of simulations. Evaluation data were subsequently collected from the simulations and analysed.

According to the taxonomy by Zelkowitz and Wallace, the study can be classified as a controlled experiment, more specifically a simulation, where a technology (the imputation method) is evaluated in a model of a real environment. With respect to the taxonomy by Glass et al., the research approach of the study can be classified as evaluative-deductive, while the research method can be classified as simulation.

The classification of the thesis studies is summarised in Table 1.3.

Table 1.3 **Classification of Thesis Studies**

Chapter	Zelkowitz and Wallace [125]	Glass et al. (approach – method) [48]
2	literature search	review of literature – literature review/analysis
3	case study	evaluative-deductive – case study
4	case study	evaluative-deductive – case study
5	legacy data, possibly synthetic environment experiment	formulative-process/method/algorithm – laboratory experiment (software)
6	simulation	evaluative-deductive – simulation

1.4.5 Research Validity

Kitchenham et al. bring up a discouraging observation about empirical software engineering – that the quality of its research generally is low [59]. In particular, they point out that statistical analysis often is incorrectly used within the software engineering field, mostly due to the fact that many researchers in the field lack enough statistical knowledge and experience. This problem has also been acknowledged by Miller, who provides an in-depth presentation of the many statistical traps that even experienced software engineering researchers fall into [80]. He states that a fundamental problem is that the software engineering field is widely different from the sciences for which statistical significance testing was originally intended.

Since two of the thesis chapters (Chapters 3 and 4) contain statistical analyses, it is relevant to comment on these analyses based on some of the guidelines provided by Kitchenham et al. They give the following important guidelines related to statistical analysis (see [59] for the complete list):

- **Ensure that the data do not violate the assumptions of the tests used on them** – In both studies, non-parametric tests were used in the data analysis. In the Chapter 3 study, the reason was that the data were ordinal and nominal. Thus, non-parametric Kruskal-Wallis and Chi-Square tests (see, for example, [106]) were used. In the Chapter 4 study, the data were first tested for departure from normality, and a Kruskal-Wallis test was used accordingly.
- **Present the raw data whenever possible. Otherwise, confirm that they are available for confidential review by the reviewers and independent auditors** – For confidentiality reasons, only screened data were possible to disclose.
- **Provide appropriate descriptive statistics** – This type of statistics can be found to some extent in both chapters. Chapter 3, for example, shows the distribution of answers on all questions posed in the study. Also, Chapter 4 shows box plots for the statistically significant issues found in the study.
- **Differentiate between statistical significance and practical importance** – This is a strong point that applies to both of the studies. As can be seen in the chapters, the practical implications of the results are discussed in addition to the statistical results. This is especially important in the light of Miller's assertion on the use of statistics in fields other than the intended ones (see above).
- **Specify limitations of the study** – Both chapters contain sections that discuss threats to the validity of the results, and how these threats were sought to be eliminated. Some further general points on validity are given below.

The guidelines above are mostly related to conclusion validity, which is concerned with the statistical relationship between the treatment and the outcome [122]. Three other types of validity are discussed below.

Construct Validity. Construct validity is concerned with the design of the main study instrument and that it measures what it is intended to measure [98]. This type of validity is relevant for all studies except the literature study in Chapter 2. The studies in Chapters 3 and 4 are questionnaire-based and interview-based, respectively. In order to ensure construct validity, the instruments used in both studies were constructed in close cooperation with Ericsson. Furthermore, the interview in the Chapter 4 study was piloted before the real interviews, in order further validate the correctness of the interview instrument.

In the study in Chapter 5, two tools were used. One was a custom tool developed specifically for the study, while the other was an existing tool publicly available. The correctness of the custom tool was ensured through rigorous testing, while the correctness of the already-developed tool was assumed based on its origins (see [11] and [99]).

In the study in Chapter 6, a custom tool was developed specifically for the study. The correctness of this tool was ensured through manual checking of the validity of its output for constructed test input.

External Validity. External validity is concerned with the generalisability of the results [122]. It is therefore tightly connected to how study respondents are sampled. The most appropriate sampling technique is *random sampling*, as it gives each person in the population an equal chance of being selected for the sample [98].

In the study in Chapter 3, the questionnaire recipients were sampled using *systematic sampling*, since every second person in an employee directory was selected. However, only one third of the persons in the sample responded to the questionnaire. To confirm that the respondents did represent the population, it was verified that the departmental distributions were similar.

In the study in Chapter 4, the interviewees were sampled based on recommendations from persons in the organisation. This approach is often referred to as *convenience sampling* [98]. However, as the sampling was based on recommendations, the selected interviewees should be relevant representatives for their respective organisational levels.

It is important to be aware of how far the generalisability of the results from the two studies mentioned above extends. Since both studies were performed at Ericsson, it can be argued that the results in both cases are generalisable to employees at Ericsson. However, when presenting research results, it is desirable to assert a wider generalisability. Given the brief presentation in Section 1.1, it is apparent that the company is one of the major players in its domain, and that it develops large-scale software products. Furthermore, Ericsson is an international company with development taking place in many different countries. Also, it has a rather generic change management process (see, for example, [70]). Thus, it is reasonable to believe that some of the observations made at Ericsson are generalisable also to other companies with similar characteristics.

Internal Validity. Internal validity is concerned with the relationship between the treatment and the outcome [122]. The imputation of data in the study in Chapter 3 has been justified by the evaluation of the imputation method used, presented in Chapter 6. In Chapter 4, the assignment of study participants to organisational levels was performed in several different ways in order to ensure that it was correct.

1.5 Contribution

This section presents the contribution of the thesis in its entirety, but also the more detailed contribution of each individual chapter.

1.5.1 Thesis

The most prominent contribution of the thesis as a whole is its clear focus on the organisational aspects of impact analysis. These are essential to understand in order to maximise the gain from impact analysis as a requirements engineering and change management tool. The organisational aspects are particularly brought forward in Chapters 4 and 5, but also in Chapter 3.

The second important contribution is the emphasis on impact analysis as a requirements engineering activity. Given the prevalent view of impact analysis as a software maintenance task, it is important to further explore and understand the requirements engineering aspects. The requirements engineering focus is especially reflected in Chapter 2, but is also inherent in Chapter 4.

Finally, the thesis is an example of empirical research about impact analysis, which is necessary to strengthen the connection between impact analysis research and impact analysis in industry (see Section 1.2.5).

1.5.2 Chapters

The contribution of Chapter 2 is a view of the state-of-the-art of impact analysis in the requirements engineering field. As articles about impact analysis mainly occur in software maintenance contexts, there is a clear need for complementing current research with a stronger focus on requirements engineering. This is further supported by the fact that impact analysis is a crucial activity in any change management process. As implied in the chapter, the challenges of impact analysis are not very different in requirements engineering than in software maintenance. The main difference is that in software maintenance, impact analysis is used to control changes made to an already released system for adaptive, corrective or perfective purposes [89], while in requirements engineering, impact analysis is concerned with the impact of new or changed requirements, possibly before the implementation has started. Furthermore, requirements engineering is more decision-oriented than software maintenance (for example, regarding inclusion or exclusion of new requirements).

The contribution of Chapter 3 is threefold:

- The results show that it is relevant to look at the role perspective when preparing software process improvement (SPI). To obtain as much support as possible among process users, and to maximise the understanding of issues to address, the ultimate approach would be to investigate viewpoints and priorities of every single process user. As this is clearly not feasible, it is necessary to approach process users in larger clusters. The results presented in the chapter indicate that clustering based on roles is sensible.
- Furthermore, the chapter provides an example of how roles can differ when it comes to priorities. While the exact differences are most likely specific for Ericsson, some plausible and possibly generalisable explanations for the differences are given (for example, that product managers have more market focus than other roles and therefore differ).
- Finally, the chapter highlights impact analysis in an organisational context (i.e., a non-technical context). Impact analysis was seen as an important improvement of architecture documentation by most of the participating roles.

The contribution of Chapter 4 is twofold:

- First and foremost, the chapter presents views of impact analysis on different organisational levels, thereby, like Chapter 3, departing from the common technical perspective or impact analysis. Uses and issues associated with impact analysis are examined, which increases the understanding of challenges in working with impact analysis.
- Second, the chapter has strong focus on the systematic interview-based method that was used to capture knowledge about uses and issues associated with impact analysis. The method serves as an explicit example of empirical research.

The contribution of Chapter 5 is the proposal and exploration of a method for performing semi-automatic impact analysis of new and changing requirements based on lexicographic relationships between requirements and architecture component descriptions. As discussed in Section 1.2.3, current impact analysis methods often fall short due to incomplete traceability among development artefacts. Thus, it is necessary to research methods and tools that work despite imperfect circumstances. Moreover, a certain degree of automation facilitates the impact analysis activity and is therefore an important step towards more structured impact analysis in industry. While the study only is a feasibility study, it serves as an initial step towards the intended goal.

Finally, the contribution of Chapter 6 is the evaluation of k-Nearest Neighbour imputation on software engineering Likert data, and the benchmarking against other imputation methods. Imputation methods can be expected to perform differently depending on the type of data being imputed, and it has been suggested that the effects of different types of data are more important to study than how various test statistics (such as mean and variance) are affected [93].

1.6 Future Work

The main character of this thesis has been the organisational perspective of impact analysis. Future work will continue in this area, in cooperation with Ericsson, but will focus more on support techniques than investigations of organisational views. The following sections show some directions of future work.

1.6.1 Semi-Automatic Impact Analysis

Future work will continue exploring the method proposed in Chapter 5. As stated, the method has two benefits: it works without a full-scale infrastructure of software development artefacts, and it provides a degree of automation that should be well-received in industry, where manual work is costly.

While the initial evaluation presented in Chapter 5 shows somewhat discouraging results, there are incentives for continued work. First and foremost, further evaluations will be performed to better determine the performance of the method, given that the initial evaluation may have been too coarse-grained to provide accurate results. Second, there are a number of parameters of the method that need to be investigated in greater detail. The effects of these parameters may be significant.

1.6.2 Technical Support for Organisational IA

Bearing on previous discussions, future work will focus on technical support for impact analysis in an organisational context. This includes methods and tools that can facilitate the impact analysis effort in different stages of software development, while being non-invasive with respect to changes to current ways of working. In practise, this means to make use of current tools, current development artefacts and existing expertise and knowledge.

The results from the study in Chapter 4 serve as an indication of the various aspects of impact analysis that should be pursued. The most important uses represent organisational activities that possibly can be supported further. Also, the most important issues reflect an improvement potential that can be explored.

1.6.3 Practitioners' Impact Analysis Ability

As stated in Section 1.4.1, the paper by Lindvall and Sandahl [74] about how well software practitioners can estimate change impact is well-cited, but represents after all only one data point. A replication of this study at Ericsson (however at a different office) is planned for two main reasons:

- It would make developers at Ericsson aware of their own ability of estimating the impact of new requirements (as well as changes to requirements). Furthermore, it would provide additional insight into organisational needs pertaining to impact analysis, and possibly identify further openings for support techniques.

- It would provide a second data point, and thus contribute further to the impact analysis field with knowledge about the actual state of impact analysis in the software development industry.

1.7 Summary

The aims of this chapter have been to briefly introduce some important aspects of the impact analysis field, to present the outline of the thesis, and to discuss the research methodologies used in the thesis studies.

Impact analysis research is commonly found in the software maintenance field, whereas it is an essential part of software change management, and thus requirements engineering. Chapter 2 provides an overview of the state-of-the-art of impact analysis in requirements engineering, and serves therefore as one step towards complementing current impact analysis research with a stronger requirements engineering focus.

Furthermore, impact analysis methods and tools are commonly of technical nature, and are not seldom based on the somewhat inappropriate assumption that software development organisations have full traceability among development artefacts and use structured and updated models and documentation. Thus, there is a need to investigate more organisational aspects of the area, such as how impact analysis is viewed and used by software practitioners. In this thesis, three chapters communicate the organisational perspective. Chapter 3 presents a study of organisational nature, where impact analysis surfaces as an important improvement of architecture documentation. Chapter 4 presents a study of uses and issues of impact analysis on different organisational levels, in order to create an understanding of how the levels differ in viewing the activity. Finally, Chapter 5 presents a feasibility study of a method for performing semi-automatic impact analysis based on existing written development artefacts. The method can potentially serve as a non-invasive support technique for impact analysis in different development stages.

Missing data is a problem that researchers in most research fields are faced with. Imputation, i.e. the reconstruction of missing data, was used in Chapter 3 to obtain a complete data set for the analysis. Chapter 6 provides a thorough evaluation and benchmarking of the imputation method used, and supports its use in the Chapter 3 study.

CHAPTER

2

Impact Analysis

In Engineering and Managing Software Requirements, Springer-Verlag, 2005

Per Jönsson and Mikael Lindvall

It is widely recognised that change is an inescapable property of any software, for a number of reasons. However, software changes can and will, if not properly controlled, lead to software deterioration. For example, when Mozilla's 2 000 000 Source Lines of Code (SLOC) were analysed, there were strong indications that the software had deteriorated significantly due to uncontrolled change, making the software very hard to maintain [50].

Software deterioration occurs in many cases because changes to software seldom have the small impact they are believed to have [120]. In 1983, some of the world's most expensive programming errors each involved the change of a single digit in a previously correct program [118], indicating that a seemingly trivial change may have immense impact. A study in the late 90s showed that software practitioners conducting impact analysis and estimating change in an industrial project underestimated the amount of change by a factor of 3 [74]. In addition, as software systems grow increasingly complex, the problems associated with software change increase accordingly. For example, when the source code across several versions of a 100 000 000 SLOC, fifteen-year-old telecom software system was analysed, it was noticed that the system had decayed due to frequent change. The programmers estimating the change effort drew the conclusion that the code was harder to change than it should be [37].

Impact analysis is a tool for controlling change, and thus for avoiding deterioration. Bohner and Arnold define impact analysis as "the activity of identifying the potential consequences, including side effects and ripple effects, of a change, or estimating what needs to be modified to accomplish a change before it has been made." [14] Consequently, the output from impact analysis can be used as a basis for estimating the cost associated with a change. The cost of the change can be used to decide whether or not to implement it depending on its cost/benefit ratio.

Impact analysis is an important part of requirements engineering since changes to software often are initiated by changes to the requirements. In requirements engineering textbooks, impact analysis is recognised as an essential activity in change management, but details about how to perform it often left out, or limited to reasoning about the impact of the change on the requirements specification (see, for example, [63], [70], [75], [97] and [111]). An exception is [120], where Wiegers provides checklists to be used by a knowledgeable developer to assess the impact of a change proposal.

Despite its natural place in requirements engineering, research about impact analysis is more commonly found in literature related to software maintenance. In this chapter, we present impact analysis from a requirements engineering perspective. In our experience, impact analysis is an integral part of every phase in software development. During requirements development, design and code do not yet exist, so new and changing requirements affect only the existing requirements. During design, code does not yet exist, so new and changing requirements affect only existing requirements and design. Finally, during implementation, new and changing requirements affect existing requirements as well as design and code. This is captured in Figure 2.1. Note that in less idealistic development processes, the situation still holds; requirements changes affect all existing system representations.

The chapter is organised as follows. In Section 2.1, we define concepts, discuss software change and outline the history of impact analysis. In Section 2.2, we present common strategies for impact analysis. Section 2.3 discusses impact analysis in the context of non-functional requirements. We explore a number of metrics for impact analysis and give an example of an application of such metrics in Section 2.4. In Section 2.5, we look at tool support for impact analysis and discuss impact analysis in requirements management tools. Finally we outline the future of impact analysis in Section 2.6 and provide a summary of the chapter in Section 2.7.

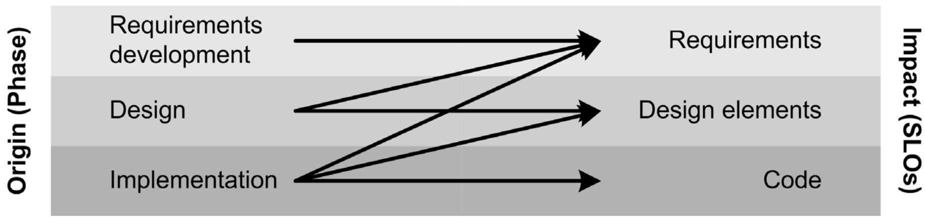


Figure 2.1 Software Lifecycle Objects (SLOs) Affected (*right*) Due to Requirements Changes in Different Phases (*left*)

2.1 Background

This purpose of this section is to provide a background for the remaining chapter. We begin by defining concepts related to impact analysis. Then, we discuss how impact analysis fits into software change. Finally, we outline a short impact analysis history.

2.1.1 Concepts and Terms

Throughout this chapter, we use several terms and concepts that are relevant in the field of impact analysis. In this section, we briefly visit these terms and concepts, and explain how each relates to impact analysis and to other terms and concepts.

Software lifecycle objects (SLOs – also called software products, or working products) are central to impact analysis. An SLO is an artifact produced during a project, such as a requirement, an architectural component, a class and so on. SLOs are connected to each other through a web of relationships. Relationships can be both between SLOs of the same type, and between SLOs of different types. For example, two requirements can be interconnected to signify that they are related to each other. A requirement can also be connected to an architectural component, for example, to signify that the component implements the requirement.

Impact analysis is often carried out by analysing the relationships between various entities in the system. We distinguish between two types of analysis: *dependency analysis* and *traceability analysis* [14]. In dependency analysis, detailed relationships among program entities, for example variables or functions, are extracted from source code. Traceability analysis, on the other hand, is the analysis of relationships that have been identified during development among all types of SLOs. Traceability analysis is thus suitable for analysing relationships among requirements, architectural components, documentation and so on. It is evident that traceability analysis has a broader application within requirements engineering than dependency analysis; it can be used in earlier development phases and can identify more diverse impact in terms of different SLO types.

It is common to deal with sets of impact in impact analysis. The following sets have been defined by Arnold and Bohner [14]:

- The *System Set* represents the set of all SLOs in the system – all the other sets are subsets of this set.
- The *Starting Impact Set* (SIS) represents the set of objects that are initially thought to be changed. The SIS typically serves as input to impact analysis approaches that are used for finding the Estimated Impact Set.
- The *Estimated Impact Set* (EIS) always includes the SIS and can therefore be seen as an expansion of the SIS. The expansion results from the application of change propagation rules to the internal object model repeatedly until all objects that may be affected are discovered. Ideally, the SIS and EIS should be the same, meaning that the impact is restricted to what was initially thought to be changed.
- The *Actual Impact Set* (AIS), finally, contains those SLOs that have been affected once the change has been implemented. In the best-case scenario, the AIS and EIS are the same, meaning that the impact estimation was perfect.

In addition to the impact sets, two forms of information are necessary in order to determine the impact of a change: information about the dependencies between objects, and knowledge about how *changes propagate* from object to object via dependencies and traceability links. Dependencies between objects are often captured in terms of references between them. Knowledge about how change propagates from one object to another is often expressed in terms of rules or algorithms.

It is common to distinguish between *primary* and *secondary change*. Primary change, also referred to as *direct impact*, corresponds to the SLOs that are identified by analysing how the effects of a proposed change affect the system. This analysis is typically difficult to automate because it is mainly based on human expertise. Consequently, little can be found in the literature about how to identify primary changes. It is more common to find discussions on how primary changes cause secondary changes, also referred to as *indirect impact*.

The indirect impact can take two forms. *Side effects* are unintended behaviours resulting from the modifications needed to implement the change. Side effects affect both the stability and function of the system and must be avoided. *Ripple effects*, on the other hand, are effects on some parts of the system caused by making changes to other parts. Ripple effects cannot be avoided, since they are the consequence of the system's structure and implementation. They must, however, be identified and accounted for when the change is implemented.

We have previously mentioned architectural components as an example of SLOs. The *software architecture* of a system is its basic structure, consisting of interconnected components. There are many definitions of software architecture, but a recent one is "the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them." [8] Several other definitions exist as well (see [109]), but most echo the one given here. Software architecture is typically designed early in the project, hiding low-level design and implementation details, and then iteratively refined as the knowledge about the system grows [27]. This makes architecture models interesting from a requirements engineering and impact analysis point-of-view, because they can be used for early, albeit initially coarse, impact analysis of changing requirements.

2.1.2 Software Change and Impact Analysis

Software change occurs for several reasons, for example, in order to fix faults, to add new features or to restructure the software to accommodate future changes [78]. Changing requirements is one of the most significant motivations for software change. Requirements change from the point in time when they are elicited until the system has been rendered obsolete. Changes to requirements reflect how the system must change in order to stay useful for its users and remain competitive on the market. At the same time, such changes pose a great risk as they may cause software deterioration. Thus, changes to requirements must be captured, managed and controlled carefully to ensure the survival of the system from a technical point of view. Factors that can inflict changes to requirements during both initial development as well as in software evolution are, according to Leffingwell and Widrig [70]:

- The problem that the system is supposed to solve changes, for example for economic, political or technological reasons.
- The users change their minds about what they want the system to do, as they understand their needs better. This can happen because the users initially were uncertain about what they wanted, or because new users enter the picture.
- The environment in which the system resides changes. For example, increases in speed and capacity of computers can affect the expectations of the system.
- The new system is developed and released leading users to discover new requirements.

The last factor is both real and common. When the new system is released, users realise that they want additional features, that they need data presented in other ways, that there are emerging needs to integrate the system with other systems, and so on. Thus, new requirements are generated by the use of the system itself. According to the "laws of software evolution", a system must be continually adapted, or it will be progressively less satisfactory in its environment [71].

Problems arise if requirements and changes to requirements are not managed properly by the development organisation [70]. For example, failure to ask the right questions to the right people at the right time during requirements development will most likely lead to a great number of requirements changes during subsequent phases. Furthermore, failure to create a practical change management process may mean that changes cannot be timely handled, or that changes are implemented without proper control.

Maciaszek points out: "Change is not a kick in the teeth, unmanaged change is." [75] In other words, an organisation that develops software requires a proper change management process in order to mitigate the risks of constantly changing requirements and their impact on the system. Leffingwell and Widrig discuss five necessary parts of a process for managing change [70]. These parts, depicted in Figure 2.2, form a framework for a change management process allowing the project team to manage changes in a controlled way.

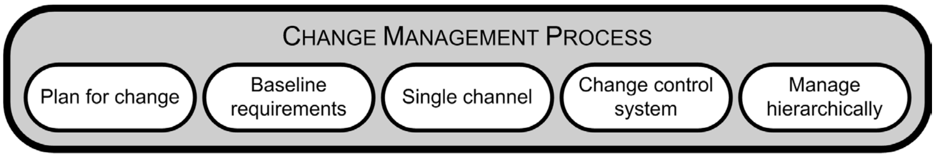


Figure 2.2 Change Management Process Framework

Plan for change involves recognising the fact that changes occur, and that they are a necessary part of the system's development. This preparation is essential for changes to be received and handled effectively.

Baseline requirements means to create a snapshot of the current set of requirements. The point of this step is to allow subsequent changes in the requirements to be compared with a stable, known set of requirements.

A *single channel* is necessary to ensure that no change is implemented in the system before it has been scrutinised by a person, or several persons, who keep the system, the project and the budget in mind. In larger organisations, the single channel is often a change control board (CCB).

A *change control system* allows the CCB (or equivalent) to gather, track and assess the impact of changes. According to Leffingwell and Widrig, a change must be assessed in terms of impact on cost and functionality, impact on external stakeholders (for example, customers) and potential to destabilise the system. If the latter is overlooked, the system (as pointed out earlier) is likely to deteriorate.

To *manage hierarchically* defeats a perhaps too common line of action – a change is introduced in the code by an ambitious programmer, who forgets, or overlooks, the potential effect the change has on test cases, design, architecture, requirements and so on. Changes should be introduced top-down, starting with the requirements. If the requirements are decomposed and linked to other SLOs, it is possible to propagate the change in a controlled way.

This framework for the change process leaves open the determination of an actual change process. Requirements engineering textbooks propose change management processes with varying levels of detail and explicitness [75, 97, 111]. The process proposed by Kotonya and Sommerville is, however, detailed and consists of the following steps [63]:

1. Problem analysis and change specification
2. Change analysis and costing, which in turn consists of:
    a. Check change request validity
    b. Find directly affected requirements
    c. Find dependent requirements
    d. Propose requirements changes
    e. Assess costs of change
    f. Assess cost acceptability
3. Change implementation

Impact analysis is performed in steps 2b, 2c and 2e, by identifying requirements and system components affected by the proposed change. The analysis should be expressed in terms of required effort, time, money and available resources. Kotonya and Sommerville suggest the use of traceability tables to identify and manage dependencies among requirements, and between requirements and design elements. We discuss traceability as a strategy for performing impact analysis in Section 2.2.1.

2.1.3 History and Trends

In some sense, impact analysis has been performed for a very long time, albeit not necessarily using that term and not necessarily resolving the problem of accurately determining the effect of a proposed change. The need for software practitioners to determine what to change in order to implement requirement changes has always been present. Strategies for performing impact analysis were introduced and discussed early in the literature. For example, Haney's paper from 1972 on a technique for module connection analysis is often referred to as the first paper on impact analysis [52]. The technique builds on the idea that every module pair of a system has a probability that a change in one module in the pair necessitates a change in the other module. The technique can be used to model change propagation between any system components including requirements. Program slicing, which is a technique for focusing on a particular problem by retrieving executable slices containing only the code that a specific variable depends on, was introduced already in 1979 by Weiser [119]. Slicing, which is explained in Section 2.2.1, can be used to determine dependencies in code and can be used to minimise side effects. Slicing can also be used to determine dependencies between sections in documents, including requirements, which is described below. Requirements traceability was defined in ANSI/IEEE Std 830-1984 in 1984 [3]. Traceability describes how SLOs are related to each other and can be used to determine how change in one type of artifact causes change in another type of artifact. The notion of ripple effect was introduced by Yau and Collofello in 1980 [123]. Their models can be used to determine how change in one area of the source code propagates and causes change in other areas.

Impact analysis relies on techniques and strategies that date back a long time. It is however possible to identify a trend in impact analysis research over the years. Early impact analysis work focused on source code analysis, including program slicing and ripple effects for code. The maturation of software engineering among software organisations has led to a need to understand how changes affect other SLOs than source code.

For example, Turver and Munro point out that source code is not the only product that has to be changed in order to develop a new release of the software product [117]. In a document-driven development approach, many documents are also affected by new and changed requirements. The user manual is an example of a document that has to be updated when new user functionalities have been provided. Turver and Munro focus on the problem of ripple effects in documentation using a thematic slicing technique. They note that this kind of analysis has not been widely discussed before. The same approach can be applied to the requirements document itself in order to determine how a new or changed requirement impacts the requirements specification.

In 1996, Arnold and Bohner published a collection of research articles called Software Change Impact Analysis [14]. The purpose of the collection was to present the current, somewhat scattered, material that was available on impact analysis at the time. Reading the collection today, nearly ten years later, it becomes apparent that it still is very relevant. Papers published after 1996 seem to work with the same ideas and techniques. We do not mean to depreciate the work that has been done, but it indicates that the field is not in a state of flux. Rather, the focus remains on adapting existing techniques and strategies to new concepts and in new contexts. Impact analysis on the architectural level is an example of this.

When the year 2000 approached, the Y2K problem made it obvious that extensive impact analysis efforts were needed in order to identify software and parts of software that had to be changed to survive the century shift. This served as a revelation for many organisations, in which the software process previously had not included explicit impact analysis [13].

Today, software systems are much more complex than they were 25 years ago, and it has become very difficult to grasp the combined implications of the requirements and their relationships to architecture, design, and source code. Thus, a need for impact analysis strategies that employ requirements and their relationships to other SLOs has developed. Still, dependency webs for large software systems can be so complex that it is necessary to visualise them in novel ways. Bohner and Gracanin present research that combines impact analysis and 3D visualisation in order to display dependency information in a richer format than is possible with 2D visualisation [15]. Bohner also stresses the need to extend impact analysis to middleware, COTS software and web applications. The use of these types of software is becoming more common, moving the complexity away from internal data and control dependencies to interoperability dependencies. Current impact analysis strategies are not very well suited for this type of dependencies [13].

2.2 Strategies for Impact Analysis

There are various strategies for performing impact analysis, some of which are more germane to the requirements engineering process than others. Common strategies are:

- Analysing traceability or dependency information.
- Utilising slicing techniques.
- Consulting design specifications and other documentation.
- Interviewing knowledgeable developers.

We divide these impact analysis strategies into two categories: *automatable* and *manual*. With automatable strategies, we mean those that are in some sense algorithmic in their nature. These have the ability to provide very fine-grained impact estimation in an automated fashion, but require on the other hand the presence of a detailed infrastructure and result at times in too many false positives [86]. With manual strategies, we mean those that are best performed by human beings (as opposed to tools). These require less infrastructure, but may be coarser in their impact estimation than the automatable ones. We recognise that the two categories are not entirely orthogonal, but they do make an important distinction; the manual strategies are potentially easier to adopt and work with because they require less structured input and no new forms of SLOs need to be developed.

A previous study indicated that developers' impact analyses often result in optimistic predictions [74], meaning that the predicted set of changes represents the least possible amount of work. Thus, the work cannot be easier, only more difficult. The study also identified the need for conservative predictions and establishing a "worst level" prediction. The real amount of work will lie between the optimistic and the conservative level. An improvement goal would be to decrease variation as the impact analysis process stabilises and becomes more mature.

The cost associated with producing a conservative prediction depends on its expected accuracy. Since conservative predictions identify such a large part of the system, developers often cannot believe they are realistic. The benefit of having a conservative prediction is the ability to determine a most probable prediction somewhere between the optimistic and the conservative prediction. An ideal impact analysis approach would always provide an optimistic and a conservative estimate. By collecting and analysing empirical data from the predictions as well as the actual changes, it can be established where in that span the correct answer lies.

2.2.1 Automatable Strategies

Automatable impact analysis strategies often employ algorithmic methods in order to identify change propagation and indirect impact. For example, relationship graphs for requirements and other SLOs can be used with graph algorithms to identify the impact a proposed change would have on the system. The prerequisite for automatable strategies is a structured specification of the system. By structured, we mean that the specification is consistent and complete, and includes some semantic information (for example, type of relationship). Once in place, such a specification can be used by tools in order to perform automatic impact analysis. Requirements dependency webs and object models are examples of structured specifications.

The strategies presented here, traceability and dependency analysis and slicing, are typically used to assess the Estimated Impact Set by identifying secondary changes made necessary because of primary changes to the system. They are not well suited for identifying direct impact.

Traceability/Dependency Analysis. Traceability analysis and dependency analysis both involve examining relationships among entities in the software. They differ in scope and detail level; traceability analysis is the analysis of relationships among all types of SLOs, while dependency analysis is the analysis of low-level dependencies extracted from source code [14].

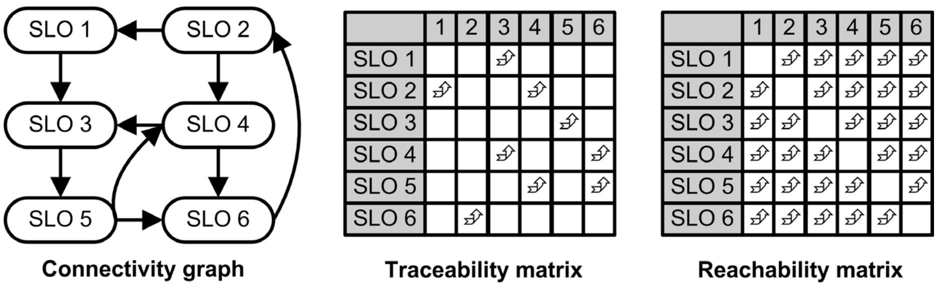
By extracting dependencies from source code, it is possible to obtain call graphs, control structures, data graphs and so on. Since source code is the most exact representation of the system, any analysis based on it can very precisely predict the impact of a change. Dependency analysis is also the most mature strategy for impact analysis available [14]. The drawback of using source code is that it is not available until late in the project, which makes dependency analysis narrow in its field of application. When requirements traceability exists down to the source, it can, however, be very efficient to use source code dependencies in order to determine the impact of requirements changes. A drawback is that very large systems have massive amounts of source code dependencies, which make the dependency web difficult to both use and to get an overview of [15].

Traceability analysis also requires the presence of relationship links between the SLOs that are analysed. Typically, these relationships are captured and specified progressively during development (known as pre-recorded traceability). The success of traceability analysis depends heavily on the completeness and consistency of the identified relationships. However, if traceability information is properly recorded from the beginning of development, the analysis can be very powerful.

A common approach for recording traceability links is to use a traceability matrix (see, for example, [63], [70] and [120]). A traceability matrix is a matrix where each row, and each column, corresponds to one particular SLO, for example a requirement. The relationship between two SLOs is expressed by putting a mark where the row of the first SLO and the column of the second SLO intersect. It is also possible to add semantic information to the relationship between SLOs. For example, the relationship between a requirement and an architectural component can be expanded to include information about whether the component implements the requirement entirely, or only partially.

Ramesh and Jarke report that current requirement practices do not fully embrace the use of semantic information to increase the usefulness of relationships between SLOs [95]. A relationship stating that two SLOs affect each other but not how, will be open to interpretation by all stakeholders. According to Ramesh and Jarke, different stakeholders interpret relationships without semantic information in different ways. For example, a user may read a relationship as "implemented-by", while a developer may read the same relationship as "puts-constraints-on".

To further illustrate the need for semantics in traceability links, we have created an example with six interconnected SLOs. Figure 2.3 shows the SLOs in a connectivity graph (left), where an arrow means that the source SLO affects the destination SLO. For example, SLO 2 affects, or has an impact on, SLO 1 and SLO 4.



	1	2	3	4	5	6
SLO 1			☞			
SLO 2	☞			☞		
SLO 3					☞	
SLO 4			☞			☞
SLO 5				☞		☞
SLO 6		☞				

	1	2	3	4	5	6
SLO 1		☞	☞	☞	☞	☞
SLO 2	☞		☞	☞	☞	☞
SLO 3	☞	☞		☞	☞	☞
SLO 4	☞	☞	☞		☞	☞
SLO 5	☞	☞	☞	☞		☞
SLO 6	☞	☞	☞	☞	☞	

Figure 2.3 Three Views of the Relationships Among SLOs

The connectivity graph corresponds exactly to a traceability matrix, shown next in the figure. An arrow in the traceability matrix indicates that the row SLO affects the column SLO. Both the connectivity graph and the traceability matrix show direct impact, or primary change needed, whereas indirect impact, or secondary change needed, can only be deduced by traversing the traceability links. For systems with many SLOs, the amount of indirect impact quickly becomes immense and hard to deduce from a connectivity graph or a traceability matrix. In order to better visualise indirect impact, the traceability matrix can be converted into a reachability matrix, using a transitive closure algorithm[1]. The reachability matrix for our example is also in Figure 2.3, showing that all SLOs eventually have impact on every other SLO. Consequently, the reachability matrix for this example is of limited use for assessing indirect impact. Bohner points out that this problem is common in software contexts, unless some action is taken to limit the range of indirect impact [13].

One way of limiting the range of indirect impact is to add distances to the reachability matrix. By doing so, it becomes possible to disregard indirect impacts with distances above a predefined threshold. This is a simple addition to the normal creation of reachability matrices, but it fails to address the fact that different types of traceability relationships may affect the range of indirect impact differently. Another solution is to equip the traceability matrix with traceability semantics and adjust the transitive closure algorithm to take such information into account. The algorithm should consider two SLOs reachable from each other only if the traceability relationships that form the path between them are of such types that are expected to propagate change.

Traceability analysis is useful in requirements engineering, which we view as an activity performed throughout the entire software lifecycle. Initially, traceability links can only be formed between requirements, but as design and implementation grow, links can be created from requirements to other SLOs as well.

Slicing Techniques. Slicing attempts to understand dependencies using independent slices of the program [45]. The program is sliced into a decomposition slice, which contains the place of the change, and the rest of the program, a complement slice. Slicing is based on data and control dependencies in the program. Changes made to the decomposition slice around the variable that the slice is based on are guaranteed not to affect the complement slice. Slicing limits the scope for propagation of change and makes that scope explicit. The technique is, for example, used by Turver and Munro for slicing of documents in order to account for ripple effects as a part of impact analysis [117]. Shahmehri et al. apply the technique to debugging and testing [105]. Pointer-based languages like C++ are supported through the work of Tip et al. and their slicing techniques for C++ [115]. Slicing tools are often based on character-based presentation techniques, which can make it more difficult to

1. The *transitive closure* of a graph is a graph where an edge is added between nodes A and B if it is possible to reach B from A in the original graph.

analyse dependencies, but visual presentation of slices can be applied to impact analysis as shown by Gallagher [44].

Architectural slicing was introduced by Zhao, and is similar to program slicing in that it identifies one slice of the architecture that is subject to the proposed change, and one that is not [126]. As opposed to conventional program slicing, architectural slicing operates on the software architecture of a system. As such, it can be employed in early development, before the code has been written. The technique uses a graph of information flows in order to trace those components that may be affected by the component being changed. In addition, those components that may affect the component being changed are also identified. This means that there must be a specification of the architecture that exposes all the information flows that it contains.

Slicing techniques can be useful in requirements engineering to isolate the impact of a requirements change to a specific part of the system. In order to provide a starting point for the slicing technique, the direct impact of the change must first be assessed.

2.2.2 Manual Strategies

Manual impact analysis strategies do not depend as heavily on structured specifications as their automatable counterparts do. Consequently, there is a risk that they are less precise in their predictions of impact. On the other hand, they may be easier to introduce in a change management process and are, in our experience, commonly employed in industry without regard to their precision.

The strategies presented here, using design documentation and interviewing, are primarily used for assessing the Starting Impact Set by identifying direct impact. The identification of indirect impact is possible, but is better handled by automatable strategies. Note that manual strategies, like the ones described here, can be used to capture traceability links between SLOs, to be used in traceability analysis.

Design Documentation. Design documentation comes in many different forms, for example as architecture sketches, view-based architecture models, object-oriented UML diagrams, textual descriptions of software components and so on. The quality of design documentation depends on the purpose for which it was written, the frequency with which it is updated, and the information it contains. It is far too common in industry that design documentation is written early in a project only to become shelfware, or that the documentation is written after the project, just for the sake of writing it. To perform impact analysis and determine how a new or changed requirement affect the system based on design documentation requires the documentation to be up-to-date and consistent with any implementation made so far. In addition, a prerequisite for using design documentation to assess direct impact is the possibility of relating requirements to design SLOs found in the documentation. The success and precision of this activity depends on a number of factors:

- The knowledge and skills of the persons performing the analysis. Persons with little insight into the system will most likely have problems pinpointing the impact of changed requirements in the system.
- The availability of the documentation. Documentation that is "hidden" in personal computers or stored in anonymous binders may be overlooked in the analysis.
- The amount of information conveyed in the documentation. Simple design sketches are common, but fail to express the semantics in connections between classes or architectural components. Ill-chosen naming schemes or inconsistent notation makes the analysis task arduous.
- Clear and consistent documentation. Ambiguous documentation is open for interpretation, meaning, for example, that the impact of a proposed change is coupled with great uncertainty, simply because another interpretation would have yielded different impact.

If the factors above have been taken into account, impact analysis of a requirements change can be performed by identifying the design SLOs that implement or in any other way depend on the requirements affected by the change. Additional measures that can be taken in order to alleviate the impact analysis effort are:

- Keep a design rationale. A design rationale is documentation describing why decisions are made the way they are. Bratthall et al. have performed an experiment on the effect of a design rationale when performing impact analysis [17]. The results from the experiment suggest that a design rationale in some cases can shorten the time required for impact analysis, and increase the quality of the analysis.

- Estimate impact of requirements as soon as the requirements are developed. The estimated impact is necessarily coarse to begin with, but can be improved incrementally as knowledge about the system increases.

Of course, structured design documentation can also be used with traceability analysis (see Section 2.2.1) to identify indirect impact. For example, Briand et al. propose a method for performing impact analysis in UML models, where they use a transitive closure algorithm to find indirect impacts in the models [18]. They do point out, however, the essential criterion that the UML models are updated as the system undergoes changes.

Interviews. Interviewing knowledgeable developers is probably the most common way to acquire information about likely effects of new or changed requirements according to a study on impact analysis [73]. The study found that developers perceive it as highly cost-effective to ask a knowledgeable person instead of searching in documents or other forms of information sources. Extensive communication between developers was also mentioned by developers as a success factor for software development projects. Analysis of source code was the second most common way of acquiring information about the likely impact of new or changed requirements. While all developers said they interviewed other developers and consulted source code, about half of the developers answered that they also consulted information, such as use-case models and object models, stored in the CASE tool in use. When asked why information in object models was not used more extensively, the developers answered that the information in object models was not detailed enough for impact analysis. In addition, they did not believe that the information in the models was up-to-date. "Source code, on the other hand, is always up-to-date." Among some developers, especially newcomers, the attitude towards using object models as the basis for determining change as an effect of new or changed requirements was less than positive. Object models (and the particular CASE tool that was used) were, however, mentioned as a good tool for documenting impact analysis and for answering questions about the relation between requirements and design objects using the support for traceability links.

2.3 Non-Functional Requirements

Requirements are often divided into *functional* and *non-functional* requirements. Non-functional requirements, or quality requirements, are those requirements "which are not specifically concerned with the functionality of the system" [63]. Non-functional requirements are often harder to deal with than functional ones, because their impact is generally not localised to one part of the system, but cuts across the whole system.

A non-functional requirement that, for example, relates to and calls for high security, often requires fundamental support in the software architecture, as it may constrain data access, file management, database views, available functionality and so on. Changes to functional requirements may also affect non-functional requirements. For example, if a change involves replacing a data transfer protocol to one that is more data intensive, overall system performance may be degraded.

One approach for dealing with non-functional requirements is to convert them into one or more functional requirements [16]. For example, a requirement stating that "no unauthorised person should be allowed access to the data" may be broken down into the more tangible requirements "a user must log into the system using a password" and "the user's identity must be verified against the login subsystem upon data access." Not all non-functional requirements can be converted in this way, however, which means that changes to them still have system-wide impact. Unfortunately, most impact analysis techniques deal exclusively with changes that can be initially pinpointed to a specific component, class or the like.

Lam and Shankararaman stress the distinction between functional impact analysis and quality impact analysis, i.e. impact analysis for functional and quality requirements, respectively [64]. They suggest the use of Quality Function Deployment (QFD) for dealing with changes to both functional and non-functional requirements. In QFD, a matrix connecting customer requirements with design features is constructed. A change to a requirement can be mapped to design features through the QFD matrix.

Cleland-Huang et al. accomplish performance-related impact analysis through event-based traceability [26]. In their approach, requirements are interconnected as event publishers to subscribing performance models. Whenever a change to a requirement is proposed, the relevant performance models are re-calculated. The resulting impact analysis is subsequently compared to constraints in the requirements specification. If several requirements are linked to the same performance model, they will all be verified against the impact analysis.

The impact of non-functional requirements is commonly dealt with in software architecture evaluation. Bosch has created a software architecture design method with a strong focus on non-functional requirements [16]. In the method, an initially functional architecture is progressively transformed until it is capable of meeting all non-functional requirements posed on the system. Parts of the method lend themselves well to impact analysis, since they deal with the challenge of assessing the often system-wide impact that non-functional requirements have. For most operational non-functional attributes (for example performance and reliability), a profile consisting of usage scenarios, describing typical uses of the system-to-be is created. The scenarios within the profile are assigned relative weights, in accordance with their frequency or probable occurrence. In scenario-based assessment, an impact analysis is performed by assessing the architectural impact of each scenario in the profile. For performance, the impact may be expressed as execution time, for example. Based on the impact and the relative weights of the scenarios, it is possible to calculate overall values (for example, throughput and execution time) for the quality attribute being evaluated. These values can be compared to the non-functional requirements corresponding to the quality attribute, in order to see whether they are met or not. Furthermore, they serve as constraints on the extent to which non-functional requirements can change before an architectural reorganisation is necessary. Also, should a functional requirement change, it is possible to incorporate the change in a speculative architecture, re-calculate the impact of the scenarios in the scenario profile, and see whether the non-functional requirements are still met or not.

2.4 Impact Analysis Metrics

Metrics are useful in impact analysis for various reasons. They can, for example, be used to measure and quantify change caused by a new or changed requirement at the point of the impact analysis activity. Metrics can also be used to evaluate the impact analysis process itself once the changes have been implemented. This is illustrated in Figure 2.4, in which two measure points are depicted;

one after the requirements phase has ended and design is about to start, and one when testing has been completed. Using these measure points, one can capture the predicted impact (the first point) and compare it to the actual impact (the second point). This kind of measurement is crucial for being able to do an analysis and learn from experiences in order to continuously improve the impact analysis capability. The figure is simplified and illustrates a learning cycle based on a waterfall-like model. As discussed earlier, impact analysis can be used throughout the life cycle in order to analyse new requirements and the measure points can be applied accordingly: whenever a prediction has been conducted and whenever an implementation has been completed.

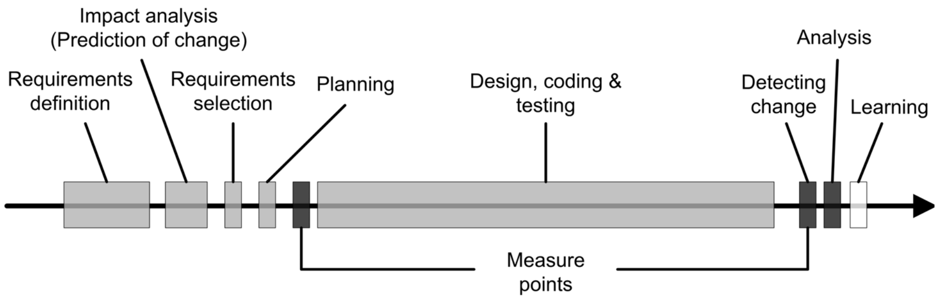


Figure 2.4 Measuring Impact Using Metrics

2.4.1 Metrics for Quantifying Change Impact

Metrics for quantifying change impact are based on the SLOs that are predicted to be changed as an effect of new or changed requirements. In addition, indicators of how severe the change is can be used. Such measures of the predicted impact can be used to estimate the cost of a proposed change or a new requirement. The more requirements and other SLOs that are affected, the more widespread they are and the more complex the proposed change is, the more expensive the new or changed requirement will be. Requirements that are costly in this sense but provide little value can, for example, be filtered out for the benefit of requirements that provide more value but to a smaller cost.

Change impact can be measured based on the set of requirements that is affected by the change. For example, the number of requirements affected by a change can be counted based on this set. The affected requirements' complexity often determines how severe the change is and can be measured in various ways. Examples are the size of each requirement in terms of function points and the dependencies of each requirement on other requirements. For other SLOs, the metrics are similar. For architecture and design, measures of impact include the number of affected components, the number of affected classes or modules, and number of affected methods or functions. For source code, low-level items such as affected lines of code can be measured and the level of complexity for components, classes, and methods can be measured using standard metrics such as cyclomatic complexity and regular object-oriented metrics.

In determining how severe or costly a change is, it is useful to define the impact factor. Lindvall defined the impact factors in Table 2.1 to measure the impact of a suggested change [73]. The impact factor is based on empirical findings in which it was determined that changes to different types of SLOs can be used as an indicator of the extent of the change. The higher the impact factor, the more severe the change. For example, changes that do not affect any other type of SLO but the design object model are relatively limited in scope. Changes that affect the use-case model are instead likely to require changes that are related to the fundamentals of the system and are therefore larger in scope. In addition, changes to the use-case model most likely also involve changes of all other SLOs making this kind of changes even more severe.

Table 2.1 Impact Factors

Factor	Impact	Description
M1	Change of the design object model.	These changes regard the real or physical description of the system and may generate change in the software architecture about the size of the change in the model.
M2	Change of the analysis object model.	These changes regard the ideal or logical description of the system. A small change here may generate change in the software architecture larger than the change in this model.
M3	Change the domain object model.	These changes regard the vocabulary needed in the system. A small change here may generate large change in the software architecture.
M4	Change the use-case model.	These changes require additions and deletions to the use-case model. Small changes here may require large change in the software architecture.

2.4.2 Metrics for Evaluation of Impact Analysis

Bohner and Arnold proposed a number of metrics with their introduction of impact sets [14]. These metrics are relations between the cardinalities of the impact sets, and can be seen as indicators of the effectiveness of the impact analysis approach employed (# denotes the cardinality of the set):

1. **#SIS / #EIS**, i.e. the number of SLOs initially thought to be affected over the number of SLOs estimated to be affected (primary change and secondary change). A ratio close to 1 is desired, as it indicates that the impact is restricted to the SLOs in SIS. A ratio much less than 1 indicates that many SLOs are targeted for indirect impact, which means that it will be time-consuming to check them.
2. **#EIS / #System**, i.e. the number of SLOs estimated to be affected over the number of SLOs in the system. The desired ratio is much less than 1, as it indicates that the changes are restricted to a small part of the system. A ratio close to 1 would indicate either a faulty impact analysis approach or a system with extreme ripple effects.
3. **#EIS / #AIS**, i.e. the number of SLOs estimated to be affected over the number of SLOs actually affected. The desired ratio is 1, as it indicates that the impact was perfectly estimated. In reality, it is likely that the ratio is smaller than 1, indicating that the approach failed to estimate all impacts. Two special cases are if AIS and EIS only partly overlap or do not overlap at all, which also would indicate a failure of the impact analysis approach.

Fasolino and Visaggio also define metrics based on the cardinalities of the impact sets [41]. They tie the metrics to properties and characteristics of the impact analysis approach, as per the tree in Figure 2.5.

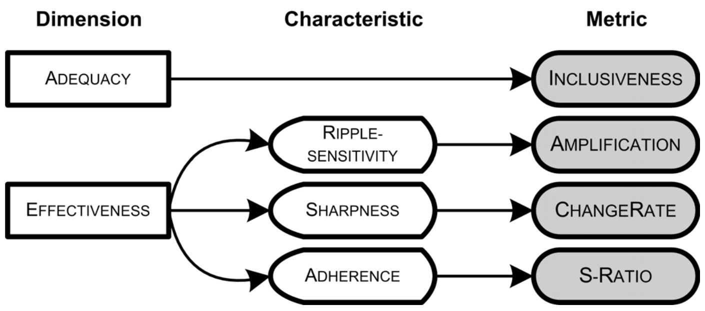


Figure 2.5 Tree of Impact Analysis Metrics

Adequacy is the ability of the impact analysis approach to estimate the impact set. It is measured by means of the binary metric *Inclu-*

siveness, which is strictly defined to 1 if all SLOs in AIS also are in EIS and 0 otherwise. *Effectiveness* is the ability of the approach to provide beneficial results. It is refined into *Ripple-sensitivity* (the ability to identify ripple effects), *Sharpness* (the ability not to overestimate the impact) and *Adherence* (the ability to estimate the correct impact).

Ripple-sensitivity is measured by *Amplification*, which is defined as **(#EIS - #SIS) / #SIS**, i.e. the ratio between the number of indirectly impacted SLOs and the number of directly impacted SLOs. This ratio should preferably not be much larger than 1, which would indicate much more indirect impact than direct impact. Sharpness is measured by *ChangeRate*, which is defined as **#EIS / #System**. This is the same metric as the second of Arnold and Bohner's metrics presented previously. Adherence is measured by *S-Ratio*, which is defined as **#AIS / #EIS**. S-Ratio is the converse of the third of Arnold and Bohner's metrics presented previously.

Lam and Shankararaman propose metrics that are not related to the impact sets. These metrics are more loosely defined and lack consequently recommended values [64]:

- *Quality deviation*, i.e. the difference in some quality attribute (for example, performance) before and after the changes have been implemented, or between actual and simulated values. A larger than expected difference could indicate that the impact analysis approach failed to identify all impact.
- *Defect count*, i.e. the number of defects that arise after the changes have been implemented. A large number of defects could indicate that some impact was overlooked by the impact analysis approach.
- *Dependency count*, i.e. the number of requirements that depend on a particular requirement. Requirements with high dependency count should be carefully examined when being subjected to change.

Lindvall defined and used metrics in a study at Ericsson in order to answer a number of questions related to the result (prediction) of impact analysis as conducted in a commercial software project and performed by the project developers as part of the regular project work [73]. The study was based on impact analysis conducted in the requirements phase, as Figure 2.4 indicates, and the term *requirements-driven impact analysis* was coined to capture this fact. The results from the impact analysis was used by the Ericsson project to estimate implementation cost and to select requirements for implementation based on the estimated cost versus perceived benefit. The study first looked at the collected set of requirements' predicted and actual impact by answering the following questions: "How good was the prediction of the change caused by new and changed requirements in terms of predicting the number of C++ classes to be changed?" and "How good was this prediction in terms of predicting which classes to be changed?" The last question was broken down into the two sub questions: "Were changed classes predicted?" and "Were the predicted classes changed?"

There were a total of 136 C++ classes in the software system. 30 of these were predicted to be changed. The analysis of the source code edits showed that 94 classes were actually changed. Thus, only 31.0% (30 / 94) of the number of changed classes were predicted to be changed.

In order to analyse the data further, the classes were divided into the two groups *Predictive group* and *Actual group*. In addition, each group was divided into two subgroups: *Unchanged* and *Changed*. The 136 classes were distributed among these four groups as shown in Table 2.2.

Table 2.2 Predicted vs. Actual Changes

		Predictive group		
		Unchanged	**Changed**	
Actual group	**Unchanged**	A: 42 (30.9%)	B: 0 (0.0%)	A+B: 42 (30.9%)
	Changed	C: 64 (47.1%)	D: 30 (22.1%)	C+D: 94 (69.1%)
		A+C: 106 (77.9%)	B+D: 30 (22.1%)	N: 136 (100.0%)

Cell A represents the 42 classes that were not predicted to change and that also remained unchanged. The prediction was correct as these classes were predicted to remain unchanged, which also turned out to be true. The prediction was implicit as these classes were indirectly identified – they resulted as a side effect as complement of predicting changed classes.

Cell B represents the zero classes that were predicted to change, but actually remained unchanged. A large number here would indicate a large deviation from the prediction.

Cell C represents the 64 classes that were not predicted to change, but turned out to be changed after all. As with cell B, a large number in this cell indicates a large deviation from the prediction.

Cell D, finally, represents the 30 classes that were predicted to be changed and were, in fact, changed. This is a correct prediction. A large number in this cell indicates a good prediction.

There are several ways to analyse the goodness of the prediction. One way is to calculate the percentage of correct predictions, which was (42 + 30) / 136 = 52.9%. Thus, the prediction was correct in about half of the cases. Another way is to use Cohen's Kappa value, which measures the agreement between two groups ranging from -1.0 to 1.0. The -1.0 figure means total discompliance between the two groups, 1.0 means total compliance and 0.0 means that the result is no better than pure chance [28]. The kappa value in this case is 0.22, which indicates a fair prediction. We refer to [74] for full details on the Kappa calculations for the example. A third way to evaluate the prediction is to compare the number of classes predicted to be changed with the number of classes actually changed. The number of classes predicted to be changed in this case turned out to be largely underpredicted by a factor of 3. Thus, only about one third of the set of changed classes was identified. It is, however, worth noticing that all of the classes that were predicted to be changed were in fact changed.

The study then analysed the predicted and actual impact of each requirement by answering similar questions for each requirement. The requirements and the classes that were affected by these requirements were organised in the following manner: For each requirement, the set of classes predicted to be changed, the set of changed classes and the intersection of the two sets, i.e. classes that were both predicted and changed. In addition, the sets of classes that were predicted but not changed and the set of classes that were changed but not predicted were identified.

The analysis showed that in almost all cases, there was an underprediction in terms of number of classes. In summary, the analysis showed that the number of changed classes divided by the number of predicted classes ranged from 1.0 to 7.0. Thus, up to 7 times more classes than predicted were actually changed.

Estimating cost in requirements selection is often based on the prediction like it was in the Ericsson case, which means that requirements predicted to cause change in only a few entities are regarded as less expensive, while requirements predicted to cause change in many entities are regarded as more expensive. This makes the rank-order of requirements selection equal to a requirements list sorted by the number of items predicted. By comparing the relative order based on the number of predicted classes with the relative order based on the number of actually changed classes, it was possible to judge the goodness of the prediction from yet another point of view. In summary, the analysis on the requirements level showed that a majority of the requirements were underpredicted. It was also clear that it is relatively common that some classes predicted for one requirement are not changed because of this particular requirement, but because of some other requirement. This is probably because the developers were not required to implement the changed requirements exactly as was specified in the implementation proposal resulting from the impact analysis. The analysis of the order of requirements based on number of predicted classes showed that the order was not kept entirely intact; some requirements that were predicted to be small proved to have a large change impact, and vice versa.

In order to try to understand the requirements-driven impact analysis process and how to improve it, an analysis of the various characteristics of changed and unchanged classes was undertaken. One such characteristic was size, and the questions were: "Were large classes changed?", "Were large classes predicted?" and "Were large classes predicted compared to changed classes?"

The analysis indicated that large classes were changed, while small classes remained unchanged. The analysis also indicated that large classes were predicted to change, which leads to the conclusion that class size may be one of the ingredients used by developers, maybe unconsciously, when searching for candidates for a new or changed requirement.

2.5 Tool Support

The complexity of the change management process makes it necessary to use some sort of tool support [75, 111]. A change management tool can be used to manage requirements and other SLOs, manage change requests, link change requests to requirements and other SLOs, and monitor the impact analysis progress. A simple database or spreadsheet tool may be used as basic change management support, but still requires a considerable amount of manual work, which eventually may lead to inconsistencies in the change management data. If the tool support is not an integral part of the change management process, there is always a risk that it will not be used properly. A change management system that is not used to its full extent cannot provide proper support to the process.

A problem with many change management tools is that they are restricted to working with change and impact analysis on the requirements level. Ideally, a change management tool would support impact analysis on requirements, design, source code, test cases and so on. However, that would require the integration of requirement management tools, design tools and development environments into one tool or tool set. In a requirements catalogue for requirements management tools, Hoffmann et al. list both traceability and tool integration as high-priority requirements, and analysis functions as a mid-priority requirement, confirming the importance of these features [53].

In a survey of the features of 29 requirements management tools supporting traceability, we could only find nine tools for which it was explicitly stated on their web sites that they supported traceability between requirements and other SLOs, such as design elements, test cases and code. Depending on the verbosity and quality of the available information, this may not be an exact figure. However, it indicates that in many cases it is necessary to use several different tools to manage traceability and perform impact analysis, which can be problematic depending on the degree of integration between the tools.

There are tools that extract dependency information from existing system representations, for example source code and object models, but the task of such tools is nonetheless difficult and often requires manual work [36]. Higher-level representations may be too coarse, and source code may have hidden dependencies, for instance due to late binding. Egyed, for example, proposes an approach for extracting dependencies primarily for source code [36]. Input to the approach is a set of test scenarios and some hypothesised traces that link SLOs to scenarios. The approach then calculates the footprints of the scenarios, i.e. the source code lines they cover, and based on footprints and hypothesised traces generates the remaining traces. The approach can also be used when no source code exists, for example by simulating the system or hypothesising around the footprints of the scenarios.

Tools that deal with source code are mostly used in software maintenance contexts, and are obviously of limited use within the development project. Natt och Dag et al. have studied automatic similarity analysis as a means to find duplicate requirements in market-driven development [82]. In addition to the original field of application, they suggest that their technique can be used to identify dependency relationships between requirements, for example that two requirements have an "or" relation, or that several requirements deal with similar functionality.

Tools that aid in performing impact analysis can be synonymous with the underlying methods. Methods that rely on traceability analysis are well suited for inclusion in tools that try to predict indirect impact. For example, Fasolino and Visaggio present ANALYST, a tool that assesses impact in dependency-based models [41]. Lee et al. present another tool, ChAT, that calculates ripple effects caused by a change to the system [68]. Many such tools are commonly proof-of-concept tools, constructed to show or support a particular algorithm or methodology. What is lacking is the integration into mainstream change management tools.

2.6 Future of Impact Analysis

Most strategies for impact analysis work under the assumption that changes only affect functionality. It is thus more difficult to assess the impact of changes to non-functional requirements, or changes where non-functional requirements are indirectly affected. Some work on this topic exists (see [26] and [64]), but a stronger focus on impact analysis for non-functional requirements is needed.

As we have pointed out, impact analysis is mostly referred to in software maintenance contexts. We have argued that impact analysis is an essential activity also in requirements engineering contexts, and that standard impact analysis strategies apply in most cases (for example, traceability approaches are commonly exercised for requirements). There is still, however, a need for more research focusing on the requirements engineering aspects of impact analysis, for example how to relate requirements to other SLOs and how to perform change propagation in this context.

Most automatable strategies for impact analysis assume complete models and full traceability information. Since it is common in industry to encounter models that are not updated and traceability information that is only partial, there is a need for more robust impact analysis strategies that can work with partial information. Egyed has proposed one such approach [36].

Existing tools for impact analysis are often proof-of-concept tools, or work only with limited impact analysis problems, such as the extraction of dependencies from system representations. Some mainstream requirements management tools incorporate impact analysis of not only requirements, but also design, code and test, but far from all. Full-scale impact analysis must be an integral part of requirement management tools in order for change to be dealt with properly.

Impact analysis needs to be adapted to the types of systems that become increasingly common today, such as web applications and COTS software. Web applications, for example, often consist of standalone components that connect to a central repository, such as a database. Thus, there are few control dependencies between components, and instead rich webs of data dependencies towards and within the central repository. The fact that such repositories can be shared among several distinct systems introduces interoperability dependencies that impact analysis strategies especially tailored for these technologies must address in order to be effective.

2.7 Summary

Impact analysis is an important part of requirements engineering since changes to software often are initiated by changes to the requirements. As the development process becomes less and less waterfall-like and more of new and changed requirements can be expected throughout the development process, impact analysis becomes an integral part of every phase in software development. In some sense, impact analysis has been performed for a very long time, albeit not necessarily using that term and not necessarily fully resolving the problem of accurately determining the effect of a proposed change. The need for software practitioners to determine what to change in order to implement requirement changes has always been present. Classical methods and strategies to conduct impact analysis are dependency analysis, traceability analysis and slicing. Early impact analysis work focused on applying such methods and strategies onto source code in order to conduct program slicing and determine ripple effects for code changes. The maturation of software engineering among software organisations has, however, led to a need to understand how change requests affect other SLOs than source code, including requirements, and the same methods and strategies have been applied. Typical methods and strategies of today are based on analysing traceability or dependency information, utilising slicing techniques, consulting design specifications and other documentation, and interviewing knowledgeable developers. Interviewing knowledgeable developers is probably the most common way to acquire information about likely effects of new or changed requirements. Metrics are useful and important in impact analysis for various reasons. Metrics can, for example, be used to measure and quantify change caused by a new or changed requirement at the point of the impact analysis activity. Metrics can also be used to evaluate the impact analysis process itself once the changes have been implemented. In determining how severe or costly a change is, it is useful to determine the impact factor as it indicates the likely extent of a change to a certain type of SLO. To summarise: Impact analysis is a crucial activity supporting requirements engineering. The results from impact analysis feed into many activities including estimation of requirements' cost and prioritisation of requirements. These activities feed directly into project planning, making impact analysis a central activity in a successful project.

CHAPTER 3

Understanding the Importance of Roles in Architecture-Related Process Improvement – A Case Study

Intl. Conference on Product Focused Software Process Improvement, 2005

Per Jönsson and Claes Wohlin

Constraining factors such as time and budget make software development a challenging task for many organisations – a challenge that is leveraged by the fact that software plays an increasingly large role in society. In order to handle the challenge and to turn the software industry into an engineering discipline, it is necessary to put the processes in focus [124]. The goal of Software Process Improvement (SPI) is to create an infrastructure that enables effective methods and practices to be incorporated into the business [1].

The success of SPI depends on a number of factors, one of which is user (staff) involvement [94]. It has been reported that process users' attitudes often are disregarded in quality initiatives, and that putting them in the spotlight when designing SPI is an important step towards success [1, 51]. To involve process users and to regard their attitudes can be far from trivial, because process users do neither necessarily have the same viewpoints, nor the same priorities. This chapter presents a case study in which we examined the viewpoints and priorities of process users at Ericsson, to pinpoint differences and similarities among roles. We selected the role perspective since a number of publications report that role can be a discriminating factor when it comes to views in SPI [5, 10, 27, 29, 51, 58, 114]

Generic SPI frameworks, such as SEI's IDEAL^{SM} [79], Quality Improvement Paradigm (QIP) [7], and PROFES [92], all contain two important ingredients: a *characterisation* (or assessment, or appraisal) of the current process, and an *improvement plan* (or roadmap, or actions). The viewpoints of the process users are crucial in the characterisation phase, because the more diverse they are, the harder it becomes to form a baseline. Similarly, the priorities of the process users are crucial in the planning phase, because the more diverse they are, the harder it becomes to create a plan that satisfies everyone.

The chapter is structured as follows. Section 3.1 outlines the research setting and discusses architecture-related process improvement. Section 3.2 addresses related work, while Section 3.3 explains the design of the study as well as how the study was carried out. The results are presented in Section 3.4, followed by a statistical analysis in Section 3.5, a general discussion in Section 3.6 and finally conclusions in Section 3.7.

3.1 Background and Research Setting

As stated in Chapter 1, Ericsson is one of the largest suppliers of mobile systems in the world, and has as customers some of the world's largest mobile operators. The study was conducted at one of Ericsson's offices (hereafter referred to as *the company*), which at the time had about 400 employees.

The objective of the study was to prepare improvement of the architecture documentation process at the company by examining process users' viewpoints and priorities with respect to their roles. By doing so, we were able to create an awareness of the need for and scope of SPI. With "architecture documentation process", we refer to the process of documenting the software architecture and keeping the documentation up-to-date. This is not necessarily an explicitly defined process of its own, but could, for example, be part of the development process. Our tool for examining viewpoints and priorities was a questionnaire with quantitative questions about architecture documentation.

In advance, we expected to see both diverse viewpoints and diverse priorities among process users regarding the architecture documentation process. The reason for this was mainly that architecture documentation typically has different stakeholders, such as project managers, product managers, designers and testers, most of whom have different needs and knowledge. Both needs and knowledge are factors that tend to affect how you view things and what you think is important. This is one of the reasons that software architectures should be designed and documented using multiple architectural views [27]. Since the organisational role most likely affects both needs and knowledge, we anticipated differences in both viewpoints and priorities.

We apply statistical methods to test for differences among roles. For this purpose, the following statistical hypotheses are evaluated (independently for viewpoints and priorities):

- **Null hypothesis, H_0**: There are no differences among roles.
- **Alternative hypothesis, H_A**: There is a difference among roles.

This study is justified by two main reasons. First, it adds to existing research about similarities and non-similarities among roles, which is necessary to better understand the impact of roles in various research contexts. Second, it provides an example of empirical SPI research targeted at industry, with the focus on creating an understanding of process users' viewpoints and priorities.

3.1.1 Architecture-Related Process Improvement

As mentioned, we explored process users' viewpoints and opinions in order to prepare for improvement of the architecture documentation process. However, the questions posed in the questionnaire were more germane to the product (i.e., the architecture documentation) than to the process (i.e., documenting the architecture). Our reason for posing questions about the product rather than the process was that we considered the product to be more tangible to the process users than the process itself. In other words, we expected that we would get more well-founded answers by asking about the product. Furthermore, we argue that the quality of the documentation reflects the quality of the process of documenting it as much as, for example, the quality of requirements reflects the quality of the process of eliciting, formulating and managing them.

Since the software architecture of a system is a fundamental building block that has many stakeholders within the organisation, changes to architecture-related processes can have great organisational impact, including new workflows, altered mindsets and changed team structures. This further establishes the need for investigating differences in viewpoints and priorities among process users.

3.2 Related Work

For an overview of the Software Process Improvement area, we recommend Zahran's book, which covers the topic from a practitioner's perspective [124]. Zahran discusses the general challenges of SPI, such as management commitment, buy-in from process users etc. There are some publications (see below) that discuss differences among roles in various contexts. In general, they show that there are differences among roles in some situations, but not in others. It all depends on what you evaluate.

Baddoo and Hall have studied de-motivators in SPI among software practitioners, in order to understand what hinders SPI success [5]. Dividing the practitioners into developers, project managers and senior managers, they find both common and unique de-motivators. They conclude that differences in de-motivators for SPI often are related to the roles that software practitioners have.

Berander and Wohlin have looked at agreement on SPI issues among traditional software development roles [10]. Their findings indicate that there is agreement among roles about communication of processes, but disagreement about, for example, importance of improvement and urgency of problems.

Conradi and Dybå have investigated how the use of formal routines for transferring knowledge and experience is perceived by developers and managers [29]. Their results show that there is a difference between the two groups; developers are more sceptical to formal routines, while managers take them for granted.

Hall and Wilson have studied views of quality among software practitioners in UK companies, with focus on two groups – managers and developers [51]. According to their findings, developers and managers differ in that they have different primary quality concerns, although the differences are not conflicting.

Karlström et al. present a method for aggregating viewpoints of process users in an organisation [58]. The essence of the method is to let process users rate factors believed to affect the SPI goal according to their viewpoints. The authors divide the process users into two groups, managers and engineers, based on their roles, and conclude that the groups differ on some factors, but not on all.

In [114], Svahnberg has studied how participants in an architecture assessment form groups when prioritising quality attributes and assessing architecture candidates. Svahnberg concludes that the role of the participant is the main influence when prioritising quality attributes, but not when assessing architecture candidates.

Questionnaires are often used as instruments in Software Process Assessment, for example the SEI Maturity Questionnaire for CMM-based assessment and the BOOTSTRAP questionnaire [124]. Klappholz et al. have developed an assessment tool, ATSE, for assessing attitude towards and knowledge of the development process [60]. While questionnaires in process assessment commonly measure the effectiveness of an improvement programme, our questionnaire was rather a part of the preparations for an upcoming improvement programme. In other words, it was not a substitute for a process assessment questionnaire.

3.3 Design

In this section, we outline the design of the study. We describe the contents of the questionnaire, the sampling and response rate, the technique for treating missing data, roles in the study and finally threats to the validity of the study.

3.3.1 Questionnaire Design

In order to assess process users' viewpoints and priorities, we designed a questionnaire consisting of six questions collecting data about the current state of the architecture documentation (the *infrastructure questions*), and one question about how to improve the architecture documentation (the *improvement question*). There was also a question asking about the process user's organisational role.

The infrastructure questions, which we believed would yield different results for different roles, were formulated as follows (words in bold are keywords used for identifying the questions later):

1. "In your opinion, to what extent does architecture documentation **exist**?"
2. "How would you, in general, describe the **form** of the architecture documentation?"
3. "How would you judge the **quality** of the documentation?"
4. "In your opinion, to what extent is architecture documentation **updated** as the system evolves?"
5. "In your opinion, how well does the architecture documentation **match** the actual architecture?"
6. "Imagine being a newly hired employee – how easy would it be to gain **insight** into the system using the current architecture documentation?"

For these questions, five point Likert scales (i.e., with five response options for each question) were used. Such scales are ordinal and are often used for collecting degrees of agreement [98]. For almost all questions, a low score indicated a negative response (little, low, seldom), while a high score indicated a positive response (much, high, often). The scale for the second question was slightly different from the others, though, in that a low score indicated text-orientation, while a high score indicated model-orientation.

The improvement question was formulated as follows: "If architecture documentation was to be improved in some way, what do you think is the most important purpose it could serve in addition to the present ones?" Five predefined purposes were given: change impact analysis, risk analysis, cost analysis, driver for development and system insight. In case these were not satisfactory, the respondent could choose "other" and suggest a new purpose as the most important one.

3.3.2 Sampling and Response Rate

The recipients of the questionnaire were selected using *systematic sampling* [98]. We obtained a list of all employees from an employee directory, and selected every second person on the list for the study. The reason for this was that another study was performed simultaneously and the employees were shared evenly between the studies.

The recipients of the questionnaire were given two weeks to fill it out and return the answers. The recipients were also allowed to reject the questionnaire if they had no time available or if they felt that it was not relevant for them. The two-week deadline seemed reasonable even for people with heavy workload. After one week, a reminder was sent to those who had not already responded or explicitly rejected the first distribution.

The population consisted of the 400 persons employed at the company. The selection of employees described above resulted in a sample of around 200 persons. While some responses were discarded because they contained invalid answers to some questions, around a third of the sample, or 65 persons, did give valid responses to the questionnaire. The other two thirds explicitly rejected the questionnaire, chose not to respond or were unable to respond before the deadline. Not all 65 respondents did answer all the questions, however. Because of that, some respondents had to be discarded, while some could be kept by imputing missing data in their answers, as described in the next section. As a result, the data presented in this chapter are based on answers from 58 respondents.

In order to verify that the respondents were representative for the population, we examined the departmental distribution. We could not use the role distribution, since the roles were known only for the respondents, not the entire population. We saw that there were only minor differences in departmental distribution between the respondents and the population, and consequently considered the respondents being representative for the population. In this context, it should also be noted that the roles are in many cases closely linked to departments (i.e., testers come from the test department and so forth). While the respondent drop-out is a threat to the external validity of the study, as discussed in Section 3.3.5, we consider the absolute size of the sample to be satisfactory.

3.3.3 Treatment of Missing Data

As implied in the previous section, some of the initial respondents did not answer all the questions discussed in this chapter. To be able to keep as many data points as possible, missing answers for those respondents that had answered at least five of the seven questions were imputed. The respondents that had answered only four questions or less were discarded, leaving 40 complete cases and 18 cases to impute. Only the complete cases were used as basis for the imputation.

The answers were imputed using the hot-deck k-Nearest Neighbour imputation technique, which imputes values based on the *k* cases most similar to the target case [33]. We have found this imputation technique to be suitable for Likert data (see Chapter 6). The similarity metric used was the Euclidean distance calculated only for the infrastructure questions (the improvement question was disregarded because of its nominal scale). As a replacement for a missing data value, the median of the *k* nearest cases was used for the infrastructure questions, while the mode was used for the improvement question. Based on a recommendation by Duda and Hart [33] (see also the supporting k-NN evaluation in Chapter 6), the value of *k* was chosen to 7, which is approximately the square root of the number of complete cases ($k = 6$, while closer, would be unsuitable when calculating the median). In the few cases when the mode was not unique at $k = 7$, *k* was increased to 8 instead.

3.3.4 Roles

The organisational roles of the respondents are rather specific for the company. Thus, in order to increase the generalisability of the results, and the replicability of the study, we have mapped the organisational roles to traditional software engineering roles. We mean that these represent the parts of software development normally discussed in software engineering literature [110]. To ensure the relevance of the resulting role list, the mapping was carried out together with a company-appointed specialist. The resulting roles, with abbreviated form and number of respondents within parentheses, are:

- **Designer** (D, 6) – creates system descriptions that the programmer can base the implementation on [89].
- **Programmer** (P, 8) – writes code that implements the requirements [89].
- **Tester** (T, 13) – catches faults that the programmer overlooks [89].
- **Functional manager** (FM, 5) – responsible for staffing, organising, and executing project tasks within their functional areas [85].
- **Project manager** (PRJ, 8) – plans, directs and integrates the work efforts of participants to achieve project goals [85].
- **Product manager** (PRD, 4) – responsible for product related planning activities [72].

- **Architect** (A, 11) – makes decisions, coordinates, manages dependencies, negotiates requirements, recommends technology etc. [42, 85]
- **Process group member** (PGM, 3) – facilitates the definition, maintenance and improvement of the software processes used by the organisation [56].

The process group member role is derived from Humphrey's Software Engineering Process Group (SEPG) [56]. Persons who work with infrastructure, which is seen as a kind of process support, are also included in this role. The architect role stems from an organisational role with responsibility for the system as a whole and its survival in the product chain. Architecture work is a part of this role, but implementation and low-level design is not.

3.3.5 Validity Threats

In this section, the most important threats to the validity of the study are presented together with measures that have been taken to avoid them.

Construct Validity. Construct validity is concerned with the design of the main study instrument (i.e., the questionnaire) and that it measures what it is intended to measure [98]. A threat to construct validity is that not all respondents may have had the same perception of what architecture is. In order to avoid that, a common definition of the term "software architecture" was given in the questionnaire, and each respondent was given the opportunity to disagree and give his or her own definition.

To counter the threat that some questions would be easier to answer for persons with certain roles, the questionnaire was checked by a screening group at the company. Because several roles (product manager, project manager, architect, designer and programmer) were represented by the persons in the screening group, there should be little bias towards any particular role. The screening group also verified that the role list in the questionnaire did contain relevant organisational roles.

External Validity. External validity is concerned with the generalisability of the results [122]. The main threat to external validity is that differences (and non-differences) found among roles could be true only for the studied company. Since we are dealing with just one case, this threat cannot be ruled out. There are, however, two circumstances that we believe make our findings interesting in a wider context. First, the studied company operates, as stated, on the world market with several large, international customers. As such, it should be considered a strong industrial case. Second, by mapping the organisational roles to traditional software engineering roles, generalisability to a wider software engineering community, and greater replicability of the study, has been strived for.

Another threat to external validity is the fact that we cannot guarantee that the distribution of roles for the respondents equals the distribution of roles in the population. Since we do not know the roles of all the persons in the population, we cannot avoid this threat. We have tried to ensure that the respondents are representative of the population by looking at the departmental distribution instead.

Internal Validity. Internal validity is concerned with the relationship between the treatment (i.e., the questionnaire) and the outcome (i.e., the results collected) [122]. The imputation of data described in Section 3.3.3 is a threat, since it essentially is fabrication of data. However, of the incomplete cases, almost all had only one question out of seven unanswered. Only one incomplete case had two questions unanswered. Furthermore, we have evaluated the chosen imputation technique and seen that it performs well with the type of data used in this study (see Chapter 6).

Conclusion Validity. Conclusion validity is concerned with the statistical relationship between the treatment and the outcome [122]. A threat to conclusion validity is that we, in the data analysis, use the Kruskal-Wallis test when the number of ties in the data is large (see Section 3.5). To address this, we apply an additional Chi-Square test to validate the outcome of the Kruskal-Wallis test. It should also be noted that the Kruskal-Wallis test includes correction for ties. Thus, the effect of this threat should be minimal.

Another threat to conclusion validity is that we use the Chi-Square test when the expected frequencies in the cells are relatively small. While this may mean that the outcome is not reliable, Siegel and Castellan acknowledge that for a large number of cells, the expected frequencies should be allowed to be small [106]. In our case, the number of cells is nearly 50, which they hint should be enough to allow small expected frequencies. Thus, we believe that the effect of this threat should be small.

3.4 Results

In this section, we present the results on the questionnaire that was distributed at the company. We show the distribution of answers for the infrastructure questions as well as for the improvement question. The results are analysed statistically in Section 3.5.

3.4.1 Infrastructure Questions

Figure 3.1 shows, for each of the infrastructure questions, how the answers are distributed on the response options. The exact phrasing of the questions can be found in Section 3.3.1. The question about to what extent documentation exists is special in the sense that the respondents made use of all five response options. As response options 3 and 4 account for 50% or more of the answers from all roles, the agreement among roles can be seen as large. Internally, architects (A) and designers (D) disagree more than the other roles, whereas on the whole, there is a consensus that documentation exists to a large extent.

For the question about documentation form, it is apparent that response option 2 dominates the answers (i.e., accounts for 50% or more) from all roles except the architects, meaning that most roles consider the documentation to be more text-oriented than model-oriented.

The answers to the question about how well the documentation matches the system indicate that all roles consider the match to be medium to good (as response options 3 and 4 dominate the answers). Again, architects have greater internal disagreement than the other roles. Also, functional managers (FM) and product managers (PRD) have a slightly more positive view than the other roles.

The question about quality of documentation has the narrowest distribution of answers of all the infrastructure questions. The respondents only made use of three of the response options in total, while most roles only made use of two, namely 3 and 4. Hence, the roles seem to agree that the documentation quality is medium or slightly above medium. The designers have a somewhat more positive view than other roles, whereas testers and programmers stand out because some of them consider the quality to be below medium.

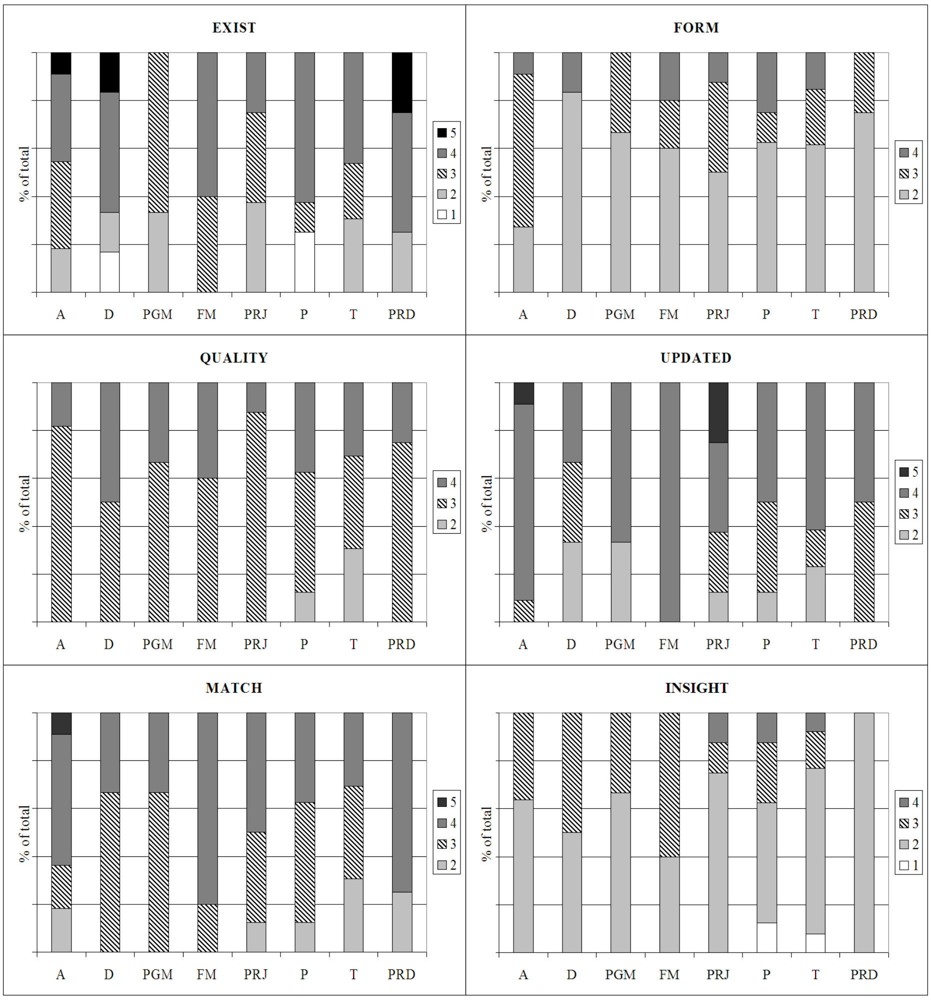


Figure 3.1 **Infrastructure Questions - Answer Distribution (*y axis*) per Role (*x axis*)**

In the answers to the question about to what extent documentation is updated, there seem to be a general consensus that the update frequency is high, with architects and functional managers having the most positive views. For this question, project managers (PRJ) have larger internal disagreement than the other roles.

Finally, for the question about how easy it is to gain insight into the system using the documentation, response option 2 dominates the answers from all roles except designers (where it is tied with response option 3) and functional managers. This means that there is agreement among the roles also for this question. Here, both programmers (P) and testers (T) have more internal disagreement than the other roles.

3.4.2 Improvement Question

When answering the improvement question, the respondents could choose “other” and add a new purpose if the predefined ones were not satisfactory (see Section 3.3.1). However, only one of the respondents chose to do this. We did not ask the other respondents to reconsider their answers with this additional purpose in mind for two reasons: (1) we estimated the added value to be minimal, and (2) we anticipated that it would be difficult to get new answers from all respondents. Consequently, only the predefined purposes are included in the analysis.

Table 3.1 **Most Important Improvement – Answer Distribution Among Roles**

Role	IA	RA	CA	DD	SI
Architect	27.3	0.0	9.1	27.3	36.4
Designer	**66.7**	0.0	0.0	0.0	33.3
Process group member	33.3	0.0	0.0	0.0	66.7
Functional manager	20.0	0.0	0.0	0.0	60.0
Project manager	0.0	0.0	12.5	37.5	50.0
Programmer	25.0	0.0	0.0	25.0	50.0
Tester	0.0	7.7	15.4	0.0	76.9
Product manager	0.0	**25.0**	**50.0**	25.0	**0.0**

Table 3.1 shows the results from the improvement question. An ocular inspection reveals a couple of interesting differences (shown in the table as shaded cells with bold text). First, system insight (SI) was frequently specified by all roles except the product managers. Second, risk analysis (RA) and in particular cost analysis (CA) were more frequent for the product manager role than for any other role. In fact, risk analysis was not specified at all by most roles except product managers and testers. Third, change impact analysis (IA) was considerably more frequent for designers than for any other role.

3.5 Analysis

In this section, the results from the questionnaire are analysed statistically. The null hypothesis stated in Section 3.1 is tested at significance level $\alpha = 0.05$. Two statistical tests are used, the Kruskal-Wallis test and the Chi-Square test [106]. We use these tests because we consider the data, being on ordinal and nominal scales, respectively, unsuitable for parametric tests. Both tests are applied to the infrastructure questions, whereas only Chi-Square is applied to the improvement question, because of its nominal scale.

Table 3.2 Kruskal-Wallis (*left*) and Chi-Square (*right*) Outcome, All Questions

Question	H (K-W)	p (df=7)	x^2	df	p
Exists	5.02	0.66	31.04	28	0.32
Form	4.38	0.73	12.46	14	0.57
Quality	3.62	0.82	16.41	14	0.29
Update	9.83	0.20	24.14	21	0.29
Match	5.88	0.55	17.56	21	0.68
Insight	4.12	0.77	14.91	21	0.83
Improve	N/A	N/A	47.71	28	0.046

3.5.1 Infrastructure Questions

The statistical significances of the results for the infrastructure questions are first calculated using the Kruskal-Wallis test. As can be seen in the second and third columns in Table 3.2, the results from the infrastructure questions are not significant at the selected significance level, as p exceeds 0.05 for all questions. This outcome aligns well with the results from the infrastructure questions presented in Section 3.4.1, where it can be seen that there is much agreement among the roles.

Since the Kruskal-Wallis test should be used with care when the number of ties in the data is large (as in our case) [46], we use the Chi-Square test for the infrastructure questions as well. The outcome of this test, presented in the rightmost three columns in Table 3.2, further confirms that there are no statistical significances in the results. This means that the null hypothesis cannot be rejected for the infrastructure questions.

3.5.2 Improvement Question

Because the data collected from the improvement question are nominal, we apply only the Chi-Square test and not the Kruskal-Wallis test. The outcome, presented in the bottom row in Table 3.2, shows that there is a statistically significant difference among the roles, as *p* is less than 0.05. Thus, the null hypothesis can be rejected in favour of the alternative hypothesis for the improvement question.

The overall Chi-Square does not pinpoint the differences. In other words, it does not identify exactly which roles that differ from others. To find the exact locations of the differences, the significances of all the partitions in the data are calculated. Each partition represents one pair of role and purpose (i.e., most important improvement), and has one degree of freedom. Simply speaking, the partition significance for a particular role-purpose pair is calculated based on the score of the pair and all scores to the left and above it. Consequently, the a priori order of the rows and columns affects the outcome of the partition significances [106]. We deal with this problem by performing an exhaustive calculation where all possible permutations of row and column order are evaluated. We argue that the pairs of role and purpose that are significant in more than 50% of the permutations can be considered significant overall. Table 3.3 shows the resulting pairs of role and purpose.

It can be seen that product managers differ from other roles in that they more frequently chose risk analysis and cost analysis. Moreover, designers differ in that they more frequently chose change impact analysis than other roles. We see that this aligns well with some of the observations made in Section 3.4.2.

Table 3.3 Significant Role-Purpose Pairs

Role	Purpose	Sig. frequency (%)
Product manager	Risk analysis	53.5
Product manager	Cost analysis	54.7
Designer	Change impact analysis	59.1

3.6 Discussion

In Section 3.4.1, we have seen that there seems to be much agreement among the roles regarding the infrastructure questions. This is supported by the statistical analysis in the previous section, which shows that there are no statistical differences among the roles for these questions. In other words, the respondents have fairly similar viewpoints. Our interpretation of this is that the architecture documentation process is well established at the company and that everyone has a joint understanding of how the architecture is documented and what the state of the documentation is. This is an important starting point when doing architecture-related process improvement, because it makes it easier to obtain a baseline of the current process. If the viewpoints had differed among roles, it could be difficult to find a common baseline.

Looking at individual roles, the results in Section 3.4.1 shows that some roles have more internal disagreement than other roles. This is true for architects and designers on the question about to what extent architecture documentation exists, and also for architects on the question about to what extent the documentation matches the system. The reason that there is disagreement within both roles may be that people with these roles work closer to the documentation and are therefore more sensitive to variations in its state. Moreover, internal disagreement is also noticeable for project managers on the question about to what extent the documentation is updated. A reason may be that project managers are more dependent on documentation update frequency when working with time and resource allocation. Finally, programmers and testers have larger internal disagreement than other roles for the question about how easy it is to gain insight into the system through the documentation. An explanation for this can be that these roles use the documentation for gaining system insight more than the other roles, and are therefore more sensitive to its ability to provide insight.

The results in Section 3.4.2 clearly show differences among the roles for the improvement question. System insight is the top improvement for all roles except designers and product managers. For these roles, change impact analysis and cost analysis are most important, respectively. The product manager role also stands out because it is the only role with strong focus on risk and cost analysis, and no focus at all on system insight. The reason that the product manager role differs on several accounts may be that this role has more market focus and less development focus than the other roles. The reason that the designer role differs may be that designers depend on the information in the architecture documentation more than other roles, and that this requires a strong focus on change impact analysis in order to handle ever-changing requirements.

The statistical analysis of the improvement question supports the differences among the roles outlined above. More specifically, designers' focus on change impact analysis and product managers' focus on cost and risk analysis are statistically significant. The fact that product managers do not consider system insight an important improvement is not significant, however. The existence of differences among the roles for the improvement question indicates that the priorities of respondents are different. This means that it becomes more difficult to create an improvement plan that satisfies all process users, since the plan needs to have a wider scope.

3.7 Conclusions

In this chapter, we have presented results from a case study where we used a questionnaire for investigating process users' viewpoints and priorities regarding the architecture documentation process. The study was conducted at an office of Ericsson, with the objective of preparing improvement of the architecture documentation process at the company by examining process users' viewpoints and priorities with respect to their roles.

A large number of employees were asked questions about the current state of the architecture documentation and possible improvements of it. As explained in Section 3.1.1, we asked about the product (architecture documentation) rather than the process in order to have more tangible questions. The process users were divided into groups according to their software engineering roles. In order to analyse the results statistically, the following hypotheses were evaluated:

- $\mathbf{H_0}$: There are no differences among roles.
- $\mathbf{H_A}$: There is a difference among roles.

The two types of questions (current state and improvement) in the questionnaire were analysed separately. When analysing the results from the questions about the current state of the process, the null hypothesis could not be rejected. Our interpretation of this is that the company has succeeded in spreading knowledge about the process to the process users. However, some roles have larger internal disagreement compared to other roles for some of the question.

This indicates that there may be underlying factors, other than role, that affect viewpoints.

When analysing the results from the improvement question, on the other hand, the null hypothesis could be rejected in favour of the alternative hypothesis. By performing a more in-depth statistical test and investigating the results from the improvement question directly, we found the following:

- System insight is not considered important to improve by the product managers. It is, however, the most important improvement for all other roles except the designers.
- Cost analysis and risk analysis are significantly more important to improve for product managers than for other roles. The reason may be a stronger market focus for this role than for the other roles.
- Change impact analysis is significantly more important to improve for designers than for other roles. The reason may be that designers use the documentation more than other roles when determining change impact.

Initially, we expected differences among roles both for viewpoints and priorities, since stakeholders of architecture documentation often are considered to have different needs and knowledge. Our expectations were however only fulfilled for priorities, as these clearly were different among the roles. It could be argued that it is "common knowledge" that priorities differ, which is why our expectations were set as they were. However, it remains difficult to foresee exactly how roles differ, which is important to know in process improvement. Furthermore, we had expected larger differences than were actually found. In any case, the fact that viewpoints, contrary to our expectations, did not differ while priorities did, leads us to conclude the following:

- It is important to cover process users' viewpoints and priorities in process improvement work, because they may not coincide with the prevalent expectations.
- Further research should investigate in which situations roles are likely to differ and in which they are likely to be similar.

CHAPTER

4

Understanding Impact Analysis: An Empirical Study to Capture Knowledge on Different Organisational Levels

Intl. Conference on Software Engineering and Knowledge Engineering, 2005
(submitted)

Per Jönsson and Claes Wohlin

Change impact analysis (IA) is a crucial part of change management and requirements engineering (RE), as all software are exposed to changing requirements. Bohner and Arnold define impact analysis as "...identifying the potential consequences of a change, or estimating what needs to modified to accomplish a change." [14]

Research about impact analysis is commonly found in the field of software maintenance (see Section 1.2.2 in Chapter 1), although IA doubtlessly plays an important role in the entire product cycle. For example, Lindvall coined *requirements-driven impact analysis* to denote the activity of identifying the impact of new requirements on an existing system [73].

The gross of IA research concerns the development of methods and algorithms for supporting and automating the analysis, or the adaptation of existing methods in new contexts. To our knowledge, there is little research about more non-technical aspects of the subject, such as process and organisational aspects. In our experience, IA is, as a part of the change management process, heavily dependent on organisational support and stakeholder views.

In this chapter, we present an empirical study of the views of IA on different organisational levels. In exploring the uses (application areas) and issues of IA at a software development company, we identified three levels with different foci: one with technical focus, one with resource focus and one with product focus. Looking at management science, we found that these levels mapped well to the decision-making model originally defined by Anthony (see, for example, [4] or [84]), where decisions are categorised as *operative*, *tactical* or *strategic*.

In order to understand how potential issues and uses associated with IA are seen on the three organisational levels, we interviewed 18 employees at the company mentioned above, in their roles as industrial experts. Our hypothesis is that people on different organisational levels see IA differently, and consequently have little awareness of issues and uses on other levels.

Gathering knowledge is an important step towards being able to overcome differences. Therefore, our contribution does not only lie in the study of an organisational aspect of IA, but also in the systematic method for collecting, extracting and prioritising knowledge pertaining to issues and uses of IA.

The chapter is structured as follows. Section 4.1 covers related work. Section 4.2 describes the design of the study, whereas Section 4.3 details how the study was carried out. Section 4.4 presents results, which are subsequently analysed and discussed in Section 4.5. Finally, conclusions are drawn in Section 4.6.

4.1 Related Work

Aurum and Wohlin tie RE activities to decision-making models, arguing that RE is a decision-intensive process [4]. They suggest that studying decision-making within RE helps organisations structure their RE decisions better and, ultimately, produce software with higher quality. We mean that the same argument holds for IA, due to the strong connection between IA and RE.

Several researchers report on differences between managers and engineers in the context of software process improvement (SPI), for example concerning views of quality [51], use of formal routines to transfer knowledge and experience [29] and how they rate factors affecting an SPI goal [58]. These examples demonstrate the relevance in studying different organisational levels.

Some uses of IA are mentioned in the literature, for example estimating resource needs, assessing system impact, weighing change proposals against each other and finding the overlap of parallel changes [14, 108]. Issues are mentioned as well, for example lack of automation and tools, insufficient traceability, documentation that is not updated, inconsistent models and high time-consumption [14, 27]. In this chapter, we explicitly focus on both uses and issues.

4.2 Method

This section describes our research setting, introduces the three organisational levels and presents our method for carrying out the study. The method consists of three main steps: (1) interviews with employees, (2) results triangulation and filtering, and (3) prioritisation of the results.

4.2.1 Research Setting

The study was conducted at Ericsson, whose organisation can be characterised as a matrix organisation, where functional areas and projects are separated [85]. Whenever a project is launched, project managers negotiate with functional managers in order to borrow resources for a limited period of time. A matrix organisation allows for a balanced utilisation of resources and fosters exchange of knowledge and experience between workers belonging to the same functional area. It may, however, also induce conflict due to the fact that workers belong both to a functional organisation and a project organisation at the same time, and consequently have two different managers.

While the study was conducted at a specific company, the population we wish to generalise to is industrial software development experts in general rather than just within the company.

4.2.2 Organisational Levels

As mentioned earlier, the three organisational levels we identified map well to the decision-making model defined by Anthony. The model differentiates between decisions at three levels as follows [84]:

- Strategic decisions have typically large scope, large impact and long-term perspective. They concern organisational or product-related goals and strategies.
- Tactical decisions concern planning of time and resources to reach strategic goals, and are often made by middle management. They have smaller scope and impact, and shorter time horizon, than strategic decisions.
- Operative decisions are made when realising the project according to the plan, and are often of technical nature.

4.2.3 Interview Design

The interview instrument contained seven main topics, of which each was associated with one or more open questions. The instrument in its entirety can be obtained from the authors by request. In this chapter, we focus on two of the main topics: potential issues and uses. The questions for these topics were as follows (translated from Swedish):

- **Issues**: Which potential issues are associated with performing impact analysis?
- **Uses**: Which potential uses does impact analysis have?

Note that we asked about potential issues and uses, rather than actual ones. The reason for this was to avoid limiting the generalisability of the results by extracting company-specific issues and uses only.

The remaining topics were more qualitative in their nature, and were intended both for providing a context for and for collecting hidden or implicit knowledge about the issues and uses. We did not intend for the participants to prioritise during the interviews, as we expected each of them to see only a subset of the possible issues and uses.

In order to ensure the appropriateness and clarity of the questions, the interview instrument was developed in close cooperation with the company where the study was conducted.

4.2.4 Results Triangulation and Filtering

A triangulation and filtering scheme was designed in order to get as complete lists as possible of both issues and uses. The scheme involved three information sources:

- the lists generated in the interviews,
- qualitative information from the interviews, and
- information from the literature.

By using information from all interview topics, it would be possible to extract both explicit and implicit knowledge about issues and uses, and by collecting information from the literature, we would be able to add issues and uses of which the participants were not aware.

The filtering part was intended to remove redundancies and inconsistencies in the lists by merging similar items together, and by discarding items that were not directly related to IA.

4.2.5 Prioritisation

In order to get prioritised lists of both issues and uses, a post-test was designed as a follow-up to the interviews. In the post-test, the participants should state the distribution of their decisions on the decision levels. Based on this, it would be possible to deduce their organisational levels. For example, a participant making mostly strategic decisions would be regarded as belonging to the strategic organisational level. This scheme was used since, as Aurum and Wohlin point out [4], Anthony's decision levels are not entirely orthogonal.

For uses, the participants should prioritise such that the use with the highest priority would be the one most relevant to the organisation if it was realised. For issues, the participants should prioritise such that the issue with the highest priority would be the one most critical to the organisation if it existed.

We chose the organisational perspective based on our initial hypothesis. However, we were also interested in knowing if the participants would prioritise differently from an individual perspective. Trying to maintain a balance between collecting much information and keeping the post-test short, we added the individual perspective to the prioritisation of issues only.

To account for the problem that the first prioritisation of issues could affect the second (due to maturation issues), a two group design was used for the post-test, such that half of the participants should prioritise from the organisational perspective first, and the other half from the individual perspective first.

When prioritising both issues and uses, the participants should assign weights to the items, such that the weights should sum to 1 000. Thus, each weight could be seen as a certain percentage of the total importance (i.e., criticality for issues and relevance for uses) of all items. This is also known as the *hundred-dollar* method [70]. Advantages of this method are that it is easy to learn and use, and that the resulting weights are on a ratio scale.

4.3 Operation

In this section, we describe relevant parts of the operation of the study based on the design presented in the previous section.

4.3.1 Organisational Levels

We did not know the organisational levels of the participants prior to the interviews, since we chose not to map roles to levels directly. During the interview round, we let a manager estimate the levels of the participants, based on the descriptions of Anthony's decision levels as provided by Ngo-The and Ruhe [84]. The estimated levels were used to determine when it was likely that enough interviews had been performed to cover all levels. It was in the interest of the company to keep the number of interviews down to save resources.

The actual level used for a participant was determined by looking at the estimated and reported (i.e., from the post-test) levels, and, if necessary, by matching the participant's work tasks to Anthony's decision levels. As an example, consider a person with mismatching reported and estimated levels. If there is another person with the same role but matching levels, use his or her level for the first person. Otherwise, deduce the level by matching the first person's work tasks to the decision levels.

It was necessary to match work tasks to the decision levels for four of the participants. To increase the certainty in the level assignment, we successfully validated the levels of the remaining participants by matching their work tasks to the decision levels as well.

4.3.2 Interviews

A pilot interview was conducted at the company in order to measure the interview time and find discrepancies, if any, in the interview instrument. Since it resulted in only minor modifications to the interview instrument, the pilot was included in the analysis.

18 interviews were conducted, including the pilot, in the course of one month. The participants were sampled using convenience sampling [98], which in practise means that they were selected based on accessibility and recommendations from people at the company. We sampled based on convenience for two main reasons. First, we did not know prior to the interview which organisational level a person belonged to, and could consequently not sample based on that. Second, we argued that a person would be more committed to participate if he or she had been recommended by someone else.

The interviews were semi-structured, meaning that it was not necessary to follow the predefined question order strictly, and that the wording of questions was not seen as crucial for the outcome of the interviews [98]. The participant could speak rather freely, but the interviewer made sure that all questions were answered in one way or another.

The participants were asked if they would accept receiving and answering a post-test where they should prioritise both issues and uses. This was done in order to prepare the participants and increase their commitment towards the post-test.

A great variety of roles were covered in the interviews, including developer, tester, technical coordinator, manager (functional, product and project) and system architect.

4.3.3 Prioritisation

Based on the complete lists of issues and uses, we constructed a post-test, as described in Section 4.2.5. The purpose of the post-test was to let the participants prioritise issues (from two different perspectives) and uses. To avoid maturation effects in the prioritisation of issues, we divided the participants into two groups based on their estimated organisational levels. Persons on each level were split at random between the two groups.

We required that the participants should specify a unique (non-tied) share for the level of their principal decisions. This way, we could deduce a non-ambiguous organisational level.

Since we were interested in potential issues and uses only, we asked the participants to prioritise without regard to actual issues and uses. In other words, we wanted to avoid priorities biased towards actual issues and uses.

4.4 Results

The mapping of participants to the organisational levels (described in Section 4.3.1) resulted in eight participants in the operative level, five in the tactical level and five in the strategic level.

The interviews resulted in 18 uses after irrelevant uses were removed and similar uses were merged together. These were subsequently combined with 11 uses found in the literature, which, due to overlap, resulted in 20 uses in total. For issues, there were 25 coming from the interview data. We found six issues mentioned in the literature, but these were already among the 25. Thus, the resulting list contained 25 issues in total.

For brevity, we do not show all issues and uses here. The list below contains only the issues (prefix *i*) and uses (prefix *u*) that are relevant for the analysis (see Section 4.5). Lists of all issues and uses can be found in the end of this chapter.

- **i1**: Hard to get resources for performing IA
- **i2**: Lack of time for performing IA
- **i3**: System impact is underestimated or overlooked
- **i4**: Unclear change requests
- **i6**: Analyses are incomplete or delayed

- **i8**: Analyses are too coarse or uncertain
- **i14**: Affected parties are overlooked
- **i15**: Analyses are performed by the wrong persons
- **i16**: Interest-based change request decisions
- **i19**: Not possible to see change request outcome
- **i22**: Cheap, short-term solutions win over good, long-term solutions
- **i23**: High levels specify solutions with too much detail
- **i24**: Hardware and protocol dependencies are difficult to handle for late change requests
- **i25**: Missing relevant structure and documentation to support the analysis
- **u1**: Planning the project with respect to time and cost
- **u2**: Determining cost versus benefit
- **u3**: Deciding whether to accept or reject the change
- **u6**: Understanding technical and market consequences of including or not including the change
- **u8**: Understanding the proposed change
- **u13**: Assessing system impact
- **u14**: Obtaining a new or changed requirements baseline
- **u20**: Revealing synergies and conflicts between change proposals

4.4.1 Threats to Validity

In this section, the most important threats to the validity of the study are presented and discussed.

External validity is concerned with the generalisability of the results [122]. The small sample size and the fact that we used convenience sampling are threats to this type of validity. Nevertheless, as the participants were selected based on recommendations, we believe they were good representatives of their respective organisational levels. Furthermore, the fact that we focused on potential issues and uses rather than actual ones should increase the external validity. Also, the participants covered all uses but two and all issues from the literature, which indicates that their views of IA were not company-specific.

Construct validity is concerned with the design of the main study instrument and that it measures what it is intended to measure [98]. A threat to this type of validity is that the participants may not have had the desired mindset when prioritising items. As stated in Section 4.3.3, we asked the participants to prioritise as if neither issues nor uses currently were present, but we could not verify if they adhered to our request.

Internal validity is concerned with the relationship between the treatment and the outcome [122]. The assignment of participants to organisational levels is a threat to this type of validity. We tried to minimise the threat by basing the assignment on several information sources (see Section 4.3.1). The use of an external source (matching work tasks to decision levels) also strengthens the external validity.

4.5 Analysis and Discussion

This section describes two separate data analyses, one qualitative and one quantitative (statistical). We also comment briefly on the most interesting results.

4.5.1 Qualitative Analysis

In the qualitative analysis, we studied two aspects of the prioritised lists of potential issues and uses. First, we looked at the top five placements (which could include more than five items due to tied priorities) for each organisational level, in order to see if there was agreement on the most important items among the levels. Table 4.1 shows the top five placements for issues from an individual perspective (left), issues from an organisational perspective (middle) and uses (right). Top issues and uses common for two or more levels are displayed in bold text. It is clear that there was much agreement on the most important issues (regardless of perspective) and uses. Note that no single issue was considered important by all three levels, while two uses were (u1 and u3). Moreover, the two issue perspectives differed somewhat. For example, issue i4 (unclear change requests) was seen as the most critical issue by the operative level from an individual perspective, but was not among the top five from an organisational perspective.

Second, we compared the top five placements in each level with the bottom five in the other levels. The intention was to find items that were considered important by one level but unimportant by another level. None of the uses matched this criterion, and only a few issues from both perspectives did. This concurs with the observation that there was much agreement among the levels. Table 4.2 shows the ranks of the issues for each level, with negative ranks counting from the end. For example, issue i16 (interest-based change request decisions) had the fourth highest priority in the operative level, but the fourth lowest in the tactical level. In the strategic level, it had neither a top five nor bottom five placement (empty cell).

Table 4.1 **Top Five Placements for Issues and Uses**

Issues, ind.			Issues, org.			Uses		
Operative	**Tactical**	**Strategic**	**Operative**	**Tactical**	**Strategic**	**Operative**	**Tactical**	**Strategic**
i4	i24	**i15**	**i3**	**i14**	**i6**	**u3**	**u3**	**u1**
i22	**i14**	**i1**	i22	**i3**	**i1/** i15/ i23	**u8**	**u8**	**u3**
i14	**i3/ i1/** i19	**i3**	**i14**	**i1**		u13	**u1**	u14
i15		i2	i16	i4	i24/ i25	**u2**	**u6,** u20	**u6**
i8		i6	i15	**i6**		**u1**		**u2**

Table 4.2 **Top Five vs. Bottom Five Issues**

Perspective	Issue	Operative	Tactical	Strategic
Organisational	i16	4	-4	
Organisational	i25	-5		5
Individual	i19	-2	3	-1

4.5.2 Quantitative Analysis

In the quantitative analysis, we tested each issue and use for departure from normality by using the Shapiro-Wilk test for normality. The test showed that the data in general did not have a normal distribution. Therefore, we used the non-parametric Kruskal-Wallis test for further analysis. The Kruskal-Wallis test is a non-parametric alternative to ANOVA, and should be used when there are more than two independent groups [106]. At a significance level of 0.05, the test showed the following:

- There were neither significant differences among the three organisational levels for uses nor issues from an organisational perspective.
- There were significant differences among the levels for two of the issues from an individual perspective. These are displayed in Table 4.3.

Table 4.3 Kruskal-Wallis for Issues i4 and i19

Issue	H	Sig.
i4	6.934	0.031
i19	9.530	0.009

Outliers were not removed due to the relatively small sample size and the fact that there were many items to prioritise. We did, however, verify that the outliers for issues i4 and i19 were not responsible for the significant differences.

Figure 4.1 shows box plots for issues i4 and i19 (from an individual perspective). As can be seen, there was a larger spread in the operative level for issue i4 than for the other levels. Similarly, the spread in the tactical level for issue i19 was larger than for the other levels. This reflects the fact that the majority of the weights given to these two issues came from participants at the operative and tactical levels, respectively.

4.5.3 Discussion

The two analyses both indicate that the participants, regardless of organisational level, had a coherent view of what IA could (or should) be used for. Top uses were planning the project with respect to time and cost (u1), deciding whether to accept or reject the change (u3) and understanding the proposed change (u8). We had, however, expected assessing system impact (u13) to have a top five placement as well.

Similarly, both analyses show that participants on different levels saw different issues as critical (individual perspective), whereas on the whole, the awareness of each other's potential issues was large (organisational perspective). We were surprised that issue i19 (not possible to see change request outcome) was only considered critical by the tactical level from an individual perspective.

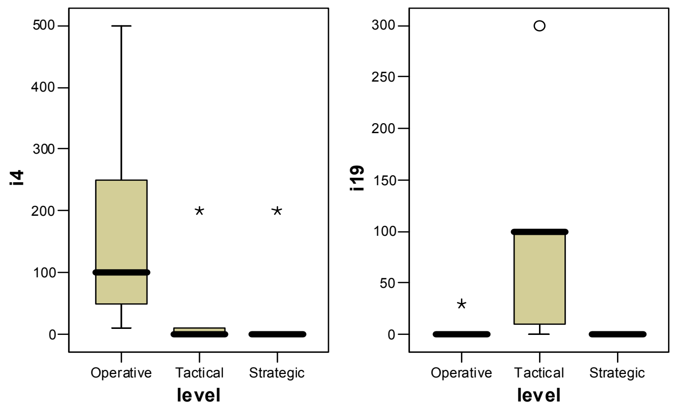


Figure 4.1 Box Plots for Issues i4 and i19, Individual Perspective

4.6 Conclusions

In this chapter, we have presented an empirical study of views of impact analysis (IA) at three different organisational levels: operative, tactical and strategic. In the study, we interviewed 18 employees, representing industrial experts, at a software development company in order to understand how potential issues and uses of IA are seen at the three levels.

The qualitative analysis shows that there was much agreement among the levels for both issues and uses. In other words, people at the different organisational levels seemed to view IA in the light of what was important for all levels, not just their own. This means that our initial hypothesis does not hold, as we expected the views to diverge. The qualitative analysis does, however, also show that there were minor differences among the levels with respect to issues, regardless of whether the participants prioritised from an individual or an organisational perspective. The statistical analysis reveals some differences for issues as well, but only for the individual perspective. Unclear change requests (issue i4) was seen as more critical by the operative level, which is reasonable given that people at this level are the ones responsible for implementing changes. Not possible to see change request outcome (issue i19) was seen as more critical by the tactical level, which may be related to a need for the ability to follow up estimates. The fact that there were some differences indicates that it is relevant to look at organisational levels when studying IA. We suggest that Anthony's decision-making model is a good basis for the levels.

Finally, we want to emphasise the importance of studying the non-technical aspects of IA in order to better understand how to use it successfully in change management. Our systematic method for extracting knowledge pertaining to potential issues and uses of IA proved to be successful in terms of providing relevant data.

Issues and Uses, Complete Lists

Complete lists of issues and uses are shown below. Table 4.1 contains all issues found during the interviews, and Table 4.2 contains all uses.

Table 4.1 All Issues

Id	Issue
i1	Hard to get resources for performing IA
i2	Lack of time for performing IA
i3	System impact is underestimated or overlooked
i4	Unclear change requests
i5	Responsibility and product/project balance are difficult in analyses that span several systems
i6	Analyses are incomplete or delayed
i7	Analyses require much expertise and experience
i8	Analyses are too coarse or uncertain
i9	Difficult to handle conflicting and synergetic change requests
i10	Analyses are not prevented from being disregarded
i11	Existing traceability is manual and cumbersome
i12	Difficult to see trends and statistics for collective impact
i13	Tools for supporting the analysis are missing
i14	Affected parties are overlooked
i15	Analyses are performed by the wrong persons
i16	Interest-based change request decisions
i17	Missing requirements and baseline for early changes
i18	Analyses and change implementation evoke stress
i19	Not possible to see change request outcome

Table 4.1 All Issues (Continued)

Id	Issue
i20	Difficult to see status and updates for a change request
i21	No method can handle all levels of change complexity
i22	Cheap, short-term solutions win over good, long-term solutions
i23	High levels specify solutions with too much detail
i24	Hardware and protocol dependencies are difficult to handle for late change requests
i25	Missing relevant structure and documentation to support the analysis

The following list shows all uses found during the interviews:

Table 4.2 All Uses

Id	Use
u1	Planning the project with respect to time and cost
u2	Determining cost versus benefit
u3	Deciding whether to accept or reject the change
u4	Identifying particularly change-prone system parts
u5	Identifying improvement potential for future releases
u6	Understanding technical and market consequences of including or not including the change
u7	Measuring the quality of the basis for analyses
u8	Understanding the proposed change
u9	Identifying affected product releases
u10	Finding test impact to simplify regression testing
u11	Assessing the feasibility of a change request
u12	Synchronising deliverables for testing
u13	Assessing system impact
u14	Obtaining a new or changed requirements baseline
u15	Identifying trends and statistics for collective change impact
u16	Updating maintenance and support plans
u17	Supporting measurement of actual impact
u18	Avoiding side effects and guaranteeing system stability
u19	Prioritising among change requests
u20	Revealing synergies and conflicts between change proposals

CHAPTER

5

Semi-Automatic Impact Analysis through Keyword-Based Relationships – A Feasibility Study

To be submitted

Per Jönsson and Claes Wohlin

Software development is an engineering activity in which many different kinds of artefacts are produced, refined, used and validated. These artefacts, such as requirements, architecture components, classes, source files, test cases and documents, are intertwined in a web of dependencies that can be complex even for small systems. Traceability, the ability to trace dependencies between artefacts [95], serves as a means to understand the web of dependencies and to use it to perform various kinds of analysis, such as impact analysis.

In this chapter, we use Lindvall's definition of impact analysis as "the identification of the set of software entities that need to be changed to implement a new requirement in an existing system." [73] Impact analysis is crucial both in software development, where it is used to assess the impact of new requirements as well as changes to requirements, and in software maintenance, where it is used to assess the impact of changes made as parts of maintenance activities. Arnold and Bohner gathered in 1996 much of the important research about impact analysis [14]. Later, Lindvall explored requirements-driven impact analysis for object-oriented systems, thereby focusing more on new requirements than on changing requirements [73].

A common way of performing impact analysis is to examine dependencies among development artefacts, both of the same type and of different types. A restraining factor is, however, that dependencies among artefacts are not always properly documented. For example, architecture and design models are sometimes only coarsely connected to requirements, thereby decreasing the precision of impact analysis and making it difficult to verify that the system fully implements all requirements. One possible way around this problem is to identify dependencies among artefacts by examining sources that implicitly convey information about relationships.

In this chapter, we present a feasibility study of a method for performing semi-automatic impact analysis based on lexicographic relationships between requirements and architecture component descriptions[1]. The method is intended to be used in software evolution, where systems have multiple succeeding releases, and each release inherits the architecture, and corresponding documentation, from the previous release.

Our hypothesis is that requirements and architecture component descriptions are written using the same technical language, which means that it is possible to estimate the impact of new requirements by finding the components whose descriptions contain keywords from the requirements. We do, however, not intend for the proposed method to provide final impact estimates, but rather to act as a decision-support system for impact analysis. Three usage scenarios are presented in Section 5.1.

The method consists of four main steps: *screen for relevance*, *identify keywords*, *identify dependencies* and *examine results and estimate impact*. The dependency identification step is entirely automatic, and uses the information retrieval technique *Latent Semantic Indexing* (LSI; also known as Latent Semantic Analysis) for identifying lexicographic

1. The *software architecture* is the basic structure of a system, consisting of software *components* and the relationships between them.

relationships. LSI provides a way to group similar documents based on their semantic (or conceptual) similarity rather than on their word-by-word similarity [30]. By looking at word structures, the technique has the ability to relate documents based on what they *should* contain rather than on what they *do* contain.

Our method differs from standard applications of LSI, and from approaches based on natural language processing (NLP), in that the identification of keywords is performed manually by someone with insight in the system and its domain. We argue that this is beneficial because the human brain still outperforms computer-based systems when it comes to understanding human language. Furthermore, written software development artefacts typically contain technical jargon and abbreviations, which may make them unsuitable for NLP-based methods.

In order to evaluate the method, we have applied it to an industrial software system with around 50 architecture components and around 400 requirements. As the architecture of the system is largely based on previous releases, its structure is established and documented. We compare the outcome of the method with existing, albeit coarse, impact predictions made by the developers early in the development phase. The comparison shows that the performance of the method, in terms of correspondence with the existing estimates, is not very impressive. There are, however, a number of method parameters that can be adjusted, and we have made a number of assumptions that may or may not be entirely appropriate. Therefore, we end the chapter by discussing future work of improving and further evaluating the method.

The chapter is structured as follows. Section 5.1 presents three usage scenarios of the method. Section 5.2 outlines related work, while Section 5.3 describes the steps of the method and the techniques used by it. The evaluation is presented in Section 5.4 and subsequently analysed in Section 5.5. Evaluation results and plans for future work are discussed in Section 5.6. Finally, a short summary is given in Section 5.7.

5.1 Usage Scenarios

As stated, our main goal is to support software engineers in their task of analysing the impact of new requirements. This is, however, only one possible application of the method. We see three possible scenarios:

- **Scenario 1**: Estimate the impact of a new requirement.
- **Scenario 2**: Estimate the impact of a change to a requirement.
- **Scenario 3**: Extract traceability of a legacy system.

The first scenario is the main scenario. When a new requirement appears, one challenge from a requirements engineering point-of-view is to make an early prediction of the impact of the requirement. Since several architecture components most likely need to include or handle the new functionality, it is not trivial to estimate the impact. During the implementation of the system, the developers can probably determine the impact with high accuracy, but before the implementation has started, other stakeholders may need more support when they are faced with the same task.

The second scenario is concerned with changes to requirements. Here, the method will estimate the impact of a requirement after it has been changed. By comparing this to the corresponding estimate from before the change, the impact difference can be calculated. The size of the impact difference may be an indicator of the scope of the change, although a zero difference need not mean that the architecture is unaffected; changes may still be necessary within the components that already implement the functionality stated in the requirement.

It should be noted that the method should not be seen as a replacement for documented traceability; if it is documented where requirements actually are implemented in the architecture, this information should be used rather than anything else to accurately determine change impact.

In the third scenario, the method is used to extract traceability from an existing, already developed system. This may be useful, for example, if the traceability has not been documented properly. If the architecture component descriptions are updated in accordance with the final architecture, the method should perform better than in the previous scenarios, where they may still be under development.

5.2 Related Work

Traceability and impact analysis are important in requirements engineering, since both new and changing requirements are significant causes for software change. When requirements change or new requirements appear, adequate traceability can provide a fair understanding of how the system is built, and accordingly, how it should be changed in the most optimal way. Traceability is a common means for performing impact analysis, something that is discussed more in detail in Chapter 2.

Lee provides a concise overview of NLP and the many inherent problems it has to face [67]. Ryan argues that the role of NLP in requirements engineering has been overstated, and that the belief that computer systems should be able to fully comprehend requirements specifications written in natural language is unrealistic [101]. He concludes that the role of NLP in future software development is one of supporting rather than one of replacing manually performed tasks. This corresponds well to our intention of using the proposed method as a decision-support system.

Natt och Dag et al. present a method for automatic similarity analysis of requirements written in natural language, based on the same underlying model as LSI [82, 83]. They observe that the method is suitable for identifying both requirements duplicates and interdependencies. While the method thus is promising, they emphasise the fact that this type of automatic analysis should not be allowed to replace human judgement, but should instead support it.

Etzkorn and Davis have studied source code comments and identifiers in legacy object-oriented systems in order to support reuse [40]. They state that source code comments are typically written in a subset of the natural language used by the developers. This facilitates the parsing process, as the grammar tends to be simpler and the vocabulary smaller, consisting of a limited set of words specific for the system. Thus, our hypothesis that software development artefacts are generally written in the same technical language seems to be appropriate.

LSI was originally proposed by Deerwester et al. [30] as a method for automatic indexing and information retrieval (see also the preceding paper by Furnas et al. [43]). The work presented in this chapter was largely inspired by Marcus and Maletic, who have used LSI to extract traceability between source code and documentation

[76]. Their approach is similar to ours, in that the words they use are based on a subset of all words in the source code, more specifically comment text and identifiers (e.g., variable names). They conclude that LSI performs well, and that it requires less processing of the source code, and thus less computation, than other information retrieval methods.

5.3 Method

In this section, we begin by discussing natural language versus technical language, in order to shed some light on the challenges for our method. Next, we describe fuzzy string matching, a technique for matching strings without requiring total letter-by-letter similarity. We also describe how LSI works. Finally, we present the steps of our proposed method.

5.3.1 Natural vs. Technical Language

According to ANSI/IEEE Std 830-1984, requirements are often written in natural language [3]. However, they also include, by necessity, technical terms and abbreviations that relate to the domain or system. Dressel means that technical language exposes far more structure than does natural language, which implies that it should be easier to understand and process [32]. In our effort to find a method for automatic processing of the mix of natural language and technical language found in requirements (as well as in architecture component descriptions), we observed the following:

- Common stop words (i.e., words without normal semantic meaning, such as FOR, OR and AND) can have significant technical meaning. For example, we have found the word ZERO in some stop lists, while in the Microsoft Windows operating system, this is probably a significant word in a requirement for the WIRELESS ZERO CONFIGURATION service.
- Words that are different forms of the same stem may have significantly different meaning, which means that *stemming* (i.e. reducing a word to its stem) may not be a viable option. For example, the Porter stemming algorithm (see, for example, [6] or [90]) would reduce both ACCUMULATED and ACCUMULATOR into ACCUMUL, whereas the two words may occur in requirements with completely different foci.
- Due to *polysemy* (i.e., the fact that many words have more than one distinct meaning), a word may appear as both non-technical

and technical in different contexts. For example, NUMBER as in NUMBER OF CALLS is likely less significant (for the meaning of a requirement) than as in MOBILE PHONE NUMBER.

- Both requirements and architecture components often contain technical abbreviations. These are particularly difficult to handle, both as they may be hard to identify as words (because they may contain non-word characters, such as punctuation), and as they sometimes occur spelled out.

Based on these observations, we have made certain assumptions regarding the processing of requirements and component descriptions. These assumptions are detailed further on.

5.3.2 Fuzzy String Matching

Several factors prevent direct comparison of words in language-based methods. One such factor has to do with word tenses, conjugation of verbs, singular and plural forms and so on. Usually, this is solved through stemming of words. By comparing stems instead of words, mismatches due to morphological variants of a word can be avoided.

Another factor is that people may misspell, especially those who write in another language than their native one. This is common in global enterprises, such as the one that has developed the system we use in the evaluation of the method.

One way of comparing words despite spelling mistakes is to employ *fuzzy string matching*. In its simplest form, fuzzy string matching is accomplished through the use of a string distance algorithm and a threshold value. Two words with a distance below the threshold are considered the same.

In our method, we use fuzzy string matching based on case-insensitive *edit distance* to account for both spelling errors and morphological variants. Edit distance is defined to be the minimum number of deletions, insertions and substitutions required to transform one word into another [107]. For example, the edit distance between RECEIVE and RECIEVE is 2, as two substitutions (I for E and E for I) are required to transform the former into the latter.

In other words, we do not use stemming in any form. Stemming is highly language-dependent (the Porter stemming algorithm, for example, does not even work properly with British English) and does not take into account misspellings. As pointed out before, normal stemming rules for English may not perform satisfactory with technical terms. Also, we are concerned that stemming may transform abbreviations into words without meaning. Furthermore, Berry et al. mention that LSI does not require stemming – if words with the same stem are used together in a document, LSI will identify and make use of that pattern [12].

5.3.3 Latent Semantic Indexing

This section outlines LSI on the conceptual level. For mathematical details and associated proofs, see, for example, [12].

LSI was introduced as a means for retrieving information from documents based or their conceptual contents rather than solely on the words contained within them [30]. The basic assumption is that documents expose a structure that makes it possible to relate them regardless of whether a term missing from a document is really missing or should have been there as it is a natural part of the conceptual contents of the document. As an example, consider two documents describing the theory of general relativity in great detail, whereas only one of them contains the word EINSTEIN. Now, someone who searches for documents about Einstein would likely consider both documents relevant for his or her query, not only the one actually containing the sought name. Here, LSI would detect that the two documents share a lot of words (related to the theory of general relativity) and therefore are semantically close to each other.

The starting point of LSI is a *term-by-document* (TBD) matrix, which is a rectangular matrix associating terms with documents. Each row corresponds to a specific term, and each column to a specific document. The occurrence of a term in a document is marked in the corresponding cell in the matrix.

In the general case, for example when the documents are web pages, all content words in all documents must appear as terms in the TBD matrix. Thus, it is necessary to extract all relevant terms from the documents, which can be done by going through the documents and removing all stop words. Furthermore, it is customary to remove words that occur in only one document (see, for example, [12] or [35]), while Kolda suggests that this is a matter of choice depending on the situation [62].

Each column in the TBD matrix is a representation of a document as a vector with as many dimensions as there are terms (i.e., each term occurrence is a vector coordinate). LSI uses by default a method called *Singular Value Decomposition* (SVD; although others are possible – see, for example, [62]), the purpose of which is to calculate an approximate representation of the original TBD matrix with fewer dimensions [12, 30, 65]. Basically, each document will be represented by a vector of semantic concepts rather than terms. The dimensional reduction means that vectors that initially are close to each other (i.e., that represent similar documents) will be merged together.

The raw outcome of SVD consists of three matrices, which together represent the approximation of the original TBD matrix. Based on these, it is possible to calculate the *cosine coefficient* of two documents, which corresponds to the similarity of the documents, ranging from a minimum of 0 to a maximum of 1 [24]. It is of course also desirable to be able to calculate the similarity between a search query and all documents. This can be done by considering the query to be a *pseudo-document*, i.e. a document where only the query terms occur. The pseudo-document is subsequently projected onto the less-dimensional representation of the TBD matrix, and compared to other documents using the cosine coefficient [12].

Dumais discusses the use of *term weighting*, which is intended to improve the results through the assignment of different weights to terms depending on, for example, how many times they occur in a document or across documents [34]. Terms that occur many times in a document are likely more important than terms that occur few times. Thus, terms can be assigned *local weights* proportional to their respective frequencies within a document. Similarly, terms that occur only in a few documents are likely more important than common terms that occur in many documents. Thus, terms can be assigned *global weights* that are inversely proportional to their respective frequencies across documents.

Dumais mentions *log weighting* as a type of local weighting. With log weighting, the logarithm of 1 plus the term frequency is used instead of only the term frequency. The benefit of this is that large differences in term frequencies are dampened [34].

Finally, smaller documents can be given more significance and larger documents less through *normalisation* [62]. This is to prevent large documents from dominating over small documents. The total weight for a term is obtained by multiplying its local weight, its global weight and its normalisation factor.

5.3.4 Method Introduction

In software development, written artefacts, whether they be requirements, test cases or architecture component descriptions, need to use the same language, or they will not be understood by all stakeholders. Here, using the same language means essentially using the same or similar words for describing relevant concepts. Our basic hypothesis is that it is possible to identify the traceability that exists due to lexicographic relationships between different kinds of written artefacts, in this research requirements and architecture component descriptions.

The impact of new requirements can be divided into *direct* and *indirect* impact (the same argument holds for changed requirements). Direct impact concerns, for example, parts of the system that must be changed for it to include the new functionality. Indirect impact concerns parts of the system that must be changed as a consequence of the directly impacted parts being changed. From this division of impact, it follows that it is not enough to look at relationships between requirements and component descriptions, but it is also necessary to consider relationships between different component descriptions. This motivates our choice of LSI for extracting implicit traceability.

Our method uses LSI as its core component, but the TBD matrix is not constructed exactly as described in the previous section. First, instead of extracting all content words from the component descriptions (the documents), keywords in the requirements are manually identified and used as terms in the matrix. This is done for the following reasons:

- Our main goal with the method is to be able to estimate the impact of new requirements (see Section 5.1 for possible usage scenarios). Thus, the words that convey the lexicographic relationships must necessarily exist in the requirements.
- We argue that the common language used for all written artefacts must be rooted in the requirements, as the requirements dictate the intended function and use of the system.
- It allows us to disregard the language-dependent lexicographic processing required to sort out words without semantic meaning from the documents. This would be troublesome as words

in technical language are highly context-dependent (recall the earlier example of the word NUMBER).

Second, instead of using the entire term weighting approach described in the previous section, we only use local weighting (based on simple non-logarithmic term frequency) combined with weighting based on *keyword class*. During the phase of identifying keywords in requirements, each identified keyword is assigned to a predefined class. Examples of classes are general keywords, context keywords and application keywords. When weighting terms in the TBD matrix, more significance can be given to terms belonging to a certain class. Our intention is to put more emphasis on application keywords (such as domain and system keywords), and less emphasis on more generic keywords.

We do not use global weighting and normalisation for the following reasons:

- Global weighting is performed to give more significance to words that are infrequent. The rationale behind this is that infrequent words, being unusual, are considered to be more meaningful than frequent, common words. In our method, the meaningfulness of a word follows from its class belonging. Furthermore, the frequency of a word across all documents reflects in our case the distribution of the corresponding functionality in the architecture, which is important when performing impact estimation.
- Normalisation is performed to prevent large documents from overwhelming smaller documents. In our case, large documents can be said to represent large architecture components, which are likely to be more affected by new requirements than smaller components. We do not want to discard this type of information.

5.3.5 Method Steps

Our proposed method is incremental, and should be run for each new requirement or each accumulated batch of requirements. Exactly how often to run the method is basically a cost-effectiveness trade-off. The following list gives an overview of the four main steps of the method:

1. **Screen for relevance**: In this step, which is optional, the requirements are checked for relevance with respect to the

traceability between requirements and architecture components. If a requirement is deemed irrelevant, it is excluded from further use of the method.

2. **Identify keywords**: This is a manual or semi-automatic step, in which keywords are identified in the requirements. The keywords are classified and stored in persistent glossaries.
3. **Identify dependencies using LSI**: In this automatic step, LSI is used to identify dependencies between the requirements and the architecture components. The strength of a dependency indicates the estimated impact of a particular requirement on a particular component.
4. **Examine results and estimate impact**: This is a necessarily manual step that serves two purposes. First, the results from the previous step are examined for accuracy. Second, the total requirements impact is estimated based on individual component impact estimates.

Step 1: Screen for Relevance. The purpose of this step is to sort out requirements that do not bear directly on the dependencies intended to be identified. More specifically, requirements that are unlikely to be strongly connected to the architecture should be disregarded. One example is non-functional requirements, which typically have very broad impact and consequently may have keywords that occur in all or none of the architecture component descriptions. Another example is requirements that concern installation and support, as these likely are disconnected from the functional scope of the architecture.

Step 2: Identify Keywords. During keyword identification, the relevant keywords in a requirement are identified manually. The resulting keywords are stored in glossaries corresponding to predefined keyword classes, if such classes have been defined, or in a global glossary that do not distinguish between keyword types. The choice of classes depends on the situation; as stated before, one example is to have a general class, a context class and an application class. Thus, keywords, especially those in more generic classes, can be reused for future releases and systems. If appropriate glossaries already exist (for example, as a result of applying the method to previous systems), they should be used as an input to the identification step. This will allow automatic keyword identification, as discussed next, which maximises the gain from previous efforts spent on identifying keywords.

While the identification of keywords is largely manual, it can be automated based on previous glossaries as well as on already identified keywords. Provided that there is a bounded space of keywords, more and more of it will be covered as the keyword identification progresses. In the end, new requirements should require little identification effort, as the majority of their relevant keywords should already have been identified (and thus can be automatically suggested as keywords). This does of course not hold if a new requirement introduces a feature based on an entirely new concept.

Note that for the automatic identification of keywords based on previously identified keywords to work, it is necessary to use fuzzy string matching as described in Section 5.3.2. Still, there will always be different words that are very similar, such as DATE and DATA. A solution is to construct an exception list based on rejected auto-suggestions. Such a list can be used to improve both the automatic suggestion of keywords and the accuracy of the impact estimation step. Another problem is words that are truly similar, but fall beyond the threshold anyway. These will most likely be recognised by the method user, who naturally will do “manual stemming” as part of his or her thought process. Such words can be stored in a synonym list.

Step 3: Identify Dependencies Using LSI. In this step, which is entirely automatic, the identified keywords are matched against the component descriptions to add to the TBD matrix (the matrix is never final, as new requirements may appear at any time; it may, however, be treated as a complete matrix after each new requirement or batch of requirements).

To build the TBD matrix, the identified keywords are put as terms along the vertical axis of the matrix. Then, each component description (i.e., each document) is examined for the occurrence of each term, and the appropriate frequencies are recorded in the column of the matrix that represents the description. In order to find terms in component descriptions, fuzzy string matching is used to decrease the sensitivity for misspellings and morphological variants. This is described in Section 5.3.2.

The TBD matrix is subsequently fed to the SVD algorithm, which performs dimensional reduction as described earlier. Finally, each requirement is seen as a query for relevant component descriptions. Thus, the requirement is projected onto the same space as the descriptions, and similarity measures are calculated for each description. The similarity measure can be seen as the strength of a requirement-component dependency, which in turn can be regarded as the estimated impact of the requirement on the component. Note that this individual estimate is only one fraction of the total impact of the requirement; it may be zero or close to zero if the component is not related to the requirement.

Both the previous and this step can be tailored with respect to the algorithm used for fuzzy string matching. However, this step is more sensitive to the choice of algorithm than the keyword identification step, as the entirely automatic procedure does not provide any means for manual confirmation or rejection of fuzzy matches. While the edit distance algorithm is straightforward, alternatives exist, for example Hamming distance and Levenshtein distance (both closely related to the edit distance) [107].

The distance threshold can (and should) be adjusted depending on the language in both requirements and component descriptions, and the types of misspellings and morphological variants that can be expected.

Step 4: Examine Results and Estimate Impact. In the final step, which is performed manually, the requirement-component dependencies are examined for accuracy. This is particularly important in early use of the method, before the relevant thresholds have been adjusted to give maximum confidence in the results. For example, if the estimated individual impact of a requirement on a components seems to be too large (as judged by system experts), the fuzzy string matching threshold could be made more restrictive.

The total estimated impact of a requirement is based on the individual estimates for each component. There are four ways in which the total impact can be estimated:

1. Select all components with an estimated impact that exceeds a predefined threshold.
2. Always select the *N* highest ranked components (i.e., those with highest impact estimates).
3. Use a combination of (1) and (2).
4. Visualise the component impacts, and select components manually based on differences in impact estimates. For example, if the two highest ranked components have similar estimated impact,

but there is a leap to the third ranked component, select only the first two.

The obvious drawback with the first three ways is that it is unlikely that there is an appropriate threshold, or rank limit (i.e., value of *N*), that is suitable for all requirements. A lower threshold or a higher rank limit (i.e., larger *N*) will result in a greater total estimated impact, but also a higher possibility that the estimate is too large.

If a requirement has no estimated impact, there are three probable causes. It is important to determine which cause is the real one, and to deal with the requirement accordingly:

- The method failed, because of improper parameters, wrong keywords or insufficient level of detail in requirements and/or component descriptions. If this happens for many requirements, an appropriate action would be to consider if the method should be used at all.
- The requirement does not represent functionality, for example because it is a compatibility requirement or a non-functional requirement. If this is the case, then it can be dismissed without further action. Feeding the method with this kind of requirements may lead to unnecessary effort spent on keyword identification, though. To prevent this, such requirements should be weeded out already in the first step.
- Functional responsibility is missing in the system. In other words, the architecture does not contain any component responsible for the functional area the requirement belongs to. Should this be the case, it is necessary to extend the architecture with additional components, or to expand the responsibility of existing components.

In the long run, the method may reveal components that seldom or never are included in the total impact estimates. The reasons for this are similar to those for requirements without estimated impact:

- The method failed; see above.
- The component is a component whose main function is to assist other components in order to achieve separation of concerns. An example is a database object framework that all other components use to retrieve information from the database.
- The system has more functionality than dictated by the requirements. This can happen if the component remains from an earlier release or version, even though the requirements for it have

been removed. Appropriate actions would be to consider removing the component, or to add requirements that dictate its functionality.

5.4 Evaluation

In order to evaluate the method, we have applied it to a release of an industrial software system developed for the telecom domain. This section describes the evaluation context, the tools used, and the execution of the method.

The evaluation is the first in a series of three planned evaluations, of gradually increasing thoroughness:

1. Laboratory evaluation using available traceability data from an existing system.
2. Interview-based evaluation involving the developers of an existing system. This evaluation should allow a comparison to undocumented, actual traceability in addition to documented traceability.
3. Evaluation in a sharp situation, for new requirements on a system being developed. The intention is to investigate how the method performs compared to a system expert performing the impact estimation task.

5.4.1 Evaluation Context

The system has around 50 architecture components and around 400 requirements. Since the release has already been developed, we use the method in a less incremental way than described in Section 5.3.5. More specifically, we collect all requirements in one batch and run the method steps sequentially.

The traceability we have used to evaluate against consists of dependencies between features and requirements, and features and architecture components. Each feature is connected to a number of requirements and a number of components. Thus, it is not possible to see dependencies for individual requirements. It should also be noted that the documented traceability exist in the form of solution proposals, which means that it corresponds to predicted impact rather than actual impact. Therefore, we cannot assert the exact performance of our method, but only if it seems to provide estimates that agree with the predictions documented by the developers of the system.

5.4.2 Tools Used

In order to test our method, we constructed a tool that supports the keyword identification step as described earlier, by collecting keywords in glossaries and automatically suggesting relevant keywords based on fuzzy string matching. The tool also builds the TBD matrix and evaluates the similarities between requirements and components.

For the SVD method, we used the SVDLIBC tool developed by Rohde [99]. This tool is based on SVDPACK by Berry [11], and uses the Lanczos algorithm for calculating the decomposition of the TBD matrix. It outputs all relevant matrices that are necessary for further calculation of query-document similarity.

5.4.3 Step 1: Screen for Relevance

Of the 400 initial requirements, around 80 were deemed irrelevant. Some of these requirements were non-functional, others concerned backwards compatibility, installation, hardware etc. The remaining 320 requirements were subsequently fed to the keyword identification step.

5.4.4 Step 2: Identify Keywords

Table 5.1 shows the keyword classes used in the evaluation (as suggested in the example earlier). For each class, the type of words in the class are stated, as well as how the keywords were distributed over the classes.

The class weights, also shown in Table 5.1, were set to reflect the differences in word significance among the classes. We recognise, however, that we need to explore the effect of class weights further in order to determine more well-founded weights. The classification of keywords was not a trivial process, indicating that it is crucial to select non-overlapping classes and to explicitly define the types of words that go into the different classes.

Furthermore, during initial testing of the tool, the following observations were made:

- A maximum edit distance of 2 seemed to be appropriate for long words, whereas it was unacceptable for short words. For example, the edit distance between NAME and BLAME is 2, even though the words are definitively not similar.
- Even with a distance threshold of 1 for short words, technical abbreviations (which, in general, tend to be short) often was regarded by the algorithm as similar to words like FOR and ALL. It should be noted that we chose to not remove stop words, in accordance with the discussion in Section 5.3.1.

Table 5.1 Keyword Classes Used in the Evaluation

Class	Word type	Share	Weight
General	general words (relevant from a requirements formulation point-of-view)	18%	1
Context	consisting of words related to events or functionality in the system	48%	2
Application	words specific for the domain or the system	34%	3

The solution we decided on was to use the following thresholds:

- 0 (i.e., exact match required) if either of the words was shorter than four letters, and
- 1 if either of the words was shorter than seven letters, and
- 2 otherwise.

The limits and exact thresholds were selected purely based on observation, and seemed suitable in our case. More sophisticated solutions are briefly mentioned in Section 5.6.

The keyword identification was complicated by the fact that it was not always clear whether or not a word should be regarded as a keyword. In software requirements, very generic words may have great significance; consider, for example, the words ABOVE and BELOW in relation to the word THRESHOLD. To not miss any keywords, we erred on the side of having too many keywords.

Some of the effort was spent on rejecting keywords automatically suggested because of polysemy or because of clearly different words falling within the fuzzy string matching thresholds. Furthermore, a synonym list was maintained based on “manual stemming” of keywords that were not automatically detected as similar.

5.4.5 Step 3: Identify Dependencies Using LSI

As SVD performs a dimensional reduction of the original TBD matrix, there is the question about how many dimensions to use in the reduction. In the original LSI paper by Deerwester et al., the authors argue that neither too few nor too many dimensions should be used, but rather the number that results in the best performance [30]. Landauer et al. classify the problem as being an empirical issue, and point out that the optimal dimensionality is the one that causes correct induction of underlying relationships in the documents [65]. Sarwar et al. resolve the issue by evaluating a number of different dimensionalities and picking the one with the best performance [103].

With no firm guideline for selecting the number of dimensions, we went for a 75% reduction as recommended in a data conversion script designed for SVDPACK. Thus, with an original dimensionality of 50 (given by the fact that there was 50 architecture components), the destination dimensionality was set to 12. There is clearly an incentive for studying the effect of dimensionality in future work.

As stated in Section 5.3.4, we have chosen to disregard global weighting and normalisation in the TBD matrix in favour of class weighting. Thus, in the evaluation, we assigned to each cell in the TBD matrix the product of the in-document frequency and the class weight of the term.

We did not remove keywords from the TBD matrix that occurred in only one document, as we regard such occurrence patterns as important in the context of impact estimation (as they communicate important information about the architectural scope of a functional concept).

5.4.6 Step 4: Examine Results and Estimate Impact

The purposes of this step are to examine the identified dependencies in order to assess the accuracy of the method, and to estimate a total requirements impact based on individual component impact estimates. In this special case of evaluating the method on an already implemented system, we verify the accuracy of the results by comparing them to the somewhat coarse-grained documented traceability of the system. For the sake of structuredness, the actual analysis of this is discussed further in Section 5.5.

Due to the exploratory character of the evaluation, we did neither consider a similarity threshold nor a fixed rank limit in advance. Deerwester et al. [30] seem to consider a cosine coefficient above 0.9 to represent nearness between documents, while Marcus and Maletic [76] set the limit at 0.7. We are not certain, however, exactly in which contexts these figures are appropriate or where the threshold is for a "good enough" coefficient. Furthermore, as pointed out before, having a static threshold or rank limit can be problematic.

As mentioned in Section 5.4.1, the documented traceability of the system connects clusters of requirements to clusters of architecture components. As a result, we could only perform a coarse comparison of the impact estimates produced by the method and the impact predictions documented by the developers. Furthermore, we were not able to access all relevant traceability documentation, which means that only around 200 of the 320 requirements could be subjected to the comparison. Finally, the documented traceability did not encompass all 50 components in the system; a majority of the components were not connected to any requirements. This naturally affects the evaluation outcome, as all 50 components were considered as impact candidates.

Figure 5.1 shows the average cosine coefficients for the 10 highest-ranked component descriptions. For example, the value of the cosine coefficient of the highest ranked component description for each requirement was on average around 0.65. The error bars in the figure each correspond to one standard deviation. It is clear that even if we had used the lower similarity threshold of 0.7 mentioned in the previous section, most requirements would only have been considered to have an estimated impact on at most one component.

5.5 Analysis

Two common performance indicators for information retrieval methods are *precision* and *recall* [6]. Assume that an information retrieval method (such as LSI) deems a set of documents A to be relevant for a query. Furthermore, let R be the set of all documents truly relevant for the query. It follows that $Ra = R \wedge A$ is the set of documents that are actually relevant among the ones deemed relevant by the method. Now, precision and recall can be defined as follows (|*X*| denotes the cardinality of set *X*):

$$\text{Recall} = \frac{|Ra|}{|R|} \tag{5-1}$$

$$\text{Precision} = \frac{|Ra|}{|A|} \tag{5-2}$$

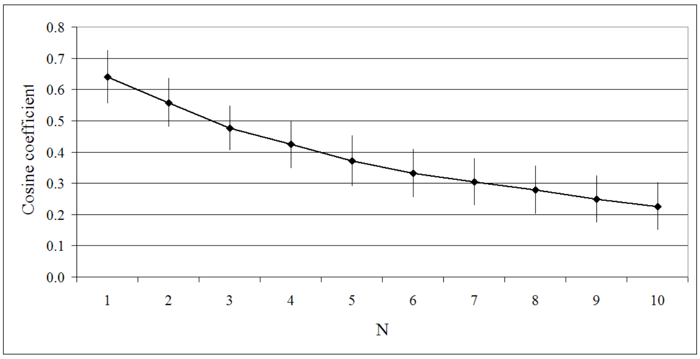


Figure 5.1 **Average Query-Document Similarities**

Table 5.2 shows average precision and recall figures for the 200 requirements used in the evaluation, based on rank limits (corresponding to |*A*|) from 1 to 10. The table does not show rank limits above 10, since no requirement was predicted by the developers to have an impact on more than 10 architecture components. Both |*R*| and |*Ra*| in the table are average values over all 200 requirements.

The recall reflects in our case how many of the components predicted to be impacted by the developers that the method was able to pinpoint. For example, if only the highest ranked component would be regarded as our method's estimated impact, then that component would only correspond to the "true" predicted impact for 2.6% of the requirements. The precision is a measure of how large the estimated impact needs to be to reach a certain recall level. For example, in order to reach 30% recall, we would have to accept the estimated impact to consist of slightly more than 80% false positives.

Normally, precision and recall figures are used to compare the performance of two information retrieval methods, often by interpolating the precision for a set of 11 standard recall values (0-100%)

[6]. The figures in Table 5.2 show that the method was not able to reach maximum recall (i.e., where all relevant documents have been captured) even for a rank limit of 10. Marcus and Maletic, who used LSI to extract traceability between source code and documentation, reported on recall values starting at 60% and reaching 100% at a rank limit of 11, and corresponding precision values from 77% down to 12% [76].

Table 5.2 Recall and Precision for the Evaluation

|A|	|R|	|Ra|	Recall	Precision
1	5.46	0.14	2.6%	14.4%
2	5.46	0.33	6.1%	16.6%
3	5.46	0.50	9.1%	16.5%
4	5.46	0.69	12.6%	17.2%
5	5.46	0.87	15.9%	17.3%
6	5.46	1.07	19.6%	17.8%
7	5.46	1.25	22.9%	17.8%
8	5.46	1.45	26.5%	18.1%
9	5.46	1.62	29.8%	18.0%
10	5.46	1.74	31.9%	17.4%

It is not entirely clear to us the amount of confidence that can be attributed to the recall and precision values in Table 5.2, given that the documented traceability, against which we compare results from the method evaluation, associates a cluster of components to each requirement rather than one single component. This means in practise that if a requirement is associated with five components, it may well be that in reality, only one of them corresponds to the true locus of the requirement. Thus, the figures in Table 5.2 must be seen as a worst-case scenario.

A best-case scenario would be to investigate in how many cases the method was able to pinpoint at least one of the components that was predicted by the developers to be impacted by a requirement. In the evaluation, the method was successful for around 125 of the 200 requirements. In other words, for slightly less than two thirds of the requirements, the method was able to identify at least one of the components predicted by the developers to be impacted. This is, however, still not an encouraging figure.

It should be noted that the method did differentiate between requirements with respect to component impact estimates. In other words, it did not estimate the same component impact for all requirements. This ability indicates some potential, despite the low performance in both the worst-case scenario and the best-case scenario.

5.6 Discussion and Future Work

In this section, we discuss the results of the method evaluation with focus on validity and possible explanations for the observed performance. We also share some subjective experiences from performing the evaluation, and present future work.

As seen in the previous section, the average recall and precision values were very low. Similarly, the performance in the best-case scenario was not that high. However, these figures may be incorrect, due to the fact that the documented traceability used in the evaluation may have been too coarse to act as a fair evaluation object. Furthermore, since the documented traceability represented predicted rather than actual impact, it may in itself be incorrect. To obtain a more robust evaluation, we intend to use a different evaluation system with more detailed documented traceability. Also, as described initially in Section 5.4, further evaluations are planned.

There are certain other issues related to the validity of the evaluation. First, the identification of keywords in requirements was performed by one of the authors, who has some, but not full, insight in the system used in the evaluation. The classification of keywords was performed by the same author. Thus, the evaluation results could be different than if a true system expert had performed the identification and classification (as intended). Second, the requirements and the component descriptions used in the evaluation did not have entirely corresponding dates. More specifically, the component descriptions were last modified 3-4 months after the requirements. This may mean (although it has not been verified) that the component descriptions had been updated to correspond better to the current requirements, which potentially could have affected the evaluation results.

It is not very likely that LSI does not work well in the evaluation context (i.e., with written software development artefacts), since written software development artefacts do not have drastically different characteristics than most textual documents, and LSI has previously proved to perform well both in its intended context [30] and in a software development context [76]. There are, however, a number of parameters that may have affected the evaluation:

- The component descriptions we used as documents for LSI contained from around 800 to around 21 000 words, with an average of 5 000 words. Marcus and Maletic mention, however, that LSI should be used on paragraphs or sections rather than entire chapters, which tend to be too large [76]. This implies that we should try to break down the component descriptions into smaller parts and use these parts as documents.
- As stated earlier, the number of dimensions to use in the SVD reduction of the TBD matrix is an open research issue. We chose 12 dimensions, which may be too few or too many. It is clear that we need to experiment with different values of this parameter in order to see how the results are affected.
- When weighting terms, we used only local (non-logarithmic) weighting and class weighting. Dumais describes other weighting schemes that we need to consider [34]. Also, the class weights used (1, 2 and 3) may be inadequate with respect to properly balancing the significances of the different keyword types. Which class weights to use is a question that must be addressed in future work.

While evaluating our method, we found that the described automation of the keyword identification worked very well, in that already identified keywords were automatically suggested for new requirements. As described in Section 5.4.4, however, the fuzzy string matching algorithm occasionally failed with respect to matching words correctly. We also realised that the process of searching through the component descriptions in order to locate and count keywords can be an extremely time-consuming process, even for a fast computer. The reason is that we used a very straightforward approach of comparing each word in a description with each keyword, which obviously is slow given that the edit distance algorithm is $O(mn)$ (where *m* and *n* are the lengths of the words to compare, respectively) [107]. There are plenty of room for improvement here, for example to use at least some stop words (see Section 5.3.1 for a motivation to why we did not use stop words), to cache word distances, to use early rejection of words that differ on the first letter (see below) and so on.

Finally, some other issues that we wish to explore further are:

- The use of fuzzy string matching and proper thresholds for finding similar words despite misspellings and morphological variants. A threshold based on percentage can be problematic for short words (as each individual letter will have greater influence than in a long word), while the approach we used (to have three distinct thresholds based on word lengths) was not optimal either (see Section 5.4.4). A possible solution is to assign different significances to letters depending on where they occur in a word. For example, we believe it to be less likely that a word is misspelled in the beginning than in the middle or in the end. While UBIQUITOUS is difficult to spell completely right, most people would probably get the U right. An exception is possibly ACQUIRE, which admittedly has a difficult beginning, at least for people who do not have English as their native language.
- Even though Berry et al. [12] mention that LSI should not need stemmed words to function properly, we want to investigate the use of stemming and certain stop words (in particular simple ones such as OR, AND and THE) to further automate and support the manual keyword identification step.
- Since LSI allows us to not only find the similarity between a requirement (expressed as a query) and component descriptions (the documents), but also between different component descriptions, it may be possible to use the outcome of LSI as an indicator of problems with the distribution of functional responsibility in the architecture. For example, if many component descriptions are very similar, it could mean that the corresponding components should be joined together.

5.7 Summary

We have presented a feasibility study of a semi-automatic method for estimating the impact of new requirements on an existing system, based on lexicographic relationships between requirements and architecture component descriptions. The method can also be used for requirements changes and to extract the traceability of a legacy system.

The method consists of four steps. In the first step, the requirements are screened for relevance (e.g., non-functional requirements may be sorted out). Relevant keywords are subjected to the second step, in which keywords are manually identified and classified (although automation to a large extent is possible). In the third step, Latent Semantic Indexing is used to automatically create dependencies between requirements and architecture component descriptions. Finally, in the fourth step, the dependencies are examined, and requirements impact is estimated manually based on them (visualisation and some automation can alleviate the manual effort).

In an evaluation on an industrial software system, the performance of the method was less than expected, indicating that it would not be suitable for impact analysis. However, the documented traceability against which the method estimates were compared was coarse-grained and represented predicted impact rather than actual impact of requirements (and may thus have been incorrect in itself). Despite the discouraging results, we will perform additional, more robust, evaluations in order to better assess the performance of the method. Moreover, there are a number of parameters of the method that can be adjusted and must be explored further.

Finally, we would like to point out that we believe our hypothesis, that it is possible to find lexicographic relationships between requirements and architecture components due to the use of similar language, to be sound. Thus, we find it most relevant to pursue further exploration and evaluation of our proposed method. Also, we would consider an appropriate design guideline to be to use the same language in architecture component descriptions as is used in requirements. This would facilitate and strengthen understanding of the system among its various stakeholders.

CHAPTER

6

Benchmarking k-Nearest Neighbour Imputation With Homogeneous Likert Data

Journal of Empirical Software Engineering, 2005 (submitted)

Per Jönsson and Claes Wohlin

Missing data pose a serious problem to researchers in many different fields of research, for example artificial intelligence [47], machine learning [9] and psychology [31]. The situation is, unsurprisingly, similar in software engineering [20, 81, 113]. The absence of data may substantially affect data analysis as statistical tests will lose power and results may be biased because of underlying differences between cases with and without missing data [55]. Simple ways to deal with missing data are, for example, listwise deletion, in which incomplete cases are simply discarded from the data set, or variable deletion, in which variables with missing data are discarded. However, a consequence of using a deletion procedure is that potentially valuable data are discarded, which is even worse than having missing data in the first place. Another approach, advantageous because it does not require useful data to be removed, is to use a method for *imputing* data. Imputation methods work by substituting replacement values for the missing data, hence increasing the amount of usable data.

A multitude of imputation methods exist (see, for example, [54] for a categorisation), whereas this chapter deals mainly with hot-deck k-Nearest Neighbour imputation, but also with Random Draw Substitution, Random Imputation, Median Imputation and Mode Imputation. In hot-deck imputation, a missing value is replaced by a value derived from one or more complete cases (the donors) in the same data set. The choice of donors should depend on the case being imputed, which means that Median Imputation, for example, in which a missing value is replaced with the median of the non-missing values, does not qualify as a hot-deck method [102]. There are different ways of picking a replacement value, for example by choosing a value from one of the donors by random [55] or by calculating the mean of the values of the donors [9, 20].

The k-Nearest Neighbour (k-NN) method is a common hot-deck method, in which *k* donors are selected from the available neighbours (i.e., the complete cases) such that they minimise some similarity metric [102]. The method is further described in Section 6.3.5. An advantage over many other methods, including Median and Mode Imputation, is that the replacement values are influenced only by the most similar cases rather than by all cases. Several studies have found that the k-NN method performs well or better than other methods, both in software engineering contexts [20, 112, 113] and in non-software engineering contexts [9, 22, 116].

This chapter builds on previous work, where we have evaluated the k-NN method and concluded that the performance of the method was satisfactory. In order to better assess the relative performance of the method, we extend the evaluation by benchmarking the k-NN method against four other methods: Random Draw Substitution, Random Imputation, Median Imputation and Mode Imputation. These methods are clearly less sophisticated than k-NN, but can be said to form an imputation baseline. Thus, the main research question concerns the performance of the k-NN method in relation to the other methods.

The data used in the evaluation is of *Likert* type in a software engineering context. A Likert scale is ordinal, and consists of a number of alternatives, typically weighed from one and up, that concern level of agreement (e.g., disagree, agree, strongly agree etc.). Such scales are commonly used when collecting subjective opinions of individuals in surveys [98]. The evaluation is performed by running the k-NN method and the other imputation methods on data sets with simulated non-response.

Apart from the benchmarking, we discuss the following questions related to the k-NN method:

- How many donors should preferably be selected?
- At which amount of missing data is it no longer relevant to use the method?
- Is it possible to decrease the sensitivity to the amount of missing data by allowing imputation from certain incomplete cases as well?
- What effect has the number of attributes (variables) on the results?

The remainder of the chapter is structured as follows. In Section 6.1 and Section 6.2, we outline related work, describe the data used in the evaluation and discuss different mechanisms for missing data. In Section 6.3, we present the k-NN method as well as the other imputation methods against which we benchmark k-NN. In Section 6.4, we describe the process we have used for evaluating the k-NN method. Although the process is generic in the sense that it supports any imputation method, we focus on k-NN. In Section 6.5, we briefly describe how we instantiated the process in a simulation, but also how we performed additional simulations with other imputation methods. In Section 6.6, we present the results and relate them to our research questions. In Section 6.7, we discuss validity threats and outline possible future work. Finally, we draw conclusions in Section 6.8.

6.1 Related Work

As Cartwright et al. point out, publications about imputation in empirical software engineering are few [20]. To our knowledge, those that exist have focused on comparing the performance of different imputation methods. For example, Myrtveit et al. compare four methods for dealing with missing data: listwise deletion, mean imputation, full information maximum likelihood and similar response pattern imputation (which is related to k-NN with $k = 1$) [81]. They conclude, among other things, that similar response pattern imputation should only be used if the need for more data is urgent. Strike et al. describe a simulation of listwise deletion, mean imputation and hot-deck imputation (in fact, k-NN with $k = 1$), and conclude that hot-deck imputation has the best performance in terms of bias and precision [113]. Furthermore, they recommend the use of Euclidean distance as a similarity measure. In these two studies, the context is software cost estimation. Cartwright et al. themselves compare sample mean imputation and k-NN, and reach the conclusion that k-NN may be useful in software engineering research [20]. Song et al. evaluate the difference between MCAR and MAR using k-NN and class mean imputation [112]. Their findings indicate that the type of missingness does not have a significant effect on either of the imputation methods, and furthermore that class mean imputation performs slightly better than k-NN. In these two studies, the context is software project effort prediction.

It is common to compare imputation methods in other research areas as well. Batista and Monard compare k-NN with the machine learning algorithms C4.5 and C2, and conclude that k-NN outperforms the other two, and that it is suitable also when the amount of missing data is large [9]. Engels and Diehr compare 14 imputation methods, among them one hot-deck method (however, not k-NN), on longitudinal health care data [38]. They report, however, that the hot-deck method did not perform as well as other methods. Huisman presents a comparison of imputation methods, including Random Draw Substitution and k-NN with $k = 1$ [55]. He concludes that Random Draw Substitution is among the worst performers, that the k-NN method performs well when the number of response options is large, but that corrected item mean imputation generally is the best imputation method. In the context of DNA research, Troyanskaya et al. report on a comparison of three imputation methods: one based on single value decomposition, one k-NN variant and row average [116]. They conclude that the k-NN method is far better than the other methods, and also that it is robust with respect to amount of missing data and type of data. Moreover, they recommend the use of Euclidean distance as a similarity measure. Gmel compares four different imputation methods, including single-value imputation based on median and k-NN with $k = 1$ [49]. He argues that single-value imputation methods are considered poor in general as they disturb the data distribution by repeatedly imputing the same value. He concludes that the k-NN method seems to perform better than the other methods. Chen and Åstebro evaluate six methods for dealing with missing data, including Random Draw Substitution and Mode Imputation, by looking at the sample statistics mean and variance [23]. They report that Random Draw Substitution systematically biases both the mean and the variance, whereas Mode Imputation only systematically biases the variance.

Imputation in surveys is common, due to the fact that surveys often are faced with the problem of missing data. De Leeuw describes the problem of missing data in surveys and gives suggestions for how to deal with it [69]. Downey and King evaluate two methods for imputing data of Likert type, which is often used in surveys [31]. Their results show that both methods, item mean and person mean substitution, perform well if the amount of missing data is less than 20%. Raaijmakers presents an imputation method, relative mean substitution, for imputing Likert data in large-scale surveys [93]. In comparing the method to others, he concludes that it seems to be beneficial in this setting. He also suggests that it is of greater importance to study the effect of imputation on different types of data and research strategies than to study the effectiveness of different statistics. Nevertheless, Chen and Shao evaluate k-NN imputation with $k = 1$ for survey data, and show that the method has good performance with respect to bias and variance of the mean of estimated values [22].

Gediga and Düntsch present an imputation method based on non-numeric rule data analysis [47]. Their method does not make assumptions about the distribution of data, and works with consistency between cases rather than distance. Two cases are said to be consistent when their non-missing values are the same whenever they occur in both cases, i.e., donorship is allowed both for complete and incomplete cases. This resembles our relaxation of the k-NN method rules when it comes to selecting neighbours (see Section 6.3.5), in that both approaches allow values that will not contribute to the similarity measure to be missing in the donor cases.

6.2 Research Data

In this section, we present the data used in the evaluation. We also discuss different missingness mechanisms (i.e., different ways in which data can be missing).

6.2.1 Evaluation Data

The data used in the evaluation comes from the case study on architecture documentation at Ericsson, described in detail in Chapter 3. In the case study, a questionnaire containing questions pertaining to viewpoints on architecture documentation was distributed to employees in the organisation. For the evaluation, we chose to use the answers to six questions selected such that the resulting data set was complete and contained as many respondents as possible. The six questions were answered by 54 respondents.

Each of the six questions used a Likert scale for collecting answers, where the numbers 1 to 5 were used to represent different levels of agreement to some statement or query. Each of the numbers 1 to 5 was associated with a short text explaining its meaning, and we tried to make sure that distances between two adjacent numbers were conceptually similar everywhere.

We have previously examined the original data with respect to differences between roles, and found that there are no differences for the questions involved in this evaluation (see Chapter 3). We have also sought differences based on other ways to group the data, but found none. Hence, we presuppose that the data is homogeneous. Figure 6.1 shows the distribution of response options of the six questions as well as on average (rightmost bar). As can be seen, options 1 and 5 are largely underrepresented, while in particular options 3 and 4 are common answers to most of the questions.

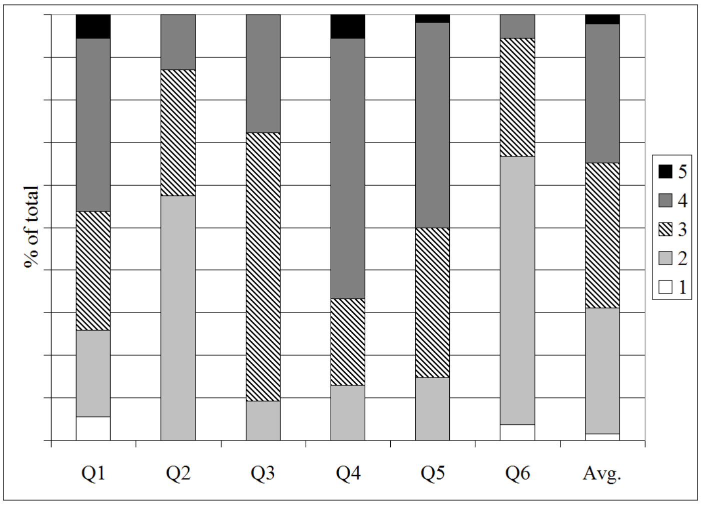


Figure 6.1 **Distribution of Response Options**

6.2.2 Missing Data

There are three main ways in which data can be missing from a data set [9, 20, 104]. These ways, or missingness mechanisms, are:

- *MCAR* (Missing Completely At Random), means that the missing data are independent on any variable observed in the data set.
- *MAR* (Missing At Random), means that the missing data may depend on variables observed in the data set, but not on the missing values themselves.
- *NMAR* (Not Missing At Random, or NI, Non-Ignorable), means that the missing data depend on the missing values themselves, and not on any other observed variable.

Any action for dealing with missing data must take the missingness mechanism into account. For example, to discard cases with missing data altogether is dangerous unless the missingness mechanism is MCAR [104]. Otherwise, there is a risk that the remaining data are severely biased. NMAR is the hardest missingness mechanism to deal with, because it, obviously, is difficult to construct an imputation model based on unobserved data.

When data are missing from the responses to a questionnaire, it is more likely that the missingness mechanism is MAR than MCAR [93]. For example, a respondent could leave out an answer because of lack of interest, time, knowledge or because he or she did not consider a question relevant. If it is possible to distinguish between these different sources of missing data, an answer left out because of lack of question relevance could be regarded as useful information rather than a missing data point. If so, the degree of missingness would be different than if the source of missing data could not be distinguished.

6.3 Imputation Methods

In this section, we describe the k-NN imputation method as well as the imputation methods used for benchmarking. We divide the imputation methods into the three categories *uninformed*, *informed* and *intelligent*:

- Uninformed imputation methods do not take into consideration properties of the data that are important from an imputation perspective, such as distribution of response options.

Random Draw Substitution, where a replacement value is randomly drawn from the set of response options, falls into this category.

- Informed imputation methods do take data properties into consideration. Random Imputation, where a replacement value is randomly drawn from the available (observed) answers, Median Imputation and Mode Imputation fall into this category.
- Intelligent imputation methods are those that base the imputation on hypothesised relationships in the data. The k-NN method falls into this category.

We go deeper into details for the k-NN method than for the other methods, in particular with respect to how the properties of the method affect the imputation results. Based on this, we differentiate between two different strategies for selecting neighbours. The standard strategy adheres to the rules of the method in that only complete cases qualify as neighbours, while the other relaxes this restriction slightly.

6.3.1 Random Draw Substitution

Random Draw Substitution (RDS) is an imputation method in which a missing value is replaced by a value randomly drawn from the set of available response options [55]. In our case, this means that we randomly generate replacement values from 1 to 5 such that all values have equal possibilities of being generated.

RDS falls into the category of uninformed imputation methods, as it does not consider data distribution or any other relevant properties. The relevance in benchmarking against RDS, or any other uninformed method for that matter, can of course be debated. However, we argue that a hallmark of any method necessarily must be to beat the entirely random case.

6.3.2 Random Imputation

Hu et al. describe generic Random Imputation (RI) as a method where replacement values are drawn at random from observed data, given some sampling scheme [54]. In our use of the method, we replace a missing value for a particular question with a value drawn randomly from all available answers to the question. Thus, we effectively set the probabilities of the response options in accordance with the distribution of response options for the question. This means that RI can be categorised as an informed imputation method.

By obeying the distribution of observed response options, we can expect RI to outperform RDS unless the possible response options are equally distributed for each question. This is not the case in our data, where response options 1 and 5 in particular are largely under-represented (see Figure 6.1).

6.3.3 Median Imputation

Due to the fact that our original data are ordinal, we impute based on median rather than mean. In Median Imputation (MEI), a missing value is replaced by the median of all available answers to the question. As with any type of single-value imputation, this method disturbs the distribution of response options, since the same value is used to replace each missing value for a particular question [49].

If the number of available answers to a question is even, the median may become a non-integer value. Since non-integer values are not compatible with ordinal data, we round the value either downwards or upwards at random in order to get an integer value.

MEI is highly sensitive to the distribution of response options for a question. More specifically, if the median corresponds to a response option with low frequency, the percentage of correct imputations will be low. Conversely, if the median corresponds to a frequent response option, MEI will have good performance.

6.3.4 Mode Imputation

Mode Imputation (MOI) is similar to MEI, except the mode is used instead of the median. As with MEI, MOI disturbs the distribution by imputing the same value for all missing values for a particular question.

A problem with using the mode as replacement value is that the distribution of available answers may be multimodal, i.e., have several modes. If that is the case, we obtain a unique replacement value by randomly selecting one of the modes.

If the mode corresponds to a response option with high frequency compared to the other response options, the percentage of correct imputations will be high. Otherwise, i.e., if the difference in frequency to the next most common value is small, the imputation performance decreases. Similarly, if the mode is a response option towards one of the ends of the scale, and a response option in the other end is common as well, the relative error of incorrectly imputed values will be high.

6.3.5 k-Nearest Neighbour

In the k-NN method, missing values in a case are imputed using values calculated from the *k* nearest neighbours, hence the name. The nearest, most similar, neighbours are found by minimising a distance function, usually the Euclidean distance, defined as (see, for example, [121]):

$$E(a, b) = \sqrt{\sum_{i \in D} (x_{ai} \angle x_{bi})^2} \qquad (6\text{-}1)$$

where

- $E(a, b)$ is the distance between the two cases *a* and *b*,
- x_{ai} and x_{bi} are the values of attribute *i* in cases *a* and *b*, respectively, and
- *D* is the set of attributes with non-missing values in both cases.

The use of Euclidean distance as similarity measure is recommended by Strike et al. [113] and Troyanskaya et al. [116]. The k-NN method does not suffer from the problem with reduced variance to the same extent as single-value imputation, because when mean imputation imputes the same value (the mean) for all cases, k-NN imputes different values depending on the case being imputed.

Consider the data set shown in Table 6.1; when calculating the distance between the cases Bridget and Eric, the attributes for which both have values are Q1, Q3, Q4 and Q5. Thus, $D = \{Q1, Q3, Q4, Q5\}$. We see that Bridget's answer to Q2 does not contribute to the calculation of the distance, because it is not in *D*. This implies that whether a neighbour has values for attributes outside *D* or not does not affect its similarity to the case being imputed. For example, Bridget and Eric are equally similar to Susan, because $E(\text{Bridget, Susan}) = E(\text{Eric, Susan}) = \sqrt{2 \times (4 \angle 2)^2}$ despite the fact that Bridget is more complete than Eric.

Another consequence of how the Euclidean distance is calculated, is that it is easier to find near neighbours when D is small. This occurs because the number of terms under the radical sign has fairly large impact on the distance. Again, consider the data set in Table 6.1; based on the Euclidean distance, Bridget and Eric are equally similar to Quentin (in fact, their distances are zero). Still, they differ considerably on Q5, and Eric has not answered Q2 at all. This suggests that the distance function does not necessarily reflect the *true* similarity between cases when D is small.

Table 6.1 Example Incomplete Data Set

Person	Q1	Q2	Q3	Q4	Q5
Bridget	2	3	4	2	1
Eric	2	–	4	2	5
Susan	–	–	2	4	–
Quentin	2	–	–	–	–

Once the k nearest neighbours (donors) have been found, a replacement value to substitute for the missing attribute value must be estimated. How the replacement value is calculated depends on the type of data; the mode can be used for discrete data and the mean for continuous data [9]. Because the mode may be tied (several values may have the same frequency), and because we use Likert data where the magnitude of a value matters, we will instead use the median for estimating a replacement value.

An important parameter for the k-NN method is the value of k. Duda and Hart suggest, albeit in the context of probability density estimation within pattern classification, the use of $k \approx \sqrt{N}$, where N in our case corresponds to the number of neighbours [33]. Cartwright et al., on the other hand, suggest a low k, typically 1 or 2, but point out that $k = 1$ is sensitive to outliers and consequently use $k = 2$ [20]. Several others use $k = 1$, for example Myrtveit et al. [81], Strike et al. [113], Huisman [55] and Chen and Shao [22]. Batista and Monard [9], on the other hand, report on $k = 10$ for large data sets, while Troyanskaya et al. [116] argue that the method is fairly insensitive to the choice of k. As k increases, the mean distance to the donors gets larger, which implies that the replacement values could be less precise. Eventually, as k approaches N, the method converges to ordinary mean imputation (median, in our case) where also the most distant cases contribute.

Neighbour Strategy. In hot-deck imputation, and consequently in k-NN imputation, only complete cases can be used for imputing missing values [9, 20, 102]. In other words, only complete cases qualify as neighbours. Based on the discussion in the previous section about how the Euclidean distance between cases is unaffected by values of attributes not in *D*, we suggest that it is possible to relax this restriction slightly. Thus, we see two distinct strategies for selecting neighbours.

The first strategy is in line with how the method normally is used, and allows only the complete cases to be neighbours. This means that no incomplete cases can contribute to the substitution of a replacement value in an incomplete case. We will refer to this strategy as the CC (*complete case*) strategy.

The second strategy allows all complete cases and certain incomplete cases to be neighbours. More specifically, a case can act as a neighbour if and only if it contains values for all attributes that the case being imputed has values for, and for the attribute being imputed. We will refer to this strategy as the IC (*incomplete case*) strategy.

It is important to note that we do not permit already imputed cases to be donors in any of the strategies. Thus, imputed data will never be used to impute new data.

For an example of the two strategies, consult again Table 6.1. Assuming we are about to impute attribute Q1 for Susan, the CC strategy would only allow Bridget to be a neighbour. The IC strategy, however, would allow both Bridget and Eric to be neighbours, because Eric contains values for at least the necessary attributes: Q1, Q3 and Q4. Because the IC strategy potentially has more neighbours to select donors from, it can be expected to be able to handle large amounts of missing data better than the CC strategy.

6.4 Evaluation Process

The process for evaluating the k-NN method consists of the three main steps data removal, imputation and evaluation (illustrated in Figure 6.2). In this section, we describe each step with respect to consumed input, responsibility, calculated metrics and produced output. The process is generic in that it does not depend on any particular data removal mechanism or imputation method. Here, we present it using k-NN as the imputation method. In Section 6.5, we detail the actual simulation of the process and describe how we have reused parts of it for benchmarking k-NN against the other imputation methods.

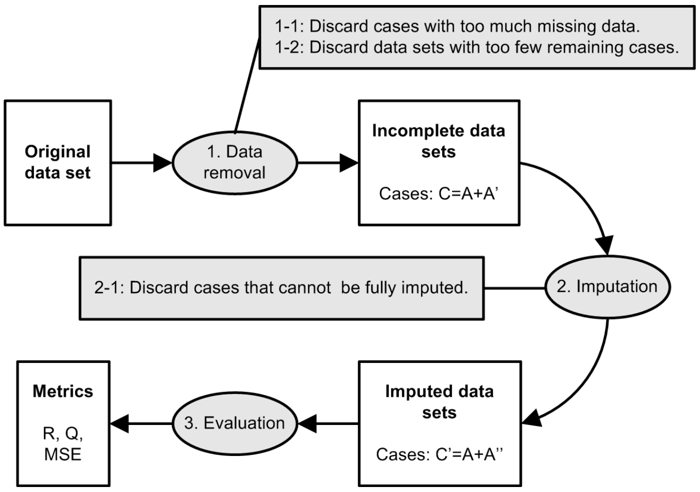

1-1: Discard cases with too much missing data.
1-2: Discard data sets with too few remaining cases.
Original data set
1. Data removal
Incomplete data sets
Cases: C=A+A’
2-1: Discard cases that cannot be fully imputed.
2. Imputation
Metrics
R, Q, MSE
3. Evaluation
Imputed data sets
Cases: C’=A+A”

Figure 6.2 Evaluation Process Outline

6.4.1 Data Removal – Step 1

Input to the data removal step is a data set where no data are missing. The responsibility of the step is to generate one or more artificially incomplete data sets from the complete data set, in order to simulate non-response. The generated data sets are subsequently sent to the imputation step.

In order to obtain a wide range of evaluation conditions for the k-NN method, it would be beneficial to use both the MCAR and MAR missingness mechanisms when generating incomplete data sets. In order to remove data to simulate MAR, a model for the non-responsiveness is required. In a longitudinal study of health data, for example, Engels and Diehr devised a model where the probability of removing a value increased if the previous value had been removed, thereby modelling a situation where a serious health condition could result in repeated non-response [38].

Possible models for simulating MAR in our data could involve, for example, experience in software architecture issues, organisational role or number of years in the industry, where different values would yield different probabilities for removing data from a case. However, given that our data is homogeneous, these models would not affect the imputation in other ways than would an MCAR-based model. Thus, we use only MCAR, and remove data in a completely random fashion.

We do not try to simulate different sources of missing data (e.g., lack of relevance, simple omission etc.), which means that we consider all removed data points as being truly missing.

There are two parameters that guide the data removal step, the *case reduction limit* and the *data set reduction limit*. These are called reduction limits because they prevent the data from being reduced to a level where it is unusable. The effects of the parameters can be seen in Figure 6.2. If it is decided in step 1-1 that a case contains too many missing values after data removal, as dictated by the case reduction limit, it is discarded from the data set. The reason for having this limit is to avoid single cases with so little data that it becomes meaningless to calculate the Euclidean distance to other cases. If it is decided in step 1-2 that too few cases remain in the data set, as dictated by the data set reduction limit, the entire data set is discarded. The idea with this limit is to avoid a data set with so few cases that it no longer can be said to represent the original data set.

These limits mean, in a way, that we combine the k-NN imputation method with simple listwise deletion. As discussed earlier, this is dangerous unless the missing data truly is MCAR. However, we argue that keeping cases with very little data left would also be dangerous, because the imputed data would contain loosely grounded estimates. In other words, it is a trade-off that has to be made.

The removal step is executed for a number of different percentages. Furthermore, it is repeated several times for each percentage. Thus, the output from the removal step is a large number of incomplete data sets to be fed to the imputation step. For each incomplete data set coming from the removal step, we define

- A as the number of complete cases remaining,
- A' as the number of incomplete cases remaining, and thus
- $C = A + A'$ as the total number of cases remaining.

Since entire cases may be discarded in the removal step, the actual percentage of missing data may be different from the intended percentage. For the incomplete data sets generated in the simulation, both the intended percentages and the actual percentages of missing data are presented. When analysing and discussing the results, it is the actual percentages that are used, though.

6.4.2 Imputation – Step 2

Input to the imputation step is the incomplete data sets generated in the data removal step. Here, each data set is fed to the imputation method in order to have its missing values imputed. We exemplify this step using the k-NN method.

With the k-NN method, several imputations using different *k*-values and different neighbour strategies are performed for each incomplete data set. As discussed earlier, a missing value is replaced by the median of the answers given by the *k* nearest neighbours, which means that the replacement value may become a non-integer value if *k* is even. However, since the data in the data set are of Likert type, non-integer values are not permitted. To avoid this problem, only odd *k*-values are used.

The *k* cases with least distances are chosen as donors, regardless of ties among the distances, i.e., two cases with equal distances are treated as two unique neighbours. This means that it is not always possible to pick *k* cases such that the remaining $K \angle k$ cases (where *K* is the total number of neighbours) have distances greater to that of the *k*th case. Should such a situation occur, it is treated as follows. If *l*, $0 \le l < k$ cases have been picked, and there are *m*, $(k \angle l) < m \le (K \angle l)$ cases with distance *d*, then the $k \angle l$ first cases of the *m*, in the order they appear in the original data set, are picked. This procedure is safe since the cases in the original data set are not ordered in a way that could affect the imputation.

If there are not enough neighbours available, cases may get lost in the imputation process. For the CC strategy, this will always happen when *k* is greater than the number of complete cases in the incomplete data set. The IC strategy has greater imputation ability, though, but will inevitably lose cases when *k* is large enough. This second situation where cases can be discarded is numbered 2-1 in Figure 6.2.

The output from the imputation step is a number of imputed data sets, possibly several for each incomplete data set generated in the data removal step (depending on the imputation method used and its parameters). For each imputed data set, we define

- A'', $0 \le A'' \le A'$ as the number of cases that were imputed (i.e., that were not lost in step 2-1), and consequently
- $C' = A + A''$ as the total number of cases, and also
- B as the number of imputed attribute values.

6.4.3 Evaluation - Step 3

In the evaluation step, each imputed data set from the imputation step is compared to the original data set in order to measure the performance of the imputation. Three separate metrics are used: one *ability* metric and two *quality* metrics. The two quality metrics differ both in what they measure and how they measure it. The first quality metric is a measure of how many of the imputed attribute values that were imputed correctly. In other words, it is a *precision* metric. The second quality metric is a measure of how much those that were not imputed correctly differ from their correct values, which makes it a *distance* (or error) metric.

We define the ability metric as

$$R = \frac{A''}{A'} \tag{6-2}$$

which equals 0 if all incomplete cases were lost during the imputation (in step 2-1), and 1 if all incomplete cases were imputed. To define the precision metric, let B' be the number of matching imputed attribute values. Then, the metric can be expressed as

$$Q = \begin{cases} \dfrac{B}{B'} & \text{if } B > 0 \\ \text{undefined} & \text{if } B = 0 \end{cases} \tag{6-3}$$

which equals 0 if all the imputed attribute values are incorrect, and 1 if all are correct. Finally, we calculate the mean square error of the incorrectly imputed attribute values as

$$MSE = \begin{cases} \dfrac{\sum_i (x_i \angle \hat{x}_i)^2}{B \angle B'} & \text{if } B > 0 \text{ and } B' < B \\ \text{undefined} & \text{if } B = 0 \text{ or } B' = B \end{cases} \tag{6-4}$$

where x_i is the correct value and $\hat{x}_i$ is the imputed value of the *i*th incorrectly imputed attribute value.

Since $B = 0$ when $R = 0$, it is apparent that both the precision metric and the mean square error are invalid when the ability metric is zero. Moreover, the mean square error becomes invalid when $Q = 1$. Consequently, the three metrics need to have different priorities: *R* is the primary performance metric, *Q* is the secondary, and *MSE* is the tertiary. Recognising that it would be difficult to create one single metric for measuring the performance, no attempts to accomplish this have been made.

Average values of *R*, *Q* and *MSE* are presented in the results, because several imputations are performed with identical parameters (percentage, and for k-NN, value of *k* and neighbour strategy). For *R*, the mean includes all measured instances, while for *Q* and *MSE*, only those instances where the metrics are not undefined are included.

6.5 Simulation

The previous section described the outline of the evaluation process. In this section, we briefly address the actual simulation of the process and which parameters we used to control it. We also provide some information about the simulation software used. Finally, we explain how we reused the process to run additional simulations with different imputation methods in order to obtain benchmarking figures.

6.5.1 Parameters

Each of the three steps in the process described in Section 6.4 is guided by a number of parameters. As discussed, two reduction limits, the case reduction limit and the data set reduction limit, constrain the data removal step. Based on the number of attributes and cases in the original data set, we used the following values in the simulation:

- Case reduction limit = 3 (inclusive)
- Data set reduction limit = 27 (inclusive)

With six attributes in each case, the case reduction limit means that cases with less than 50% of the attribute values left were discarded in step 2-1. The reason for this limit is that we wanted each imputed case to have at least equally much real data as imputed data.

With 54 cases in the original data set, the data set reduction limit means that data sets with less than 50% of the cases left were discarded in step 2-2. Since each case is a respondent, we wanted to make sure that each data set being imputed contained at least half of the respondents in the original data set.

The removal step generated data sets where 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55 and 60 percent data had been removed (however, as discussed in Section 6.4.1, the actual percentages became different). For each percentage, 1 000 data sets were generated, which means that a total of 12 000 data sets were generated. The simulation was controlled so that the removal step would generate the requested number of data sets even if some data sets were discarded because of the data set reduction limit.

In the imputation step, the controlling parameters depend on the imputation method. For the k-NN method, the only controlling parameter is the choice of which *k*-values to use when imputing data sets. We decided to use odd values in an interval from 1 to *C*, inclusively. Even though we knew that the CC strategy would fail at $k = A + 1$, we expected the IC strategy to be able to handle larger *k*-values.

6.5.2 Software

In order to execute the simulation, an application for carrying out the data removal, imputation and evaluation steps was written. In addition, Microsoft Excel and Microsoft Access were used for analysing some of the results from the evaluation step.

In order to validate that the application worked correctly with respect to the k-NN method, a special data set was designed. The data set contained a low number of cases, in order to make it feasible to impute data manually, and was crafted so that the imputation should give different results both for different *k*-values, and for the two neighbour strategies. By comparing the outcome of the imputations performed by the application to the outcome of imputations made manually, it was decided that the implementation of the k-NN method was correct. To further assess this fact, a number of application features were inspected in more detail: the calculation of Euclidean distance, the calculation of median, and the selection of *k* donors for both strategies. Finally, a number of entries in the simulation results were randomly picked and checked for feasibility and correctness.

The implementation of the remaining imputation methods was deemed correct through code reviews.

6.5.3 Process Reuse

To be able to benchmark k-NN against the other imputation methods, we took advantage of the fact that we could reuse the results from the data removal step. We instructed the application to save the 12 000 incomplete data sets before passing them on to the imputation step. To obtain the benchmarking figures, we ran the simulation again for each of the other imputation methods except Random Draw Substitution, this time skipping the data removal step and feeding the saved incomplete data sets directly to the imputation step. This way, the other imputation methods worked with the same incomplete data sets as the k-NN method.

Moreover, we had constructed the application to accept a reference data set in the evaluation step, in case the data removal step was omitted. This allowed us to obtain values for the *R*, *Q* and *MSE* metrics.

6.6 Results

In this section, we present the results from the simulations of the k-NN method and the other imputation methods. First, we provide descriptive statistics of the incomplete data sets generated in the initial simulation (and reused in subsequent simulations). Then, we address the questions posed initially as follows:

- We compare the results for different values of *k* in order to find the appropriate number of donors. In doing so, we also look at differences between the CC and IC strategies, to assess whether or not the IC strategy is appropriate to use.
- We compare the results from the original simulation with results from simulations using data sets with 12 and 18 attributes, respectively.
- We look at how the performance of k-NN changes for different percentages of missing data, in order to find a limit where the method stops being usable.
- Finally, we compare the performance of k-NN with the performance of the other imputation methods described in Section 6.3, in order to be able to judge its relative goodness.

6.6.1 Incomplete Data Sets

As discussed in Section 6.4.1, there is a difference between the amount of data removed from the original data set and the amount of data actually missing from the resulting, incomplete, data sets. The main reason for this is that entire cases may be discarded because of the case reduction limit. Another, less significant, reason is rounding effects. For example, removing 5% of the data in the original data set means removing 16 attribute values out of 324, which equals 4.9%.

Table 6.2 shows descriptive statistics for the incomplete data sets generated in the removal step. Each row represents the 1 000 data sets generated for the percentage stated in the left-most column. The second and third columns contain the mean and standard deviation (expressed with the same magnitude as the mean) of the percentage of missing data, respectively. The fourth and fifth columns contain the average number of cases and the average number of complete cases in each data set, respectively. Finally, the sixth column contains the average number of imputations made on each data set. This corresponds roughly to the average number of cases ($\bar{C}$), which is the upper limit of *k*.

Table 6.2 Overview of Incomplete Data Sets

Pct.	Mean missing data (%)	s	$\bar{C}$	$\bar{A}$	Avg. #imp.
5	4.9	0.1	54.0	39.8	54.0
10	9.8	0.3	53.9	28.8	54.0
15	14.5	0.5	53.7	20.4	53.9
20	19.0	0.8	53.2	14.2	53.6
25	23.4	1.0	52.1	9.6	52.6
30	27.2	1.2	50.5	6.3	51.0
35	30.8	1.3	48.4	4.0	48.9
40	34.4	1.3	46.0	2.4	46.5
45	38.0	1.3	43.1	1.5	43.6
50	42.1	1.3	40.1	0.8	40.6
55	46.5	1.3	37.4	0.4	37.9
60	51.5	1.3	34.9	0.2	35.4

6.6.2 Comparison of k-Values and Strategies

For each percentage of missing data, we plotted the ability metric and the quality metrics for different values of *k* and for both of the neighbour selection strategies. It is not necessary to show all the 24 resulting diagrams, as there is a common pattern for all percentages. To illustrate this pattern, we show the diagrams for the data sets with 14.5% and 19.0% missing data, respectively, in Figure 6.3.

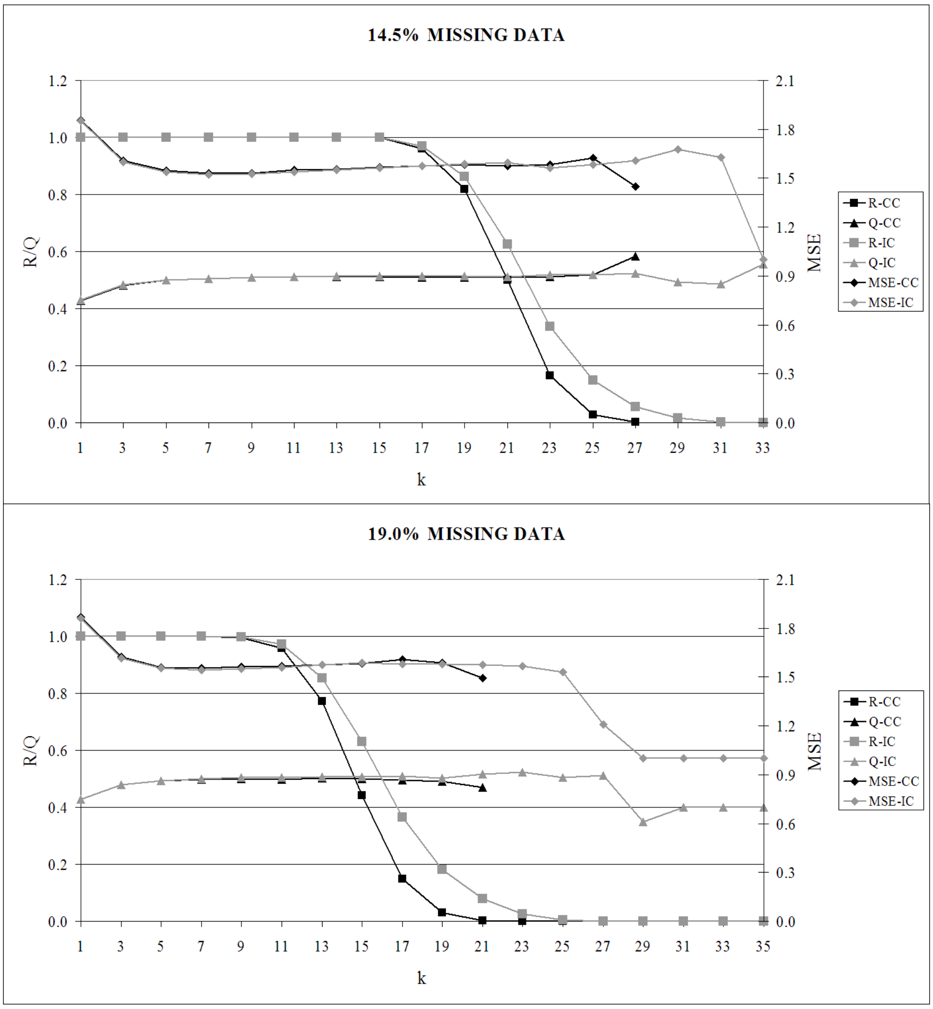


Figure 6.3 Performance at 14.5% and 19.0% Missing Data, CC and IC

The diagrams in the figure show the ability and quality for both the CC strategy and the IC strategy. In the upper diagram, the ability (*R*) is 1.0 up until *k* is around 15 for both strategies, after which it falls and reaches 0.5 when *k* is around 21 for the CC strategy and slightly more for the IC strategy. The latter limit coincides with the average number of complete cases ($\overline{A}$) in the data sets for this percentage (see Table 6.2). Similarly, in the lower diagram we see that the ability is 1.0 up until *k* is around 9, and falls to 0.5 when *k* is around 15. Such limits, albeit different, exist for other percentages as well.

Both diagrams further show that the precision (*Q*) of the method starts at around 0.4 when *k* is 1, and increases up to around 0.5 when *k* reaches 5. Thereafter, the precision is fairly unaffected by the value of *k* and varies only slightly on a "ledge" of *k*-values, an observation similar to that made by Troyanskaya et al. [116]. This is true for both strategies. Because of the priorities of the performance metrics, discussed in Section 6.4.3, the ledge has a natural upper limit as the ability of the method drops. The initial increase in precision and the ledge of *k*-values exist for other percentages as well, up to a percentage where the drop in ability occurs already for a low *k*. In our data, this happens when around 30% data is missing, in which case the ability drops to 0.8 for the CC strategy and 0.9 for the IC strategy already when *k* is 3.

The mean square error (*MSE*), which is the tertiary performance metric, starts off high but shows a noticeable decrease as *k* increases to 7. Then, it slowly increases for higher *k*-values on the aforementioned ledge. Although the increase is minimal, it seems to concur with the observation made in Section 6.3.5, that the estimated replacement values get worse as the mean distance to the donors increase. The described pattern in mean square error occurs for both strategies and for other percentages as well.

The differences between the neighbour strategies can be seen by comparing the black curves, representing the CC strategy, to the grey curves, representing the IC strategy. As can be seen, the curves for *R*, *Q* and *MSE* are nearly identical between the strategies. The main difference is that the ability (*R*) of the method, as expected, does not drop as fast for the IC strategy as it does for the CC strategy. Two important observations regarding the IC strategy are that the precision is generally not lower than for the CC strategy, and the mean square error is not larger.

We see, based on the discussion about the performance metrics above, that *k* should be selected so that it is large enough to be on the ledge, but low enough to minimise the mean square error. Since the ledge gradually diminishes for higher percentages of missing data, *k* would preferably depend on the amount of missing data. In fact, the dependency should be on the number of available neighbours for at least two reasons. First, the drop in ability occurs because the number of available neighbours decreases. For the CC strategy, the number of available neighbours is the number of complete cases. For the IC strategy, it is slightly more, but not so much more that the number of complete cases is an unfit approximation. Second, removing a certain percentage of data from two data sets with different numbers of attributes but the same number of cases would result in different numbers of complete cases.

Table 6.3 and Table 6.4 show the observed optimal *k*-values for the CC strategy and the IC strategy, respectively, given the average number of complete cases for the simulated percentages. It can be seen that the optimal value of *k* for a certain number of neighbours is the same regardless of strategy. The tables also show the values of *R*, *Q* and *MSE* for each optimal *k*-value. As can be seen, the quality metrics get gradually worse as the number of complete cases, and thus the ability of the method, decreases.

Table 6.3 Optimal k-Values with R, Q and MSE for the CC Strategy

	39.8	**28.8**	**20.4**	**14.2**	**9.6**	**6.3**	**4.0**	**2.4**	**1.5**	**0.8**	**0.4**	**0.2**
k	7	7	7	7	5	3	1	1	1	1	1	1
R	1.00	1.00	1.00	1.00	0.99	0.98	0.99	0.93	0.80	0.57	0.37	0.20
Q	0.52	0.51	0.51	0.50	0.48	0.47	0.42	0.42	0.41	0.41	0.40	0.40
MSE	1.56	1.54	1.53	1.55	1.58	1.63	1.88	1.89	1.94	1.93	1.95	1.95

Table 6.4 Optimal k-Values with R, Q and MSE for the IC Strategy

	39.8	**28.8**	**20.4**	**14.2**	**9.6**	**6.3**	**4.0**	**2.4**	**1.5**	**0.8**	**0.4**	**0.2**
k	7	7	7	7	5	3	1	1	1	1	1	1
R	1.00	1.00	1.00	1.00	1.00	0.99	0.99	0.98	0.92	0.82	0.69	0.56
Q	0.52	0.51	0.51	0.50	0.49	0.47	0.43	0.42	0.42	0.42	0.42	0.42
MSE	1.56	1.54	1.52	1.54	1.57	1.62	1.87	1.88	1.90	1.90	1.90	1.90

Looking for an appropriate model for *k*, we compared each optimal *k*-value to the square root of the average number of complete cases, as suggested by Duda and Hart [33]. The reason they suggest this model is that *k* should be large enough to give a reliable result, but small enough to keep the donors as close as possible. This concurs with our own requirements on *k*. Thus, we have chosen to examine

$k = \text{RoundOdd}(\sqrt{A})$, i.e. the square root of the average number of complete cases after data removal, rounded to the nearest odd integer. This function is compared to the optimal *k*-values in Table 6.5. As can be seen, the function underestimates *k* somewhat in the mid-range of missing data. This does not mean that the calculated *k*-values are inappropriate, though. The relative errors in *R*, *Q* and *MSE* between the non-matching calculated and optimal *k*-values are for the CC strategy within the ranges 0-0.80%, 0.63-4.03% and 0.44-4.19%, respectively, and for the IC strategy within the ranges 0-0.45%, 0.80-4.42% and 0.31-4.48%, respectively.

Table 6.5 Optimal k vs. Calculated k

	39.8	28.8	20.4	14.2	9.6	6.3	4.0	2.4	1.5	0.8	0.4	0.2
Optimal	7	7	7	7	5	3	1	1	1	1	1	1
Calculated	7	5	5	3	3	3	1	1	1	1	1	1

6.6.3 Comparison of Attribute Counts

As mentioned, the number of complete cases for a data set with a certain percentage of missing data depends on, among other things, the number of attributes in the data set. Thus, in order to further test our findings, we performed two additional simulations with the k-NN method. In the first, the number of attributes was increased to 12 by simply appending a copy of each case to itself. In the second simulation, the number of attributes was increased to 18 in a similar way. The case reduction limits were increased accordingly. Since the number of cases was unchanged in these extended data sets, a certain percentage of removed data yielded more incomplete cases compared to the data set with six attributes. Consequently, the ability of the method drops quicker with more attributes.

For 12 attributes and 4.9% missing data (5% removed), using $k = 3$ and the IC strategy results in $Q \approx 0.65$ and $MSE \approx 1.15$. The results are the same with 18 attributes, also with 4.9% missing data (5% removed), $k = 3$ and the IC strategy.

The diagrams in Figure 6.4 show the results of imputing data sets with on average 9.9% missing data using the IC strategy. With 12 attributes, the average number of complete cases at this percentage is 15.3, and with 18 attributes it is 8.0. The precision (*Q*) is highest at $k = 3$ in both diagrams, but declines as *k* increases instead of showing a ledge as was the case with six attributes. Another difference is that the precision generally is higher with more attributes. Also, the mean square error starts low in both diagrams, and the increase as *k* grows larger is articulated compared to the results with six attributes. These observations further support our requirements on *k*, as stated earlier.

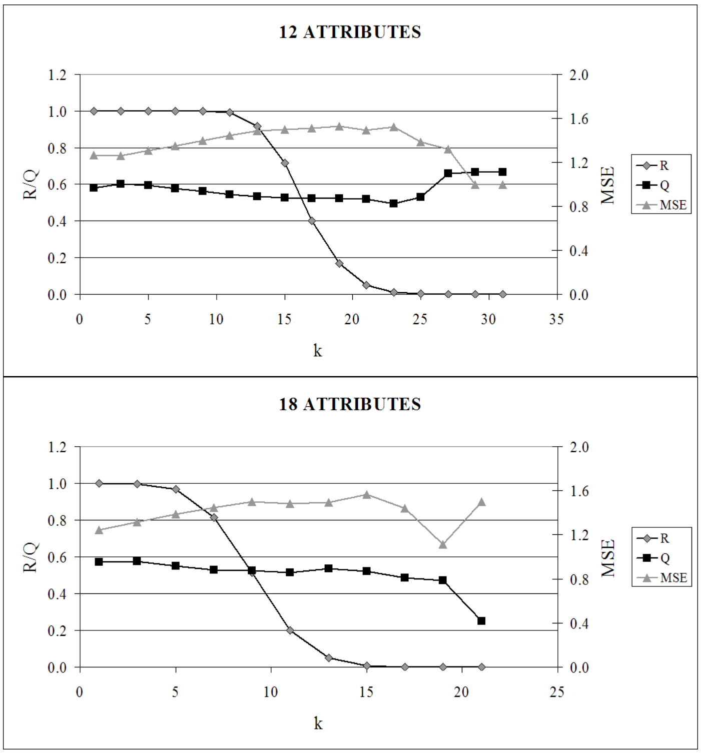


Figure 6.4 9.9% Missing Data, 12 and 18 Attributes, IC

In total, the results from the two additional simulations indicate that it is suitable to use $k = \text{RoundOdd}(\sqrt{A})$ with higher numbers of attributes as well, although comparing the optimal *k*-values and the calculated ones reveals that the optimal values are slightly lower for low percentages of missing data. As with six attributes, both *Q* and *MSE* get gradually worse as the percentage of missing data increases. For 12 attributes, the method can maintain maximum ability (at $k = 1$) up to 19.8% missing data (20% removed), whereas for 18 attributes, the corresponding limit is at 14.9% missing data (15% removed).

6.6.4 Comparison of Percentages

In addition to comparing the ability and quality for different *k*-values, we compared the ability of the method for different amounts of missing data, using for each percentage the optimal *k*-value found earlier. The diagram (for six attributes) can be seen in Figure 6.5 (for the raw numbers, see Table 6.3 and Table 6.4). Both neighbour strategies provide nearly maximum ability (*R*) up to around 30% missing data (when, on average, 88% of the cases are incomplete). After that, the ability when using the CC strategy drops rapidly down to 0.2 at around 50% missing data (when, on average, 98% of the cases are incomplete), meaning that only 20% of the incomplete cases were recovered. The IC strategy, on the other hand, drops less drastically and can recover nearly 60% of the incomplete cases at around 50% missing data.

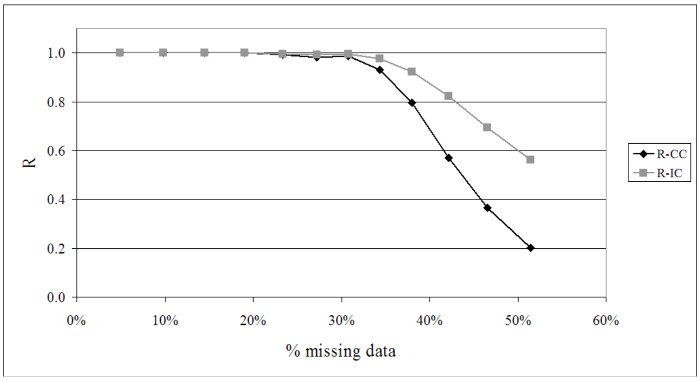


Figure 6.5 Ability vs. Amount of Missing Data

The figure clearly shows that the IC strategy is more advantageous when more data is missing. Because the comparison of *k*-values showed that the IC strategy does not give lower precision (*Q*) or larger mean square error (*MSE*) than the CC strategy, we consider it more favourable regardless of the amount of missing data.

6.6.5 Benchmarking

Here, we present the benchmarking of the k-NN method against the four other imputation methods. As basis for the comparisons, we use only the results from the original data set with six attributes imputed using the IC strategy.

For Random Draw Substitution, where the selection of replacement values does not depend on the data distribution, it is straightforward to calculate the expected values of R, Q and MSE. For the informed imputation methods, we have performed additional simulations reusing the incomplete data sets generated initially (see Section 6.5.3).

Random Draw Substitution. With RDS, we randomly draw a replacement value from the set of response options, i.e. from 1 to 5. Each response option has a 20% chance of being selected, which means that the expected value of Q is 0.2. Since this imputation technique never can fail to impute (as opposed to k-NN, which fails when there are too few neighbours), the expected value of R is 1.

The expected value of MSE can be calculated given the average distribution of response options in the original data set. Let $P(z)$ denote the probability that the correct value of a missing value is z. Given the fact that any value has a 20% chance of being imputed, the expected total MSE ($TMSE$) for all imputations can be expressed as

$$TMSE = 0.2 \times \sum_{x} \sum_{y} (P(y) \times |x \angle y|^2) \qquad (6\text{-}5)$$

where x is the imputed value and y is the correct value. The problem is that $TMSE$ includes the errors also when the correct value is imputed. These errors are all zero, which means that $TMSE$ is lower than the expected MSE, which is defined as the relative error for the incorrectly imputed values. To obtain the correct MSE, $TMSE$ must be divided by the probability of imputing an incorrect value:

$$MSE = \frac{TMSE}{0.8} \qquad (6\text{-}6)$$

With $P(z) = \{0.015, 0.296, 0.340, 0.327, 0.022\}$, $1 \le z \le 5$, which is the average distribution of response options in the original data, we obtain $MSE \approx 3.465$.

Random Imputation. With RI, we draw a replacement value from the set of available answers to the current question, which means that the distribution of response options is taken into consideration. Given that the missingness mechanism for our data is MCAR, the distributions in the incomplete data sets can be assumed to equal the distribution in the complete data set. Thus, RI can be expected to perform reasonably well. With MAR as the missingness mechanism, the performance could be worse.

As with RDS, this imputation technique cannot fail, and the expected value of R is 1. The simulation with RI as the imputation method resulted in $Q \approx 0.41$ and $MSE \approx 1.97$ averaged over all 12 000 incomplete data sets. The averages for each individual percentage did not deviate much from the total averages.

Median Imputation. With MEI, the replacement value is the median of all available answers to the current question. As pointed out in Section 6.3.3, the frequency of the response option that corresponds to the median has large effect on the imputation performance. Figure 6.1 shows that our data is favourable for MEI in this aspect, since most questions have frequent median response options.

MEI cannot fail to impute, which means that the expected value of R is 1. The simulation with MEI as the imputation method gave the values $Q \approx 0.50$ and $MSE \approx 1.59$ averaged over all incomplete data sets. The averages of Q for individual percentages did not differ much from the total average. However, for *MSE*, the averages ranged from 1.55 to 1.61.

Mode Imputation. With MOI, the replacement value is the mode of all available answers to the current question. If the distribution of answers is multimodal, one of the modes is selected randomly. As described in Section 6.3.4, MOI does not perform well if the response option that corresponds to the mode is only slightly more frequent than other response options. Figure 6.1 clearly shows that this is not the case in our data, which means that we can expect MOI to perform well.

MOI cannot fail to impute, which means that the expected value of R is 1. The simulation with MOI as the imputation method resulted in $Q \approx 0.54$ and $MSE \approx 1.85$ averaged over all incomplete data sets. The variations in average Q and average *MSE* for individual percentages were noticeable; Q varied from 0.51 to 0.56, and *MSE* varied from 1.79 to 1.90.

6.6.6 Summary and Interpretation of the Results

The results indicate that the k-NN method performs well on the type of data we have used, provided that a suitable value of k is selected. Table 6.6 presents an overview of the comparisons made in evaluating k-NN, whereas Table 6.7 shows an overview of the benchmarking against the other imputation methods.

Table 6.6 Results Overview

Attr.	CC	IC
6	• *R* starts to drop at around 30% missing data. • With maximum ability, *Q* is at best 0.52 and at worst 0.42. • With maximum ability, *MSE* is at best 1.53 and at worst 1.88.	• *R* drops less drastically than CC when the percentage of missing data increases. • *R* starts to drop between 30 and 35% missing data. • *Q* and *MSE* are similar to when using CC.
12	–	• *R* drops earlier (as there are fewer complete cases), at around 20% missing data. • *Q* is higher, at best 0.65. • *MSE* is lower, at best 1.15.
18	–	• *R* drops even earlier, at around 15% missing data, than with 12 attributes. • *Q* and *MSE* are similar to when using 12 attributes.

Table 6.7 Benchmarking Overview

Method	R	Q	MSE
k-NN (IC), 6 attr.	1 (up to 30-35% missing data)	0.42 to 0.52	1.53 to 1.88
k-NN (IC), 12/18 attr.	1 (up to 15-20% missing data)	up to 0.65	down to 1.15
RDS	1	0.2	3.465
RI	1	0.41	1.97
MEI	1	0.50	1.55 to 1.61
MOI	1	0.51 to 0.56	1.79 to 1.90

Table 6.6 shows that the IC strategy is favourable over the CC strategy, since it allows k-NN to maintain high ability for higher percentages of missing data, while the precision and mean square error are equally good. In addition, Figure 6.5 shows that, when using the IC

strategy, nearly 60% of the incomplete cases could be saved when 50% of the data were missing.

Furthermore, it can be seen that k-NN performs better with more attributes, both with respect to precision and mean square error. This is due to the fact that with more attributes, the method has more information available to discriminate between neighbours when it comes to distance.

It can be seen in Table 6.7 that RDS, as expected, does not perform very well. Comparing with the values of *Q* and *MSE* for k-NN, it is clear that k-NN easily outperforms the entirely random case. Furthermore, RI performs much better than RDS. The precision (*Q*) is twice as high, and the error (*MSE*) is much lower. However, compared to k-NN, RI falls short.

MEI touches upon the six-attribute k-NN in terms of both precision and mean square error, and given that the ability (*R*) is always 1, it seems to be a viable alternative. Disadvantages of MEI are that it is more sensitive than k-NN to the distribution of response options (see Section 6.3.3), and that it does not perform better when the number of attributes increases. Furthermore, as with all single-value imputation, MEI may result in a disturbed data distribution, which becomes particularly noticeable when much data are missing.

MOI does perform slightly better than the six-attribute k-NN with respect to precision (*Q*), but performs worse with respect to relative error (*MSE*). As with MEI, MOI is sensitive for the distribution of response option and does not improve with more attributes. The reason for MOI having worse *MSE* than MEI is that the modes of the majority of the questions in our data set correspond to response options 2 or 4, which do not dominate the distributions. Thus, if the mode is not the correct value, the error will be rather large.

With 12 or 18 attributes, k-NN outperforms both MEI and MOI. These methods does not scale with respect to number of attributes, since they only work with one attribute at a time.

Judging from the results, k-NN proved to have good performance. However, both Median Imputation and Mode Imputation could compete with k-NN, given that both these methods were favoured by the distribution of our data. Median Imputation had similar precision and similar relative error, whereas Mode Imputation had slightly better precision, but worse relative error. Both methods will always have maximum ability (i.e., save all incomplete cases) which makes them attractive when much data is missing and there are many incomplete cases.

It is of course desirable to achieve good values on all three performance metrics. However, when the performance decreases for whichever of the metrics, it is the priorities between them that should determine whether the imputation was successful or not. For example, if the quality drops but the ability stays high, the imputation may still be considered successful, because resorting to listwise deletion (or any other type of deletion procedure) may not be an option.

6.7 Validity and Future Work

In this section, we discuss threats to the validity of the evaluation and outline possible future work.

6.7.1 Threats to Validity

In the k-NN method, we used Euclidean distance as the similarity measure. However, the data was of Likert type, i.e., on an ordinal scale. This makes it debatable to perform distance calculations, which normally requires an interval scale. Still, we argue that the distance calculations were relevant, and thus the validity threat minimal, because effort was put into making the distances between Likert numbers similar. Furthermore, our results show that the k-NN imputations were successful after all.

In step 1 of the evaluation, we removed data from the original data set completely at random, which means that the missingness mechanism was MCAR. It is more likely, though, that missing responses to a questionnaire are MAR, as pointed out by Raaijmakers [93]. In other words, the missingness mechanism used in the evaluation did not fully represent a real-world situation. Due to the nature of our data, we could not avoid this problem.

It may be dangerous to use incomplete cases as donors when the missingness mechanism is MAR, for example if incomplete cases can be said to contain less valuable data. This could be the case if missing answers were an indication that the respondents did not take the questionnaire seriously. As a precaution, we recommend using a limit to prevent cases with far too much missing data both from being imputed and from acting as donors.

A threat to the generalisability of the results is that we used a fairly small data set with 54 cases as a basis for the simulation. With a small data set with missing data, the neighbours that can be used as donors are few. Hence, the outcome of the imputation is sensitive to disturbances, such as outliers, in the data. We do, however, believe that it is not uncommon to get a small data set when collecting data from a survey, which means that our simulation should be relevant from this point of view.

A threat to the evaluation of k-NN with 12 or 18 attributes is that we created these extended data sets by appending one or two copies of each case to itself. This means that the similarity, if any, between two cases is duplicated as well. In a real data set with 12 or 18 attributes, two cases could be similar for six of the attributes, but different for six other attributes. This may mean that the performance metrics for 12 and 18 attributes are overly positive.

6.7.2 Future Work

Due to the nature of our data, we used only MCAR as missingness mechanism (see Section 6.6.1). In future work, it would be interesting to study imputation of Likert data with systematic differences that allow MAR missingness. For example, Song et al. have concluded that the missingness mechanism does not significantly affect the k-NN method in the context of software project effort data [112].

Each of the questions in the original questionnaire used a Likert scale with five response options. However, it is also common to use Likert scales with more response options, for example seven or ten. Hypothetically speaking, k-NN should gain from the fact that more response options means that it would be easier to differentiate between neighbours (and find close donors), but should lose from the fact that more response options makes it easier for two respondents to answer similarly, yet differently. The other imputation methods would likely have worse performance, as more response options, provided they were used, would result in a wider distribution with smaller frequencies for individual response options. These hypotheses would be interesting to test in future work.

6.8 Conclusions

In this chapter, we have presented an evaluation of the performance of the k-Nearest Neighbour imputation method when using homogeneous Likert data. This type of ordinal data is common in surveys that collect subjective opinions from individuals. We performed the evaluation by simulating non-responsiveness in questionnaire data and subsequent imputation of the incomplete data.

Since we simulated the evaluation process, we were able to obtain great variation in the imputation parameters and operate on a large number of incomplete data sets. In the main imputation process, we used different values of *k*, and also two different strategies for selecting neighbours, the CC strategy and the IC strategy. The CC strategy, which concurs with the rules of the k-NN method, allows only complete cases to act as neighbours. The IC strategy allows as neighbours also incomplete cases where attribute values that would not contribute to the distance calculation are missing.

In order to measure the performance of the method, we defined one ability metric and two quality metrics. Based on the results of the simulation, we compared these metrics for different values of *k* and for different amounts of missing data. We also compared the ability of the method for different amounts of missing data using optimal values of *k*. Furthermore, we performed additional simulations with more attributes, in order to see how the number of attributes affected the performance of the method. Finally, we benchmarked the method against four other imputation methods, in order to be able to assess its relative goodness. The methods were Random Draw Substitution, Random Imputation, Median Imputation and Mode Imputation. The benchmarking was performed through additional simulations where the k-NN method was replaced by the other methods.

Our findings lead us to conclude the following in response to our research questions:

- **What is the performance of the k-NN method in relation to the other methods**? Our results show that the k-NN method performed well when imputing homogeneous Likert data, provided that an appropriate value of *k* was used. It outperformed both Random Draw Substitution and Random Imputation, while both Median Imputation and Mode Imputation performed equally good or slightly better. However, it is clear that our data was favourable both for Median Imputation and Mode Imputation. With a different distribution of response options, these methods could perform worse, whereas the k-NN method should not, given that it is less sensitive to the data distribution.

- **How many donors should preferably be selected**? It is not best to use $k = 1$, as we have seen is common, in all situations. Our results show that using the square root of the number of complete cases, rounded to the nearest odd integer, is a suitable model for *k*.
- **At which amount of missing data is it no longer relevant to use the method**? The outcome of the imputation depends on the number of complete cases more than the amount of missing data. The method was successful even for high proportions of incomplete cases. For example, with six attributes and the IC strategy, the method had close to maximum ability when 95% of the cases were incomplete.
- **Is it possible to decrease the sensitivity to the amount of missing data by allowing imputation from certain incomplete cases as well**? When using the IC strategy, the ability of the method increased substantially compared to the CC strategy for larger amounts of missing data, while there was no negative impact on the quality of the imputations for smaller amounts of missing data. Consequently, the IC strategy seems, from a quality perspective, safe to use in all situations.
- **What effect has the number of attributes (variables) on the results**? The k-NN method proved to scale well to more attributes, as both the precision and the mean square error improved for 12 and 18 attributes compared to six attributes. It is also evident that the other imputation methods are not positively affected by the number of attributes, as they do not make use of the additional amount of information.

References

[1] Aaen, I. (2003). Software Process Improvement: Blueprints versus Recipes. *IEEE Software*, 20, 86-92.

[2] ACM (n.d.). *ACM Guide*. Retrieved March 22, 2005.
Web site: portal.acm.org

[3] ANSI/IEEE Std 830-1984 (1984). *IEEE Guide to Software Requirements Specifications*. Institute of the Electrical and Electronics Engineers.

[4] Aurum, A. and Wohlin, C. (2003). The Fundamental Nature of Requirements Engineering Activities as a Decision-Making Process. *Information and Software Technology*, 45, 945-954.

[5] Baddoo, N. and Hall, T. (2003). De-motivators for Software Process Improvement: an Analysis of Practitioners' Views. *Journal of Systems and Software*, 66, 23-33.

[6] Baeza-Yates, R. and Ribeiro-Neto, B. (1999). *Modern Information Retrieval*. Addison-Wesley.

[7] Basili, V. and Green, S. (1994). Software Process Evolution at the SEL. *IEEE Software*, 11, 58-66.

[8] Bass, L., Clements, P. and Kazman, R. (2003). *Software Architecture in Practice*. Addison-Wesley.

[9] Batista, G. E. A. P. A. and Monard, M. C. (2001). A Study of K-Nearest Neighbour as a Model-Based Method to Treat Missing Data. In *Proceedings of the Argentine Symposium on Artificial Intelligence*, September, Buenos Aires, Argentina, pp. 1-9.

[10] Berander, P. and Wohlin, C. (2004). Differences in Views between Development Roles in Software Process Improvement – A Quantitative Comparison. In *Proceedings of the International Conference on Empirical Assessment in Software Engineering*, May 24-25, Edinburgh, Scotland, pp. 57-66.

[11] Berry, M. (1992). Large Scale Singular Value Computations. *International Journal of Supercomputer Applications*, 6(1), 13-49.

[12] Berry, M. W., Dumais, S. T. and O'Brien, G. W. (1995). Using Linear Algebra for Intelligent Information Retrieval. *SIAM Review*, 37(4), 573-595.

[13] Bohner, S. A. (2002). Extending Software Change Impact Analysis into COTS Compo-nents. In *Proceedings of the Annual NASA*

Goddard Software Engineering Workshop, December 4-6, Greenbelt, Maryland, USA, pp. 175-182.

[14] Bohner, S. A. and Arnold, R. S. (1996). *Software Change Impact Analysis*. IEEE Computer Society Press.

[15] Bohner, S. A. and Gracanin, D. (2003). Software Impact Analysis in a Virtual Environment. In *Proceedings of the Annual NASA Goddard Software Engineering Workshop*, December 2-4, Greenbelt, Maryland, USA, pp. 143-151.

[16] Bosch, J. (2000). *Design & Use of Software Architectures – Adopting and evolving a product-line approach*. Pearson Education.

[17] Bratthall, L., Johansson, E. and Regnell, B. (2000). Is a Design Rationale Vital when Predicting Change Impact? – A Controlled Experiment on Software Architecture Evolution. In *Proceedings of the International Conference on Product Focused Software Process Improvement*, June 20-22, Oulu, Finland, pp. 126-139.

[18] Briand, L. C., Labiche, Y. and O'Sullivan, L. (2003). Impact Analysis and Change Management of UML Models. In *Proceedings of the International Conference on Software Maintenance*, September 22-26, Amsterdam, Netherlands, pp. 256-265.

[19] Bud, R. (1998). Penicillin and the new Elizabethans. *British Journal for the History of Science*, 31(3), 305-333.

[20] Cartwright, M. H., Shepperd, M. J. and Song, Q. (2003). Dealing with Missing Software Project Data. In *Proceedings of the International Software Metrics Symposium*, September 3-5, Sydney, Australia, pp. 154-165.

[21] Chen, K. and Rajich, V. (2001). RIPPLES: Tool for Change in Legacy Software. In *Proceedings of the International Conference on Software Maintenance*, November 6-10, Florence, Italy, pp. 230-239.

[22] Chen, J. and Shao, J. (2000). Nearest Neighbor Imputation for Survey Data. *Journal of Official Statistics*, 16(2), 113-131.

[23] Chen, G. and Åstebro, T. (2003). How to Deal with Missing Categorical Data: Test of a Simple Bayesian Method. *Organizational Research Methods*, 6, 309-327.

[24] Chowdhury, G. G. (1999). *Introduction to Modern Information Retrieval*. Library Association Publishing.

[25] Clarke, S., Harrison, W., Ossher, H. and Tarr, P. (1999). Subject-Oriented Design: Towards Improved Alignment of Requirements, Design and Code. In *Proceedings of the Conference on Object-Oriented Programming, Systems, Languages and Applications*, November 1-5, Denver, Colorado, USA, pp. 325-339.

[26] Cleland-Huang, J., Chang, C. K. and Wise, J. C. (2003). Automating performance-related impact analysis through event based traceability. *Requirements Engineering*, 8(3), 172-182.

[27] Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J. and Little, R. (2003). *Documenting Software Architectures: Views and Beyond*. Addison-Wesley.

[28] Cohen, J. (1960). A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, 20(1), 37-46.

[29] Conradi, R. and Dybå, T. (2001). An Empirical Study on the Utility of Formal Routines to Transfer Knowledge and Experience. In *Proceedings of the Joint European Software Engineering Conference and ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, September 10-14, Vienna, Austria, pp. 268-276.

[30] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K. and Harschman, R. (1990). Indexing by Latent Semantic Analysis. *Journal of the American Society For Information Science*, 41, 391-407.

[31] Downey, R. G. and King, C. V. (1998). Missing Data in Likert Ratings: A Comparison of Replacement Methods. *Journal of General Psychology*, 125(2), 175-191.

[32] Dressel, S. (1990). Reference Bases in Natural Language and Technical Language. In *Workshop Notes of the International Professional Communication Conference*, September 12-14, Guildford, UK, pp. 86-89.

[33] Duda, R. O. and Hart, P. E. (1973). *Pattern Classification and Scene Analysis*. John Wiley & Sons.

[34] Dumais, S. T. (1991). Improving the Retrieval of Information from External Sources. *Behavior Research Methods, Instruments and Computers*, 23(2), 229-236.

[35] Dumais, S. T. (1993). LSI meets TREC: A Status Report. *The First Text REtrieval Conference (TREC1), National Institute of Standards and Technology Special Publication 500-207.*

[36] Egyed, A. (2003). A Scenario-Driven Approach to Trace Dependency Analysis. *IEEE Transactions on Software Engineering*, 29(2), 116-132.

[37] Eick, S. G., Graves, L., Karr, A. F. and Marron, J. S. (2001). Does code decay? Assessing the Evidence from Change Management Data. *IEEE Transactions on Software Engineering*, 27(1), 1-12.

[38] Engels, J. M. and Diehr, P. (2003). Imputation of Missing Longitudinal Data: A Comparison of Methods. *Journal of Clinical Epidemiology*, 56, 968-976.

[39] Engineering Village 2 (n.d.). *Compendex & Inspec*. Retrieved March 22, 2005.
Web site: www.engineeringvillage2.org

[40] Etzkorn, L. H. and Davis, C. G. (1997). Automatically Identifying Reusable OO Legacy Code. *Computer*, 30(10), 66-71.

[41] Fasolino, A. R. and Visaggio, G. (1999). Improving Software Comprehension through an Automated Dependency Tracer. In *Proceedings of the International Workshop on Program Comprehension*, May 5-7, Pittsburgh, Pennsylvania, USA, pp. 58-65.

[42] Fowler, M. (2003). Who Needs an Architect?. *IEEE Software*, 20, 11-13.

[43] Furnas, G. W., Deerwester, S., Dumais, S. T., Landauer, T. K., Harshman, R. A., Streeter, L. A. and Lochbaum, K. E. (1988). Information Retrieval Using a Singular Value Decomposition Model of Latent Semantic Structure. In *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, June 13-15, Grenoble, France, pp. 465-480.

[44] Gallagher, K. B. (1996). Visual Impact Analysis. In *Proceedings of the International Conference on Software Maintenance*, November 4-8, Monterey, California, USA, pp. 52-58.

[45] Gallagher, K. B. and Lyle, J. R. (1991). Using Program Slicing in Software Maintenance. *IEEE Transactions on Software Engineering*, 17(8), 751-761.

[46] Garson, G. D. (n.d.). *PA 765 Statnotes: An Online Textbook*. Retrieved March 3, 2005.
Web site: www2.chass.ncsu.edu/garson/pa765/statnote.htm

[47] Gediga, G. and Düntsch, I. (2003). Maximum Consistency of Incomplete Data via Non-Invasive Imputation. *Artificial Intelligence Review*, 19(1), 93-107.

[48] Glass, R. L., Vessey, I. and Ramesh, V. (2002). Research in Software Engineering: an Analysis of the Literature. *Information and Software Technology*, 44, 491-506.

[49] Gmel, G. (2001). Imputation of Missing Values In the Case of a Multiple Item Instrument Measuring Alcohol Consumption. *Statistics in Medicine*, 20, 2369-2381.

[50] Godfrey, L. W. and Lee, E. H. S. (2000). Secrets from the Monster – Extracting Mozilla's Software Architecture. In *Proceedings of the*

International Symposium on Constructing Software Engineering Tools, June 5, Limerick, Ireland, pp. 15-23.

[51] Hall, T. and Wilson, D. (1997). Views of Software Quality: a Field Report. *IEE Proceedings on Software Engineering*, 144, 111-118.

[52] Haney, F. M. (1972). Module Connection Analysis – A Tool for Scheduling Software Debugging Activities. In *AFIPS Joint Computer Conference*, December 5-7, Anaheim, California, USA, pp. 173-179.

[53] Hoffmann, M., Kühn, N. and Bittner, M. (2004). Requirements for Requirements Management Tools. In *Proceedings of the IEEE International Requirements Engineering Conference*, September 6-10, Kyoto, Japan, pp. 301-308.

[54] Hu, M., Salvucci, S. M. and Cohen, M. P. (1998). Evaluation of Some Popular Imputation Algorithms. In *Proceedings of the Survey Research Methods Section, American Statistical Association*, pp. 308-313.

[55] Huisman, M. (2000). Imputation of Missing Item Responses: Some Simple Techniques. *Quality and Quantity*, 34, 331-351.

[56] Humphrey, W. (1989). *Managing the Software Process*. Addison-Wesley.

[57] IEEE (n.d.). *IEEE Xplore*. Retrieved March 22, 2005.
Web site: ieeexplore.ieee.org

[58] Karlström, D., Runeson, P. and Wohlin, C. (2002). Aggregating Viewpoints for Strategic Software Process Improvement – a Method and a Case Study. *IEE Proceedings on Software Engineering*, 149, 143-152.

[59] Kitchenham, B. A., Pfleeger, S. L., Pickard, L. M., Jones, P. W., Hoaglin, D. C., El Emam, K. and Rosenberg, J. (2002). Preliminary Guidelines for Empirical Research in Software Engineering. *IEEE Transactions on Software Engineering*, 28(8), 721-734.

[60] Klappholz, D., Bernstein, L. and Port, D. (2003). Assessing Attitude Towards, Knowledge of, and Ability to Apply, Software Development Process. In *Proceedings of the Conference on Software Engineering Education and Training*, , Madrid, Spain, pp. 268-278.

[61] von Knethen, A. and Grund, M. (2003). QuaTrace: A Tool Environment for (Semi-) Automatic Impact Analysis Based on Traces. In *Proceedings of the International Conference on Software Maintenance*, September 22-26, Amsterdam, Netherlands, pp. 246-255.

[62] Kolda, T. G. (1997). *Limited-Memory Matrix Methods with Applications*. Ph.D. thesis, University of Maryland, Maryland, USA.

[63] Kotonya, G. and Sommerville, I. (1998). *Requirements Engineering – Processes and Techniques*. John Wiley & Sons.

[64] Lam, W. and Shankararaman, V. (1999). Requirements Change: A Dissection of Management Issues. In *Proceedings of the EuroMicro Conference, vol. 2*, September 8-10, Milan, Italy, pp. 244-251.

[65] Landauer, T. K., Foltz, P. W. and Laham, D. (1998). Introduction to Latent Semantic Analysis. *Discourse Processes*, 25, 259-284.

[66] Law, J. and Rothermel, G. (2003). Whole Program Path-Based Dynamic Impact Analysis. In *Proceedings of the International Conference on Software Engineering*, May 3-10, Portland, Oregon, USA, pp. 308-318.

[67] Lee, L. (2004). "I'm sorry Dave, I'm afraid I can't do that": Linguistics, Statistics, and Natural Language Processing circa 2001. In *Computer Science: Reflections on the Field, Reflections from the Field*, The National Academies Press.

[68] Lee, M., Offutt, J. A. and Alexander, R. T. (2000). Algorithmic Analysis of the Impacts of Changes to Object-Oriented Software. In *Proceedings of the International Conference on Technology of Object-Oriented Languages and Systems*, July 30-August 4, Santa Barbara, California, USA, pp. 61-70.

[69] De Leeuw, E. D. (2001). Reducing Missing Data in Surveys: An Overview of Methods. *Quality and Quantity*, 35, 147-160.

[70] Leffingwell, D. and Widrig, D. (1999). *Managing Software Requirements – A Unified Approach*. Addison-Wesley.

[71] Lehman, M. M., Ramil, J. F., Wernick, P. D., Perry, D. E. and Turski, W. M. (1997). Metrics and Laws of Software Evolution – The Nineties View. In *Proceedings of the International Software Metrics Symposium*, November 5-7, Albuquerque, New Mexico, USA, pp. 20-32.

[72] Lehmann, D. R. and Winer, R. S. (2002). *Product Management, 3rd ed*. McGraw-Hill.

[73] Lindvall, M. (1997). *An Empirical Study of Requirements-Driven Impact Analysis in Object-Oriented Systems Evolution*. Ph.D. thesis no. 480, Linköping Studies in Science and Technology, Linköping, Sweden.

[74] Lindvall, M. and Sandahl, K. (1998). How well do Experienced Software Developers Predict Software Change?. *Journal of Systems and Software*, 43(1), 19-27.

[75] Maciaszek, L. (2001). *Requirements Analysis and System Design – Developing Information Systems with UML*. Addison-Wesley.

[76] Marcus, A. and Maletic, J. I. (2003). Recovering Documentation-to-Source-Code Traceability Links using Latent Semantic Indexing. In *Proceedings of the International Conference on Software Engineering*, May 3-10, Portland, Oregon, USA, pp. 125-135.

[77] Merriam-Webster (n.d.). *Merriam-Webster Online Dictionary.* Retrieved March 7, 2005.
Web site: www.m-w.com

[78] Mockus, A. and Votta, L. G. (2000). Identifying Reasons for Software Changes Using Historic Databases. In *Proceedings of the International Conference on Software Maintenance*, October 11-14, San Jose, California, USA, pp. 120-130.

[79] McFeeley, B. (1996). *IDEALSM: A User's Guide for Software Process Improvement*. Software Engineering Institute, Carnegie Mellon University.

[80] Miller, J. (2004). Statistical significance testing – a panacea for software technology experiments?. *Journal of Systems and Software*, 73, 183-192.

[81] Myrtveit, I., Stensrud, E. and Olsson, U. H. (2001). Analyzing Data Sets with Missing Data: An Empirical Evaluation of Imputation Methods and Likelihood-Based Methods. *IEEE Transactions on Software Engineering*, 27, 999-1013.

[82] Natt och Dag, J., Regnell, B., Carlshamre, P., Andersson, M. and Karlsson, J. (2002). A Feasibility Study of Automated Support for Similarity Analysis of Natural Language Requirements in Market-Driven Development. *Requirements Engineering*, 7, 20-33.

[83] Natt och Dag, J., Regnell, B., Gervasi, V. and Brinkkemper, S. (2005). A Linguistic-Engineering Approach to Large-Scale Requirements Management. *IEEE Software*, 22(1), 32-39.

[84] Ngo-The, A. and Ruhe, G. (2005). Decision Support in Requirements Engineering. In A. Aurum and C. Wohlin (Eds.), *Engineering and Managing Software Requirements*. Springer-Verlag.

[85] Nicholas, J. M. (2001). *Project Management for Business and Technology: Principles and Practise, 2nd ed*. Prentice Hall.

[86] O'Neal, J. S. and Carver, D. L. (2001). Analyzing the Impact of Changing Requirements. In *Proceedings of the International Conference on Software Maintenance*, November 6-10, Florence, Italy, pp. 190-195.

[87] Orlikowski, W. J. and Baroudi, J. J. (1991). Studying Information Technology in Organizations: Research Approaches and Assumptions. *Information Systems Research*, 2(1), 1-28.

[88] Orso, A., Apiwattanapong, T., Law., J., Rothermel, G. and Harrold, M. J. (2004). An Empirical Comparison of Dynamic Impact Analysis Algorithms. In *Proceedings of the International Conference on Software Engineering*, May 23-28, Edinburgh, Scotland, UK, pp. 491-500.

[89] Pfleeger, S. L. (1998). *Software Engineering: Theory and Practise, intl. ed.* Prentice Hall.

[90] Porter, M. (1997). An Algorithm for Suffix Stripping. In K. Sparck Jones and P. Willet (Eds.), *Readings in Information Retrieval*. Morgan Kaufmann Publishers.

[91] Pour G., Griss, M.L. and Lutz M. (2000). The Push to Make Software Engineering Respectable. *Computer*, 33(5), 35-43.

[92] PROFES (n.d.). *User Manual - Final Version 1999*. Retrieved March 3, 2005.
Web site: www.vtt.fi/ele/profes/PUMv10.pdf

[93] Raaijmakers, Q. A. W. (1999). Effectiveness of Different Missing Data Treatments in Surveys with Likert-Type Data: Introducing the Relative Mean Substitution Approach. *Educational and Psychological Measurement*, 59(5), 725-748.

[94] Rainer, A. and Hall, T. (2002). Key Success Factors for Implementing Software Process Improvement: a Maturity-Based Analysis. *Journal of Systems and Software*, 62, 71-84.

[95] Ramesh, B. and Jarke, M. (2001). Towards Reference Models for Requirements Traceability. *IEEE Transactions on Software Engineering*, 27(1), 58-93.

[96] Ren, X., Shah, F., Tip, F., Ryder, B. and Chesley, O. (2004). Chianti: A Tool for Change Impact Analysis of Java Programs. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications*, October 26-28, Vancouver, Canada, pp. 432-448.

[97] Robertson, S. and Robertson, J. (1999). *Mastering the Requirements Process*. Addison-Wesley.

[98] Robson, C. (2002). *Real World Research, 2nd ed.* Blackwell Publishing.

[99] Rohde, D. (n.d.). *SVDLIBC*. Retrieved March 17, 2005.
Web site: tedlab.mit.edu/~dr

[100] Russ-Eft, D. F. (2004). So what is research anyway?. *Human Resource Development Quarterly*, 15(1), 1-4.

[101] Ryan, K. (1993). The Role of Natural Language in Requirements Engineering. In *Proceedings of IEEE International Symposium on Requirements Engineering*, January 4-6, San Diego, California, USA, pp. 240-242.

[102] Sande, I. G. (1983). Hot-Deck Imputation Procedures. In W. G. Madow and I. Olkin (Eds.), *Incomplete Data in Sample Surveys, vol. 3, Proceedings of the Symposium*. Academic Press.

[103] Sarwar, B., Karypis, G., Konstan, J. and Riedl, J. (2002). Incremental Singular Value Decomposition Algorithms for Highly Scalable Recommender Systems. In *Proceedings of the International Conference on Computer and Information Technology*, December 27-28, Dhaka, Bangladesh.

[104] Scheffer, J. (2002). Dealing with Missing Data. *Research Letters in the Information and Mathematical Sciences*, 3, 153-160.

[105] Shahmehri, N., Kamkar, M. and Fritzson, P. (1990). Semi-Automatic Bug Localization in Software Maintenance. In *Proceedings of the Conference on Software Maintenance*, November 26-29, San Diego, California, USA, pp. 30-36.

[106] Siegel, S. and Castellan Jr., N. J. (1988). *Nonparametric Statistics for the Behavioral Sciences, intl. ed*. McGraw-Hill.

[107] Smyth, B. (2003). *Computing Patterns in Strings*. Pearson Education.

[108] Sneed, H. M. (2001). Impact Analysis of Maintenance Tasks for a Distributed Object-Oriented System. In *Proceedings of the International Conference on Software Maintenance*, November 6-10, Florence, Italy, pp. 190-195.

[109] Software Engineering Institute (n.d.). *How Do You Define Software Architecture?*. Retrieved March 5, 2005.
Web site: www.sei.cmu.edu/architecture/definitions.html

[110] Sommerville, I. (2001). *Software Engineering, 6th ed*. Addison-Wesley.

[111] Sommerville, I. and Sawyer, P. (1997). *Requirements Engineering – A Good Practice Guide*. John Wiley & Sons.

[112] Song, Q., Shepperd, M. and Cartwright, M. H. (2005). A Short Note on Safest Default Missingness Mechanism Assumptions. *Journal of Empirical Software Engineering*, 10, 235-243.

[113] Strike, K., El Emam, K. and Madhavji, N. (2001). Software Cost Estimation with Incomplete Data. *IEEE Transactions on Software Engineering*, 27, 890-908.

[114] Svahnberg, M. (2003). A Study on Agreement Between Participants in an Architecture Assessment. In *Proceedings of the International Symposium on Empirical Software Engineering*, September 30-October 1, Rome, Italy, pp. 61-71.

[115] Tip, F., Jong, D. C., Field, J. and Ramlingam, G. (1996). Slicing Class Hierarchies in C++. In *Conference on Object-Oriented Programming*, October 6-10, San Jose, California, USA, pp. 179-197.

[116] Troyanskaya, O., Cantor, M., Sherlock, G., Brown, P., Hastie, T., Tibshirani, R., Botstein, D. and Altman, R. B. (2001). Missing Value Estimation Methods for DNA Microarrays. *Bioinformatics*, 17, 520-525.

[117] Turver, R. J. and Munro, M. (1994). An Early Impact Analysis Technique for Software Maintenance. *Journal of Software Maintenance Research and Practice*, 6(1), 35-52.

[118] Weinberg, G. M. (1983). Kill That Code. *Infosystems*, 30, 48-49.

[119] Weiser, M. (1979). *Program Slices: Formal, Psychological, and Practical Investigations of an Automatic Program Abstraction Method*. Ph.D. thesis, University of Michigan, Michigan, USA.

[120] Wiegers, K. E. (2003). *Software Requirements*. Microsoft Press.

[121] Wilson, D. R. and Martinez, T. R. (1997). Improved Heterogeneous Distance Functions. *Journal of Artificial Intelligence Research*, 6, 1-34.

[122] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A. (2000). *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers.

[123] Yau, S. S. and Collofello, J. S. (1980). Some Stability Measures for Software Maintenance. *IEEE Transactions on Software Engineering*, 6(6), 545-552.

[124] Zahran, S. (1998). *Software Process Improvement: Practical Guidelines for Business Success*. Addison-Wesley.

[125] Zelkowitz, M. V. and Wallace, D. R. (1998). Experimental models for validating technology. *Computer*, 31(5), 23-31.

[126] Zhao, J. (1998). Applying Slicing Technique to Software Architectures. In *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems*, August 10-14, Monterey, California, USA, pp. 87-98.