Empirical Investigations of Software Process Improvement

Joost Schalken
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Graduate School for Information and Knowledge Systems.

Promotiecommissie:
prof.dr. J.C. van Vliet (promotor, Vrije Universiteit Amsterdam)
prof.dr. S. Brinkkemper (promotor, Universiteit Utrecht)
dr. R.S. Batenburg (Universiteit Utrecht)
dr. T. Dingsøyr (Norwegian University of Science and Technology)
prof.dr.ir. M.J.I.M. van Genuchten (Technische Universiteit Eindhoven)
prof.dr. C. Verhoeef (Vrije Universiteit Amsterdam)
Empirical Investigations of Software Process Improvement

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor aan de Vrije Universiteit Amsterdam, op gezag van de rector magnificus prof.dr. L.M. Bouter, in het openbaar te verdedigen ten overstaan van de promotiecommissie van de faculteit der Exacte Wetenschappen op maandag 17 december 2007 om 10.45 uur in de aula van de universiteit, De Boelelaan 1105

door

Joost Joannes Petrus Schalken

debraen te Oss
promotoren: prof.dr. J.C. van Vliet
prof.dr. S. Brinkkemper
Preface

Performing research is a solitary activity, however it is fortunately by no means work that is performed in perfect seclusion. Because empirical software engineering research, by necessity, takes place in an industrial setting, I had the advantage of having colleagues both at the university and at the company at which I performed my research.

During my studies these colleagues offered advice, guidance, support, fascinating questions or just lend an ear to issues I was struggling with. Without false humility I can state that without the help of many others I would not have been able to complete my Ph.D. studies. In Sect. 1.6 I would like to thank the people who contributed to my research. Besides colleagues who helped me performing my research, there were people whose influence was not limited to my work, but who also had a profound effect on my personal life.

Let me start by thanking my two Ph.D. supervisors: prof.dr. Hans van Vliet and prof.dr. Sjaak Brinkkemper. Sjaak’s enthusiasm helped me always to push a little further and his insight into the dynamics of the scientific community helped me understand how the politics of science works. As a daily supervisor, Hans always had time for me whenever problems seemed unsolvable and helped me see when I got trapped in my own perfectionism. Besides our conversations about research, I also enjoyed our discussions about literature. Hans exemplifies to me how a good Doctorvati (German for promotor) should be.

I also want to thank my supervisors and colleagues at the ABN AMRO Bank. I want to express my special gratitude to Jean Kleijnen for all his support, interest in my progress and for providing me with the opportunity to investigate a large process improvement program. I also want to thank Frank Harmsen of Capgemini for his guidance at the beginning of my Ph.D. research.

In addition I want to thank my fellow software engineering Ph.D. students (Niels Veerman, Rik Farenhorst, Remco de Boer, Viktor Clerc, Erald Kulk, Peter Kampstra, Lukasz Kwiatkowski, Laurenz Eveleens, Adam Vanya and Qing Gu), staff members of the software engineering group (Patricia Lago, Steven Klusener, Johan Hoorn, Rob Peters, René Krikhaar and Chris Verhoef) and my roommates at the VU (Clemens Grabmeyer, Willem van Hage and Machiel Janssen) for the inspiring discussions and the opportunity to escape for a few minutes from my research.

Lastly I thank my family, my parents, my brother and my friends for their encouragements and interest in my work. Special thanks go to my parents for supporting me throughout my studies. But this word of thanks would not be complete without thanking Suzanne for all her love and patience when I was writing my thesis.
# Contents

1 Introduction ........................................................................................................... 1
   1.1 Research Problem ............................................................................................ 2
      1.1.1 The Software Crisis .................................................................................. 2
      1.1.2 Software Process Improvement ................................................................. 3
      1.1.3 Empirical Evidence of SPI Efficacy ......................................................... 4
      1.1.4 Lack of Existing Empirical Evidence ...................................................... 5
      1.1.5 Difficulties in Evaluating SPI Programs .................................................. 6
   1.2 Research Context .............................................................................................. 8
   1.3 Research Questions .......................................................................................... 10
   1.4 Main Contributions ......................................................................................... 11
   1.5 Structure of this Thesis ................................................................................... 12
   1.6 Acknowledgements ......................................................................................... 14
   1.7 Publications .................................................................................................... 15

2 Software Process Improvement and Process Assessment ...................................... 17
   2.1 Software Process Improvement ................................................................... 19
      2.1.1 Software Development Methods .............................................................. 20
      2.1.2 Process Frameworks .............................................................................. 20
      2.1.3 Process Improvement Approaches ......................................................... 21
   2.2 Process Improvement Approaches at ABN AMRO Bank .............................. 21
      2.2.1 Capability Maturity Model ..................................................................... 21
      2.2.2 Dynamic Systems Development Method .............................................. 24
   2.3 Process Assessment ....................................................................................... 26
      2.3.1 Software Measurement and Metrics ....................................................... 26

3 Effects of SPI on the Productivity of Software Development .............................. 27
   3.1 Introduction .................................................................................................... 27
      3.1.1 Ineffective Productivity Data Analysis .................................................... 28
      3.1.2 Research Questions ................................................................................ 29
      3.1.3 Contribution ............................................................................................ 29
      3.1.4 Outline .................................................................................................... 30
   3.2 Related Work .................................................................................................. 30
   3.3 Research Methodology ................................................................................... 32
      3.3.1 Research Design .................................................................................... 32
      3.3.2 Analysis Evaluation Criteria .................................................................. 32
      3.3.3 Data Collection ..................................................................................... 33
## Contents

3.4 Analysis Methods ............................................. 34
  3.4.1 Productivity Indices .................................... 34
  3.4.2 Regression Models ...................................... 36
  3.4.3 Hierarchical Linear Models ............................. 39

3.5 Results ......................................................... 41
  3.5.1 Productivity Indices .................................... 41
  3.5.2 Linear Regression Models ............................... 42
  3.5.3 Hierarchical Model Approach ........................... 45
  3.5.4 Power Simulation ....................................... 46

3.6 Conclusions .................................................. 49

4 Effects of Facilitated Workshops on Requirements Engineering Productivity 51
  4.1 Introduction .................................................. 51
    4.1.1 Evaluating the Effectiveness of Facilitated Workshops 52
    4.1.2 Research Questions and Results ........................ 53

  4.2 Related work ................................................ 53

  4.3 Research Methodology ...................................... 54
    4.3.1 Research Context ...................................... 54
    4.3.2 Comparing different Development Cycles ................ 55
    4.3.3 Data Collection ....................................... 55

  4.4 Discussion of results ..................................... 56
    4.4.1 Effort .................................................. 56
    4.4.2 Duration ............................................... 59
    4.4.3 Satisfaction .......................................... 59
    4.4.4 Threats to Validity .................................. 61

  4.5 Conclusion .................................................. 62

5 Exploring Information in Project Evaluation Databases 65
  5.1 Introduction .................................................. 65
  5.2 Related Work ................................................ 68
  5.3 Method ....................................................... 68
    5.3.1 Identify Success Factors ............................... 69
    5.3.2 Selection of Project Evaluations ....................... 70
    5.3.3 Identify Project Factors ............................... 70
    5.3.4 Interpret Project Evaluations .......................... 72
    5.3.5 Analyze Correlations .................................. 72

  5.4 Case Study: ABN AMRO Bank ............................... 74
    5.4.1 Context of Case Study .................................. 74
    5.4.2 Data Collection ........................................ 75
    5.4.3 Results ................................................ 76
    5.4.4 Discussion ............................................. 76

  5.5 Case Study: Kongsberg Spaceteck .......................... 79
    5.5.1 Context of Case Study .................................. 79
    5.5.2 Data Collection ........................................ 79
    5.5.3 Data Analysis .......................................... 81
    5.5.4 Results ................................................ 82
    5.5.5 Qualitative Analysis ................................... 82
## Contents

- **5.5.6** Quantitative Analysis ........................................................................ 83
- **5.5.7** Discussion ..................................................................................... 84
  - 5.6 Conclusions ....................................................................................... 86

6 Validating Information in Project Evaluation Databases ............................... 89
  - 6.1 Introduction ....................................................................................... 89
    - 6.1.1 Threats to Validity ....................................................................... 90
  - 6.2 Case Study: ABN AMRO Bank ............................................................ 93
    - 6.2.1 Context of Case Study ................................................................. 93
    - 6.2.2 Data Collection ........................................................................... 93
    - 6.2.3 Results ....................................................................................... 95
  - 6.3 Case Study: Kongsberg Spacetec ......................................................... 97
    - 6.3.1 Context of Case Study ................................................................. 97
    - 6.3.2 Data Collection ........................................................................... 97
    - 6.3.3 Results ....................................................................................... 99
    - 6.3.4 Quantitative Analysis ................................................................... 99
    - 6.3.5 Discussion .................................................................................. 100
  - 6.4 Conclusions ....................................................................................... 102

7 Measurement of Infrastructural Projects ....................................................... 103
  - 7.1 Introduction ....................................................................................... 104
    - 7.1.1 IT Infrastructure Defined .............................................................. 104
    - 7.1.2 Related Work ............................................................................... 105
    - 7.1.3 Structure of Chapter .................................................................... 107
  - 7.2 Infrastructure Effort Points .................................................................. 107
    - 7.2.1 Theory Design ............................................................................ 107
    - 7.2.2 Hardware Effort Points ................................................................. 108
    - 7.2.3 Software Effort Points ................................................................. 110
    - 7.2.4 Infrastructure Effort Points .......................................................... 111
  - 7.3 Calibration Process ............................................................................ 111
  - 7.4 Feasibility Study ................................................................................ 112
  - 7.5 Measurement Validation ..................................................................... 113
    - 7.5.1 Validity Requirements ................................................................. 113
  - 7.6 Conclusions ....................................................................................... 115

8 Influence of Research Methodology on Research Outcomes ......................... 117
  - 8.1 Introduction ....................................................................................... 117
    - 8.1.1 Evidence-Based Software Engineering ........................................ 118
    - 8.1.2 Epistemological Frameworks ......................................................... 120
    - 8.1.3 Incremental Approach .................................................................. 121
    - 8.1.4 Outline ....................................................................................... 122
  - 8.2 Rationale ........................................................................................... 122
    - 8.2.1 Decisions Based on Evidence ....................................................... 122
    - 8.2.2 Different Forms of Data ............................................................... 123
    - 8.2.3 Metrics ...................................................................................... 124
    - 8.2.4 More than Metrics ...................................................................... 125
  - 8.3 Analysis Method ................................................................................ 125
    - 8.3.1 Exploratory Cycle ....................................................................... 127
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.3.2</td>
<td>Confirmatory Cycle</td>
<td>129</td>
</tr>
<tr>
<td>8.3.3</td>
<td>Validation of Insights</td>
<td>129</td>
</tr>
<tr>
<td>8.4</td>
<td>Related Work</td>
<td>130</td>
</tr>
<tr>
<td>8.5</td>
<td>Conclusions</td>
<td>131</td>
</tr>
<tr>
<td>9</td>
<td>Empirical Software Engineering: a Personal Perspective</td>
<td>133</td>
</tr>
<tr>
<td>9.1</td>
<td>The use of Research Methodology Knowledge</td>
<td>133</td>
</tr>
<tr>
<td>9.2</td>
<td>Lessons Learnt</td>
<td>134</td>
</tr>
<tr>
<td>9.2.1</td>
<td>Research Methodology also for Software Engineers</td>
<td>134</td>
</tr>
<tr>
<td>9.2.2</td>
<td>Research Context Matters</td>
<td>135</td>
</tr>
<tr>
<td>9.2.3</td>
<td>Simple Statistics</td>
<td>135</td>
</tr>
<tr>
<td>9.2.4</td>
<td>Real-life Data is Messy</td>
<td>136</td>
</tr>
<tr>
<td>9.2.5</td>
<td>Why was the Data Collected</td>
<td>137</td>
</tr>
<tr>
<td>9.3</td>
<td>Conclusions</td>
<td>137</td>
</tr>
<tr>
<td>10</td>
<td>Conclusions</td>
<td>139</td>
</tr>
<tr>
<td>10.1</td>
<td>Research Questions revisited</td>
<td>140</td>
</tr>
<tr>
<td>10.1.1</td>
<td>How can empirical data be used to get feedback about the effects of software process improvement?</td>
<td>141</td>
</tr>
<tr>
<td>10.1.2</td>
<td>How can empirical data be used to get feedback about the processes that still have room for improvement?</td>
<td>141</td>
</tr>
<tr>
<td>10.1.3</td>
<td>Integrating the pieces of the puzzle</td>
<td>142</td>
</tr>
<tr>
<td>10.1.4</td>
<td>The benefits of SPI</td>
<td>142</td>
</tr>
<tr>
<td>10.2</td>
<td>Further Work</td>
<td>144</td>
</tr>
</tbody>
</table>

Samenvatting  

Bibliography  

SIKS Dissertation Series
Chapter 1

Introduction

Society as a whole, and large organisations in particular, have become more and more dependent on software systems to function efficiently. And although senior executives of large organisations will quickly acknowledge the necessity of software and information systems, the real added value of the information systems remains unclear to them [Dibbern et al., 2004, p. 7]. The development of software is to a large extent a creative task, that has proven to be hard to manage. Some have claimed that the management of software development has remained an art [Chang, 2001] (as opposed to a science or craft). The immaturity of software management has made software and information system development risky. Too often projects overrun their budget or schedule or do not deliver what was expected. The high risk of software and information system development combined with company executives failing to understanding the benefits and complexities of software development has put pressure on IT professionals to improve their performance.

To improve the performance of IT, one might be tempted to solve isolated problems on a project-by-project or a system-by-system basis. Perhaps this approach will lead to some quick wins on the short run. However on the long run, by solving problems on a project-by-project or problem-by-problem basis, one runs the risk of not solving underlying causes that create the problems in one’s projects. By not solving the underlying causes one has to repeat the problem solving process time and time again.

Another approach to improve the predictability and productivity of projects and the quality of delivered products is to focus on the development process instead of on the product itself. A software process can be defined as “a set of activities, methods, practices, and transformations that people use to develop and maintain software and the associated products (e.g., project plans, design documents, code, test cases, and user manuals)” [Paulk et al., 1993, p. 3]. The idea behind focusing on the development process instead of on the product itself is that if you improve a product you only improve a single product, whereas if you improve the process that produces the product, you effectively improve each project henceforth. This process oriented approach stems from the quality management movement headed by Stewart [Shewhart, 1939], Deming [Deming, 1986], and Juran [Juran, 1988].

Improving the software development process, instead of directly improving individual software systems or projects, is called software process improvement (SPI). Either by implementing the processes of well-performing companies, or by letting empirical
data guide the improvement process, the management and engineering processes can be improved.

For software process improvement to actually deliver results to an organisation, it needs to be successfully implemented it that organisation. Commitment to SPI from both IT staff and executive management is a crucial factor in successfully implementing SPI [Wiegers, 1998].

Unfortunately it is not enough to obtain commitment from executive management at the beginning of an SPI programme. This commitment needs to be maintained during the entire SPI programme [Humphrey, 1989]. Commitment to SPI programmes does not consist of a single dimension, it is a complex concept that consists of multiple facets that change over time [Abrahamsson, 2002]. In the beginning of this chapter we have noted that executives do not fully understand the value and the complexities of IT. As software process improvement is potentially even more elusive than a normal IT project, it cannot come as a surprise that executive managers do not fully comprehend SPI initiatives [Reiblein and Symons, 1997]. Reiblein and and Symons postulate that the lack of appreciation for SPI and the lack of funds made available to SPI can be explained by the fact that SPI practitioners fail to link the outcomes of SPI programmes to the business goals of an organisation.

Measuring the outcomes of an SPI program can help in maintaining commitment from executive management. Measurement programs can help to visualise the progress of an SPI program against visions and plans [Mathiassen et al., 2005], allowing executive management to steer the SPI program and stay involved. Measurement of SPI programs is not easy however: "Many organizations have difficulty establishing measurements that are relevant, meaningful, valid, and actually used as a basis for management decision and intervention" [Mathiassen et al., 2005].

In this Ph.D. thesis we will examine how one can interpret and use the data from both measurement programs and other sources of empirical data to help steer an SPI program and help practitioners retain commitment from senior executives.

1.1 Research Problem

1.1.1 The Software Crisis

Since the beginning of the software engineering discipline customers have been dissatisfied with the outcomes, performance and controllability of IT projects. In 1968 there was a NATO conference that gave birth to the term software engineering. Attendants to that conference observed "the tendency for there to be a gap, sometimes a rather large gap, between what was hoped for from a complex software system, and what was typically achieved" [Naur and Randell, 1968, p. 119]. Dijkstra was the first to name this gap, between customer expectancies and what could be delivered, the software crisis. The perception of the software industry being in a crisis has proved not to be the starting problems of a young profession, instead the software industry has been perceived as being in a constant state of crisis [Gibbs, 1994], a perception that has survived until today [Knight, 2005].

To substantiate the existence of the perceived overall software crisis, many studies have either cited a study by the U.S. Government Accounting Office on nine projects from 1979 or the CHAOS report from 1994 by the Standish Group [Standish Group,
1.1. Research Problem

The evidence that is cited to support the notion of a software crisis is not without criticism. Blum [Blum, 1991] (as cited in [Glass, 2006]) has shown that the GAO Study has been misinterpreted and the CHAOS report has been questioned by Glass [Glass, 2006] and Jørgenson [Jørgensen and Moløkken-Østvold, 2006]. Still, other non-refuted studies [Jenkins et al., 1984, Phan et al., 1988, Bergeron and St-Arnaud, 1992] (as cited in [Glass, 2006, Jørgensen and Moløkken-Østvold, 2006]) indicate serious overruns of costs in IT projects (Jenkins et al. 34% [Jenkins et al., 1984], Phan et al. 33% [Phan et al., 1988] and Bergeron and St-Arnaud 33% [Bergeron and St-Arnaud, 1992]).

1.1.2 Software Process Improvement

Software process improvement has been proposed as a strategy to improve the state of practice in software and information systems engineering. SPI standardises work processes, tailored to the organisation’s specific circumstances, based on best practices that are derived from the own organisation, academia or exemplary, well-performing organisations. By standardising work processes, projects can be more effectively managed, risks are reduced, and higher quality and productivity can be achieved.

SPI comes in two forms [Card, 1991, van Solingen and Berghout, 1999]: the analytical approach and the benchmarking approach.

The analytical approach uses both qualitative and quantitative investigations to understand software development and maintenance projects. Based on a thorough understanding of the processes guiding the projects, problems and points of improvement can be identified. Experimentation is used to test whether changes to the development and maintenance processes yield the improvements that were expected. An example of an analytical approach to software process improvement is the Quality Improvement Paradigm (QIP) [Basili and Weiss, 1994].

The advantage of analytical improvement approaches is that they are rational, fact-based methods that can objectively demonstrate their usefulness in the context of an organisation. Unfortunately the application of the analytical approach to software process improvement requires the organisation to have the capability to measure its IT projects and also requires some maturity from the organisation (as without structure, the gathered data cannot be interpreted). When software process improvement methods were introduced, only a few organisations possessed this required IT measurement capability and process maturity [Card, 1991]. Today more organisations have mature IT processes, but many organisations still are unable to reliably measure their IT processes [Mathiassen et al., 2005].

Instead of letting empirical evidence guide the improvement process, the benchmarking paradigm bases its improvement activities on best-practice models. By examining well performing IT organisations, their common features are identified. These common features can either be management processes, engineering processes, organisational aspects, and so forth. The identified best-practices are ranked, based on the necessity or complexity of the practices. These ranked best-practices are in turn input to a best-practice model that forms the basis of the improvement activities. These best-practice models often include levels of maturity, implying an incremental introduction of the best-practices.

When taking the benchmarking approach to software process improvement, an organisation compares its own software development and maintenance processes to a best-
practice model. When deviations from the best-practice model are identified, an organi-
sation either justifies why it deviates from the best-practice model or (when a good reason
is lacking) the organisation adjusts its own processes to incorporate the missing practices.

Examples of best-practice models are the Capability Maturity Model (CMM) [Paulk
et al., 1993,Paulk et al., 1995] and its successor the Capability Maturity Model Integrated
(CMMI) [CMMI Product Development Team, 2000], both developed by the Carnegie
Mellon University’s Software Engineering Institute. Alternative best-practice models are
the BOOTSTRAP approach [Kuvaja et al., 1994] and the ISO-SPICE model [ISO/IEC,
1998].

Many software process improvement methodologies offer not only a best-practice
model, but also an improvement model. These improvement models offer guidance to
implement the practices of the best-practice model and offer advice on how to diagnose
problems in the current processes. An example of an improvement model is the IDEAL
model [Gremba and Myers, 1997] that is associated with the CMM model. IDEAL stands
for Initiating, Diagnosing, Establishing, Acting and Learning.

1.1.3 Empirical Evidence of SPI Efficacy

The benchmarking approaches’ validity depends on the validity of the practices in the
best practice model. Validity in this context means whether the proposed practices re-
ally bring improved organisational performance, such as higher quality software, more
functionality delivered or less costs. Quantification of the results from software process
improvement is challenging [Herbsleb et al., 1994, p. 41].

Although the estimation of the effects of software process improvement is difficult,
it is none the less very important, as the implementation of process improvement can
be expensive. Costs to execute process improvement include consultancy, training, as-
essment, creation of a standard process and quality assurance [Rico, 2002]. Companies
can only justify the costs of implementing a process improvement initiative if the bene-
fits outweigh the investments. In other words the process improvement program should
lead to large enough improvements in productivity, time-to-market, quality and/or better
controllability of IT.

The benefits of evidence collection about the effects of software process improvement
are not limited to facilitate evaluation after the implementation of a SPI program. A
second reason to monitor the effects of SPI is to ensure continued commitment to the
SPI program from executive management. The concepts and vocabulary used by the
IT staff are usually incomprehensible to executive managers. Concepts like schedules,
quality and especially productivity on the other hand are comprehensible to executive
managers. To explain the results to executive management, IT managers and SPI staff
need to express themselves in concepts that are comprehensible to executives.

Return on investment (ROI) is a decision instrument often used by senior manage-
ment. ROI ratios function as a scale-free indicator for the attractiveness of an investment
proposition, as it expresses the amount of money gained in relation to the amount of
money invested. Before the return of investment can be calculated, one should know the
financial value of not only the required investment but also of the benefits. Therefore the
advantage of improved productivity, time-to-market, productivity quality and/or lower
schedule risks should be translated back into financial terms.

IT managers therefore need to be explain which ROI to expect from software process
improvement and they have a need to be able to establish the actual ROI of a process improvement program after completion [van Solingen, 2004].

At the moment different methods for calculating ROI are in use [Galin and Avrahami, 2006], no single method has gained the status of de facto standard. And as a result ROI calculations for process improvement remain open to debate, c.f. the criticism from Sassenburg [Sassenburg, 2002] on Rico’s work on ROI calculations [Rico, 2002].

The lack of consensus on calculation rules is not the largest problem concerning the decision making about SPI. The main problem is the lack of independent, objective evaluations and studies of the outcomes of software process improvement.

1.1.4 Lack of Existing Empirical Evidence

To convince senior management of the potential of software process improvement, it would help if a large database of previous experiences with SPI was available. Such a database would help rational, fact-based decision making regarding investment decisions in software process improvement. Without those evaluations, management has no way of knowing which improvements to expect and whether these improvements justify the investments that are required. The perception of IT being in a constant state of crisis, combined with the uncertainty regarding the outcomes of SPI will not make many executives enthusiastic to invest in SPI programs.

The lack of available empirical data on the effectiveness of certain IT processes does not only hinder organisations in their decision making regarding SPI programs, it has also hindered the development of SPI programs. Most process improvement models have been constructed based on approaches whose effectiveness has not been rigorously and scientifically demonstrated, nor by a sound theoretical basis, nor by sound empirical evidence. Instead, too often we have relied on gut feeling and anecdote, the opinions of experts, even on flawed research [Fenton et al., 1994, Pfleeger, 1994]. This has led to “standards [that] are based on conventional wisdom” [Pfleeger et al., 1994]. Although conventional wisdom does not have to be wrong, from a scientific point of view these standards that lack a scientific basis do warrant a close empirical evaluation.

For most process improvement models little empirical data has been published on the improvements in quality, productivity or time-to-market that was gained by software process improvement. Of all the benchmarking-based improvement methods the Capability Maturity Model [Paulk et al., 1993, Paulk et al., 1995] and its successor the Capability Maturity Model Integrated [CMMI Product Development Team, 2000] are the most familiar, yet there is surprisingly little data available on benefits of CMM. After extensive search for their meta-analysis, Galin and Avrahami [Galin and Avrahami, 2005, Galin and Avrahami, 2006] were only able to identify 19 empirical studies of CMM. And even those 19 studies did not discuss all factors (product quality, time-to-market, productivity and schedule risk).

The most dire lack of publicly available empirical data on software process improvement exists in the domain of commercial software development. This can be explained by the fact that companies have a commercial interest to keep such data private. Disclosing empirical data on software development practices could damage the public image of the company involved or could give competitors a competitive advantage. In addition to the commercial interests to refrain from publishing available empirical data, there are few companies that possess extensive empirical data on SPI as there are less companies that
have extensive experience with process improvement. Although SPI has been invented in development laboratories of large corporations (i.e. ITT and IBM [Jones, 2000, pp. 24–29]), the United States government (especially the Department of Defence) was the first to promote software process improvement on a larger scale. Until five to ten years ago there existed no commercial incentive to entice IT departments and service providers to use the CMM or CMMI to improve their processes or to benchmark their performance. This is why currently most data on software process improvement programs comes from the domain of military [Goldenson and Herbsleb, 1995] and large government programs.

The Capability Maturity Model and its successor the Capability Maturity Model Integrated have traditionally been used in large organisations that complete their projects in a highly linear fashion [Anderson, 2005]. The wording of the Capability Maturity Model seems to imply that only heavy, linear processes with extensive documentation can be assessed as a mature process [Wiegers, 1996, Kulpa, 1998]. Process improvement practitioners risk forgetting the warning words in the introduction of the CMM book [Paulk et al., 1995] that CMM processes should not be introduced in order to satisfy a level, yet they should only be introduced to improve the actual practice.

Currently the methodological landscape has shifted from the linear development methods, such as the waterfall method [Royce, 1970], to more agile methods [Cunningham, 2001], such as eXtreme Programming (XP) [Beck, 1999]; Crystal [Cockburn, 2000]; Feature Driven Development [Palmer and Felsing, 2002]; Dynamic Systems Development Method (DSDM) [Stapleton, 2002]; Adaptive Software Development [Highsmith, 2000]; and the Rational Unified Process [Jacobsen et al., 1994]. It remains a question whether the Capability Maturity Model-based software process improvement is also successful in organisations that use agile methods.

Without sufficient empirical data about mainstream software process improvement models (or software development methods, for that matter) IT professionals and managers are unable to determine if they should invest in process improvement. It is even more difficult for those professionals and managers to make an informed choice for a specific process improvement model that would be suitable for their type of problems and for their type of organisation.

Continuous monitoring of SPI programs, using empirical data from metrics programs and other sources of data, will therefore not only help an organisation keep its SPI program on track and its executive managers committed, it will also help the entire IT community by providing more empirical data on the effects of software process improvement models and methods.

Before such continuous monitoring of SPI program will occur, some of the problems facing organisations that wish to monitoring their SPI program (c.f. [Mathiassen et al., 2005]) will need to taken care of though.

### 1.1.5 Difficulties in Evaluating SPI Programs

According to Herbsleb et al. quantitative evaluations of Software Process Improvement are difficult for three reasons [Herbsleb et al., 1994, p. 41]: 1. measuring the efficiency of projects in a meaningful way is difficult 2. tracking the real change of project efficiency over time is challenging 3. establishing a causal link between efficiency change and process improvement is hard.

Interpreting collected empirical data on software processes efficiency is difficult, be-
1.1. Research Problem

cause the collected data should be consistent and it should be clear which processes are described by the data. In large IT organisations one finds variations in working processes across business units (which makes it unclear which process is being described by a set of data) and there exists a risk that the corporate data collection procedures are misunderstood (which makes the collected data inconsistent) [Herbsleb and Grinter, 1998]. The differences in measurement definitions across an organisation make the interpretation of the results of the process improvement hard. This effect has been observed in a SPI research program that has been executed at Philips [Trienekens et al., 2007].

Measuring the efficiency of IT projects is only feasible for a fraction of all IT projects, because accepted, reliable productivity measures are only available for software development and maintenance (e.g. lines of code per man month or function points per man month). The activities of current IT departments are however not limited to in-house development of software components. Take for example the increasing reliance on common of the shelf (COTS) components when developing information systems [Stensrud and Myrtveit, 2003]. The productivity of installing, configuring and integrating those components becomes an important factor, yet no suitable metric is available to measure the performance of installing, configuring and integrating COTS components. The state of practice in software measurement is therefore able to identify efficiency changes in only a fraction of all IT projects.

Making meaningful comparisons between the efficiency of different projects is difficult, because no two IT projects are the same. It is for example widely known that the efficiency and productivity decrease as the size of a project increases, an effect that is known as variable returns to scale (as opposed to constant returns to scale). This makes it hard to compare projects that have different sizes.

A natural solution to above and other problems is to compare only projects that come from the same part of the organisation, that come from a small episode of time in which no other changes took place, that had a similar percentage of infrastructural systems and that all have a similar size. However this would make the amount of projects that can be compared so small that no meaningful comparison can be made.

Making causal claims with respect to the observed efficiency changes, is difficult as organisations are no static entities. Clark [Clark, 2000] observed that in most organisations other changes take place during the run of a process improvement initiative. Other factors can also obscure the the causal link between process improvement and changed efficiency and productivity. Incremental implementation of process improvement in a large organisations makes it for example difficult to perform a simple pretest-posttest comparison design.

In most contexts where links between cause and effect are obscured by nuisance factors, experimentation can help to establish the causal links. Unfortunately, software process improvement is typically based on a best-practice model that contains many practices that together strengthen each others outcomes. This interlocking aspect of the practices in a best-practice model means that simple laboratory studies of single aspects of the development process are unlikely to provide realistic assessments of the contribution of a practice in the system of practices in a benchmark’s model.

What aggravates matters is that small development and maintenance tasks that would be suitable for experimentation are materially different from large projects. Organisations are therefore unable to base strong conclusions on field tests of best-practice models in small pilot projects. Because large projects have the biggest impact on the overall organ-
isational performance, best practices should work well in the context of large projects. The large projects are hard to simulate in experiments (because of the associated costs) and are hard to study in real life.

1.2 Research Context

Most of the studies described in this thesis (with the exception of the second study in Chaps. 5 and 6) have been carried out at the Information Technology department of the Dutch branch of the Consumer and Commercial Banking strategic business unit of the ABN AMRO Bank N.V. The Consumer and Commercial Banking unit provides retail banking services to individual customers and small and medium businesses.

At the time of research, 2002-2006, ABN AMRO Bank N.V. employed 110,000 staff members. Of these staff members 23,000 people worked in the Netherlands. The IT department consisted of 1500 staff members that were involved with the development and maintenance of information systems and 700 staff members that were involved with the operational maintenance and support of those information systems.

The IT development department primarily builds and maintains large, custom-built systems for transaction processing on mainframes, most of which are built in either COBOL and TELON (an application-generator for COBOL). Besides these mainframe systems, a large variety of other systems are implemented, constructed and maintained by the organization. These systems are implemented in a large variety of different programming languages (such as Java and COOL::Gen), run under various operating systems (such as Microsoft Windows and UNIX) and are distributed over different platforms (batch, block-based, GUI-based and browser-based).

The IT development department was organized as a matrix organisation, where staff members were organized in resource pools. Each resource pool was managed by a resource manager, that functioned as a hierarchical line manager for the staff members involved. Projects and programs were executed in IT domains that spanned either technological domains (such as the networking domain and the security domain) or application domains (such as the business contact database domain or the insurance domain). Each domain was managed by a manager, called the solution integrator and portfolio manager, who managed the project managers steering the individual projects. The domains were clustered into six larger units called columns, that were managed by a manager that was called the senior solution integrator.

In 2000 the organisation started with a major software process improvement program with the name “Inspiration” to improve the internal IT processes and cooperation with business parties involved. The name Inspiration is an acronym that stands for ‘Initiative for Software Process Improvement and Re-engineering of Abn amro’s Terminology, Implementation & Organisation’. The program finished successfully (as we shall see) in August 2004.

The reason for initiating the SPI program were the results of a series of two benchmark studies performed at the ABN AMRO Bank’s IT department [de Zwart, 2003]. Those benchmark studies (performed by Gartner and by Compass Management Consulting) indicated that application development and maintenance practices left ample room for productivity improvement.

The Inspiration program initially consisted of four initiatives:
the introduction of an internal metrics collection and benchmarking program.

• the introduction of a tailor-made quality system, the so-called integrated quality system (IQS) [Kleijnen, 2004]. The IQS consists of two levels that comply with the requirements of CMM [Paulk et al., 1993] level 2 and 3 respectively.

• the introduction of the Dynamic Systems Development Method (DSDM) [Stapleton, 2002] as the standard iterative development and project management method for the ABN AMRO Bank. DSDM is a rapid application development method that is suitable for incremental and iterative development, but the method can also be used for linear development projects.

• an initiative to professionalize project management through education of project managers and creating awareness of the importance of project management.

During the course of the Inspiration programme, the Inspiration programme evolved, stopping the professional project management initiative and instead focusing on a culture change program to support the other initiatives.

The Inspiration program started with the following goals [Kleijnen, 2000]:

• both the Development and Support functions have increased their productivity with at least 20%.

• A uniform development method is used throughout the Bank, accelerating Rapid Application Development (RAD) developments up to 50%.

• Quality of delivered products has increased, ‘development for maintenance’ has become reality.

• Metrics and measurement systems are in place, management is consciously using metrics.

• Co-operation between IT and Business in projects has significantly improved, user satisfaction is high, delivered solutions are perceived as high quality.

• Increased professionalism and maturity of the development and support staff, increasing job satisfaction and attractiveness to (new) staff.

During the execution of the Inspiration program the goal to increase productivity has received most of the management attention.

Before the introduction of DSDM as a standard project method, the organization used the method Method/1 Custom Systems Development, a proprietary method of Arthur Andersen [Arthur Andersen, 1988, Arthur Andersen, 1990]. The ABN AMRO Bank version of Method/1 had been tailored to meet the organization’s need of the time. However through the involvement of many external subcontractors, Method/1 lost its de facto standard status within the organisation. Method/1 was finally replaced to provide ABN AMRO Bank with an open and current development method that can be supported by multiple subcontractors.

During the improvement program, the organisation designed an on-line knowledge base (On-Track) containing reference information about the DSDM method and the supporting management and quality processes, together forming the integrated quality system. On-track not only contained process description, but also templates for documentation and review check-lists and background information about development tools. The
on-line knowledge base was a replacement of an existing on-line knowledge base (The Source), that was based on the prior (linear) development method Method/1.

To successfully implement the changes required for the software process improvement, a program with a dedicated senior vice-president was initiated. Through the SPI program the individual departments of the line organisation received assistance and coaching by dedicated internal and external consultants; IT staff received the required training and certification in CMM, DSDM and company-specific procedures; and improvement goals were set to maintain the commitment of upper management.

Besides the project organisation for the Inspiration program, a second organisational unit was introduced into the organisation to secure the new quality practices and system into the IT department. In each IT domain, a process improvement manager (PIM) was appointed to assist the domain manager (the solution integrator and portfolio manager) in making decisions about quality related issues. Each project was assigned a project quality assurance leader (PQAL) that assured that the quality procedures were followed or that the deviations from the quality system were recorded together with their rationale. The PQALs reported directly to the process improvement managers, instead of to the project leader. To complete the quality organisation, a software engineering process group (SEPG) was created to manage and maintain the quality system. The SEPG consisted of process improvement managers from different domains and of staff members of the Inspiration programs. Using process change requests, IT staff members could petition the SEPG to implement changes in the Integrated Quality System.

1.3 Research Questions

In Sect. 1.1 we have seen that organisations implementing a SPI program could benefit from measuring the effects that are achieved by their SPI program, as this makes it possible to keep the SPI program aligned with the organisation’s goals and keep executive sponsors committed to the SPI program.

In addition to tracking the effects of the SPI program, it would be beneficial for organisations to also use empirical evidence as a diagnostic instrument to determine which parts of the development and maintenance process cause problems or have a below average performance. This way the organisation’s process can be tailored to comply to the best practice model that is suggested by the SPI method and at the same time improve on weaknesses that are specific to the organisation.

The Ph.D. research project started as a study to collect data regarding the effects of software process improvement in general and the CMM model in specific. However during the analysis of the data it became clear that real problems in assessing the effects of SPI did not lie in the collection of empirical data, but rather in the interpretation of the data that has been collected. When requested by executive managers, IT staff diligently starts to collect data. However when it turns out that the collected data is collected for no apparent use, because it cannot be interpreted, the initial enthusiasm or at least cooperativeness fades away quickly. Therefore my research focus evolved from a research project that tried to assess the effects of SPI into a research program that addresses the following question:

- How can empirical data be used to strengthen a software process improvement initiative?
Empirical data can both be used to provide feedback about the current impact of a SPI program on the efficiency and effectiveness of SPI programs as well as to provide feedback on which IT processes still leave room for improvement. Therefore it is possible to split the original research question into the following two subquestions:

1. How can empirical data be used to get feedback about the effects of software process improvement?

2. How can empirical data be used to get feedback about the software development processes that still have room for improvement?

We have seen that when assessing the impact of software process improvement that one runs the risk of comparing apples with oranges. To escape these problems we can take two approaches: design comparison procedures that take the differences between projects into account (in other words to make apples and oranges comparable) or we can design new measurement devices that are sensitive to differences between the IT projects (in other words to make a separate scale for apples and a separate scale for oranges and make a sorting device to differentiate between the two of them.)

Therefore we can split the question “How can empirical data be used to get feedback about the effects of software process improvement?” again into the following two questions:

1.a How can data about software projects be made comparable?

1.b How can important aspects, for which no current metric exists, be quantified?

In addition to developing methodologies that help to evaluate software process improvement and that help to diagnose underperforming software development processes, we have also applied these methods in case studies at the ABN AMRO Bank. The outcomes of these case studies do not only provide an answer with respect to the efficacy the research method, but the case studies also provide insight into the effects of software process improvement. These results contribute to the answer of the following research question:

3. What are the quantitative effects of software process improvement on software development productivity?

1.4 Main Contributions

This thesis provides methods to analyze the impact of software process improvement and provides methods that can be used to identify processes that can be optimized.

To interpret data on the outcomes of software process improvement, we explain how statistical procedures can be used to compensate for certain differences between projects so that they can still be compared in a meaningful way. The use of regression analysis in conjunction with Box-Cox transformations [Box and Cox, 1964] can compensate for the productivity differences that are caused by differences in size (as a result of non-constant returns to scale in the IT domain). Hierarchical Linear Models [Lindley and Smith, 1972, Bryk and Raudenbush, 1992]) can be useful when an organisation consists of unit whose projects or data collection procedures are so different that the data of
Introduction

different unit cannot be compared in a meaningful way. When trying to compare projects that have been executed under different project life cycles (e.g. projects run according to Method/1 vs. projects run according to the DSDM) our suggestion of creating a life-cycle map before comparing the projects can be a useful suggestion.

Sometimes one cannot escape the fact that without additional information projects cannot be compared at all. We have identified one such factor (the amount of installation and configuration tasks that a project has to perform to get the required infrastructure up and running) and created a metric for this aspect (i.e. infrastructure effort points).

When we want to identify improvement points in the software engineering process that were previously unknown, one often also has to take new data into account. If sufficient data had been available to single out the improvement opportunities, the improvement actions would probably already have taken place. We therefore have to turn to new sources of data that can be obtained and analysed to gain new insights. We identified the project post-mortem evaluation repository of an organisation as such an important source of empirical data. For such a repository we devised a novel analysis method that can be used to extract useful information out of such an evaluation repository.

Having deconstructed the empirical investigation process into a process that tries to establish the exact impact of certain changes to the software development process and into a process that identifies improvement areas, we are now stuck with two apparently conflicting investigative tasks. To create synergy between the two empirical research tasks we show a research framework that can integrate the two tasks in one overall empirical investigative process.

In addition to these empirical research techniques we also analysed the actual software process improvement program of the ABN AMRO Bank. In a case study to examine the impact of the improved quality practices on the productivity, we analysed data of 410 projects and found a productivity improvement of 20%.

1.5 Structure of this Thesis

In Chap. 2 an overview of relevant background literature with respect to software process improvement is given.

Chapter 3 describes which statistical techniques are suitable to discern changes in overall software development productivity after the completion of a process improvement program. This chapter not only describes the required statistical techniques to detect productivity changes (such as Hierarchical Linear Models, Box-Cox transformations and Power Estimation simulations), yet it also describes the actual productivity improvements observed in the case of the ABN AMRO Bank.

Productivity changes are not only studied at the holistic, overall software development process, but also in one of the underlying subprocesses that together form the development process. In Chap. 4 we analyse the new requirements engineering process of the DSDM method, that involved facilitated workshops. This chapter also shows that for the investigation of subprocesses it can be necessary to map the practices or phases of one development method to the practices and phases of another method, before the subprocesses can be compared.

Chapter 5 explains how grounded theory can be used to analyse the content of written documents, such as project post-mortem documents. With this analysis one can recover
information about projects that was not specifically captured during the course of those projects. One could for example discover that the use of test tooling has a significant impact on productivity, even though no formal evaluation of test tooling has taken place. Grounded theory-based analysis is used in three phases: first a typology of factors that has impact on the projects is constructed, then the documented projects are classified according to the constructed typology, finally a statistical analysis is executed to discern relations between factors from the identified typology and the performance of those projects. The method was been applied to two organisations, the IT department of the ABN AMRO Bank and Kongsberg Spacetec, a Norwegian producer of receiving stations for data from meteorological and Earth observation satellites. The results are verified in Chap. 6 using a triangulation approach. The method has successful within the context of ABN AMRO Bank, but has delivered mixed results in the context of Kongsberg Spacetec.

In most information system development projects not every component is built from scratch, instead IT professionals use commercially available software components to provide standard functionality to the end users (such as word processing tools and spreadsheet programs) and to the developer (such as database systems). With the increasing reliance on these commercially available components, the installation and configuration costs can no longer be ignored. In Chap. 7, a metric to measure the size of the installation task of infrastructural components is proposed. With this size metric, the impact of software process improvement on the configuration and installation of infrastructural components can be measured.

Most empirical research guides describe research as a linear process. One starts with a clear research question, then designs an experiment, after which the experiment is executed and data collected is, finally the analysis of this data leads to the desired insights. Although these research guidelines provide structure to a researcher, the linear approach is often not suitable for process improvement. An implicit assumption of the research process guides is that the research question is sufficiently clear at the start of an investigation. This assumption fails in software developing organisations that wish to improve their performance based on sound empirical data. Organisations typically receive vague or contradicting indicators of problems that need to be solved. Chapter 8 explains an iterative research cycle that is useful for organisations that wish to improve their performance through empirical research. The iterative research framework assumes that the organisation continuously collect information using broad indicators that give an indication of the performance of their core processes. This continuous information gathering and analysis will lead to indications of problems and will identify opportunities for improvement. These indicators will typically lack statistical significance and therefore a second research cycle is started to confirm the validity of the indicators or reject them as false positives. When a problem is confirmed, subsequent improvement actions can be started.

This thesis concludes with some personal reflections on how I learned to perform empirical software engineering research in Chap. 9 and in Chap. 10 some concluding remarks and indications for further research are identified.
1.6 Acknowledgements

Without the support of many organisations and individuals it would not have been possible for me to perform the research for my Ph.D. thesis. I would like to acknowledge the support of:

- ABN AMRO Bank N.V for funding the first two years of my research under the research grant “Reflection on Inspiration” and providing me with access to their organisation, to study the process improvement program Inspiration.

- Vrije Universiteit Amsterdam for funding the last two years of my Ph.D. studies.

- Senter, the funding agency of the Dutch Ministry of Economic Affairs, for sponsoring the CALCE project (nr. TSIT3018).

- Capgemini for their cooperation and providing me with training on the Dynamic Systems Development Methodology.

I would also like to thank the cooperation of the following institutions:

- Members of the Dynamic Systems Development Methodology implementation team at ABN AMRO Bank for acting as a group of colleagues and providing me with information on the implementation of DSDM (Philippe Castermans, Perry Haandrikman, Barbara de Jong, Fred van der Linden, Geert Mooi, Flip Niesten, Wanda Smit-Horst, Liduine Thorn, Cles van de Ven, Carolien van Zuylen, Aad de Winter, Mischa van den Brand, Leo Diepstraten, Frank Harmsen, Yvonne Janssen, Onno van der Klei, and Raimond Wets).

- Members of the Capability Maturity Model implementation team at ABN AMRO Bank for providing me with information on the implementation of the CMM (Raymond Wortel, Peter Brouwer, Ron Cordemeijer, Eddy Cornelisse, Jasper Doornbos, Leon van der Giessen, Cok Koren, Roel Koblens, Han Noë, Wil Spronck and Mien Vermue).

- Member of the project office IT of the ABN AMRO Bank for providing me with explanation on the project administration and introducing me to AIMS system (Ton Groen, Tecla Buskermolen, Hetty de Leeuw, Robert Louwers, Nico Mak, John Nijholt, Cosmo Ombre, Peter de Ruijter, and Phil Willebrands).

- My supervisors and liaisons at the ABN AMRO Bank, first Jean Kleijnen and later Marcel Uleman.

- SINTEF ICT and the Norwegian University of Science and Technology (NTNU) for inviting me to Trondheim to cooperate on research on post-mortem analysis methods as well as Torgeir Dingsøyr, Nils Brede and Tor Stålhane for the nice cooperating with me during my stay in Norway.

- Geurt Jongbloed, professor at the Department of Mathematical Statistics at the Delft University of Technology, for all his advice and patient guidance on the topic of statistics and its application.

- Verdonck, Klooster and Associates for granting me some time to finish the last part of my Ph.D. thesis.
1.7 Publications

Most of the work that is presented in this Ph.D. thesis has been published previously. This section provides an overview of those publications. This information can also be found at the beginning of each chapter.

The empirical results of the study into the effects of process maturity on productivity in Chap. 3 has previously been presented at the 7th International Conference on Product Focused Software Process Improvement (PROFES 2006) [Schalken et al., 2006b]. The methodological issues in Chap. 3 have been described in a manuscript that has been submitted for publication [Schalken et al., 2007].

The study into the effects of gathering requirements using facilitated workshops in Chap. 4 has been previously presented at the 8th Conference on Evaluation & Assessment in Software Engineering (EASE-2004) [Schalken et al., 2004].

The case studies of the project post-mortem analyses, described in Chaps. 5 and 6, have been also been published previously. The ABN AMRO Bank case has been published at the 11th European Software Process Improvement Conference (EuroSPI 2004) [Schalken et al., 2004], the Kongsberg Spacetec case has been accepted for publication at 14th European Software Process Improvement Conference (EuroSPI 2007) [Dingsøyr et al., 2007]. The method to analyse project post-mortem analyses, described in Chap. 5, has briefly been described during the presentation at the 11th European Software Process Improvement Conference (EuroSPI 2004) and was published in more detail in the Software Process: Improvement and Practice journal [Schalken et al., 2006a].

The method used to measure the size of infrastructural IT projects in Chap. 7 has been presented at the 17th Conference on Advanced Information Systems Engineering (CAiSE 2005) [Schalken et al., 2005].

The chapter on incremental research strategies for empirical investigations into SPI, Chap. 8, has been based on work presented at on a paper presented at the 12th Doctoral Consortium of the Conference on Advanced Information Systems Engineering (DC CAiSE 2005) [Schalken et al., 2005] and on a manuscript that has been accepted for publication by The Journal of Systems and Software [Schalken and van Vliet, 2007].

Chapter 9, which contains some lessons learnt during the research project and some general observations, has previously been presented at the 4th International Workshop on Empirical Software Engineering (WSESE-2006) [Schalken, 2006].
Chapter 2

Software Process Improvement and Process Assessment

As we have noted in Chap. 1 there has been dissatisfaction with Information Technology (IT) performance over the past fifty years [Gibbs, 1994, Jones, 2000]. In the beginning computers were only used as number crunchers in research facilities of universities and military laboratories. However with the ever decreasing costs of computers and the relentless increase in capabilities of computers that follows Moore’s Law [Mollick, 2006], the computer has taken over the world. Now society increasingly relies on IT, the troublesome performance of IT is having an increasing impact on the performance of corporations and the government. The troublesome performance of IT is effectively having an negative impact on the lives of everybody.

Most IT related problems are problems that have to do with the software that runs on the computers, not with the reliability of the computers themselves. The problems encountered during the construction of software are more difficult, because they lie both on the technical domain (computers) as on the application domain (e.g. the business of banking) and therefore, according to Brooks, “we see no silver bullet. There is no single development, in either technology or in management technique, that by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity.” [Brooks, 1987].

Although we cannot expect to find a silver bullet, improvement of the development, maintenance and support of software is nonetheless needed. The increased reliance on Information Systems and IT of our society mean that more and more systems need to be developed and subsequently maintained. The growth (both in size and in number) of IT systems in turn leads to ever increasing required effort costs to develop and maintain those IT systems. The problem of increasing IT labour costs is aggravated by the lack of skilled IT professionals in Western Europe. For the foreseeable future, there will come no decreased demand on IT services.

In his book Rapid Development [McConnell, 1996b], McConnell distinguishes four dimensions that are of influence on the efficiency of time-constrained projects. These dimensions are: people, product, technology, and process (c.f. Fig. 2.1). It has been long known that not every IT professional is equally proficient. In 1968 Schwartz observed that “within a group of programmers, there may be an order of magnitude difference in
Figure 2.1: The four dimensions of development speed [McConnell, 1996b].

capability” [Schwartz, 1968] and Boehm [Boehm, 1975] observed that productivity variations of 5 to 1 were not uncommon. The second dimension, product, is concerned with the selection of features and quality attributes that will include and the (sometimes implicit) selection of features that will be excluded from the product. Reducing product size, by excluding features that are not needed, decreases the required effort for development now and subsequent maintenance of those features in the future. An accepted technique to select features to include in the software are the so-called MoSCoW-rules [Stapleton, 2002, Stapleton, 2003], that specify which functionality Must be present, the functionality that Should be present, the functionality that Could be present and (perhaps most important) the functionality that Won’t be present.

The two dimensions discussed so far (people and product) are characteristics that are under the control of a single project or organisation and are often very situation and problem specific. The other two dimensions (technology and process) are generic dimensions, that are applicable to a broad suite of projects and organisations. The technology dimension refers to the tools that are used by the IT professionals to support the capturing of requirements, the design of the system, to program the system (and translate it to machine language) and subsequently test the system and to make the system ready for production in a live environment. The proper use of tooling can indeed make a professional much more productive (c.f. the impact of programming languages on the size of a computer program [Capers, 1995]). However one should critically assess the claims made by vendors, who are usually over-enthusiastic and ignore the shortcomings of their products. Still, for the proper job and with the proper training and support, many tools do have a positive impact on the output of IT staff. The last dimension, process, relates to the activities and tasks performed by the IT staff and their management. Ever since the boom of quality management, it has been known that focus on the development process and not just on the outcome of process (the product or service) can drastically improve
2.1. Software Process Improvement

As one of the four dimensions of development efficiency, management and optimisation of software life cycle processes leads to new or better practices of individuals, projects, and organizations [Abran and Moore, 2004, p.9-1]. The optimisation of software development and maintenance processes is called software process improvement. Software processes improvement approaches can be categorised in one of the following three global categories:

- software development and maintenance methods: development and maintenance methods offer the most concrete guidance for IT professionals and describe step by step which activities should be performed and how the end results should look like. On can view methods as recipes to be followed by the IT staff. Examples of development methods are the Dynamic Systems Development Method (DSDM) [Stapleton, 2002], eXtreme Programming (XP) [Beck, 1999] and the Rational Unified Process (RUP) [Jacobsen et al., 1994]. One could say that methods try to answer the question ‘how?’.

- more freedom is granted by process frameworks. Card [Card, 1991] calls this class of approaches benchmark approaches and van Solingen and Berghout call these approaches top-down approaches. Process frameworks are based on commonalities between the characteristics of best-of-class IT companies. Process frameworks describe what process goals an organisation should fulfill and describe recommended practices to reach those goals. By fulfilling the goals of a framework one avoids typical pitfalls. Process frameworks do not prescribe detailed work processes and focus on the ‘what?’ question instead of on the ‘how?’ question. The best known example is the Capability Maturity Model.

- the most liberal are the process improvement processes. Process improvement approaches do not provide specific guide on what goals an organisation should pursue or what activities it should perform. Instead it provides a general approach to analyse current activities and how to improve those activities. These approaches therefore focus on the ‘why?’ question. The best known process improvement process is called the Quality Improvement Paradigm (QIP) [Basili and Green, 1994].

When we go from methods to process improvement processes, the organisations gets more latitude to chose among the possible solutions to implement an improvement. With the increased latitude, not only the freedom increases, but also the amount of guidance decreases. Some authors believe that open ended improvement approaches are more suitable for mature organisations whereas close ended improvement approaches are more suitable for immature organisations.

The categories in the above list are not fully mutually exclusive. E.g. some process frameworks, such as the Capability Maturity Model, include process improvement processes as characterising for mature organisations.
2.1.1 Software Development Methods

Software engineering methods or software development methods “impose structure on the software engineering activity with the goal of making the activity systematic and ultimately more likely to be successful. Methods usually provide a notation and vocabulary, procedures for performing identifiable tasks, and guidelines for checking both the process and the product” [Abran and Moore, 2004]. A software development method can be seen as a recipe to develop software products.

The development methods can be subdivided into three categories [Abran and Moore, 2004, p. 10-1]:

- **Heuristic methods** that focus on a linear set of activities that progress from requirement elicitation, design, build, test to implementation. Structured, data-oriented, object-oriented, and domain-specific development methods are all examples of heuristic methods because they all capture design decisions in a more or less non-formal way.

- **Formal methods** use mathematical descriptions to define the required functionality of a piece of software. When the specifications (which are far more detailed than the designs created in the heuristic methods) are completed, one can use techniques from mathematics and theoretical computer science to further refine the specification (into first a design and then code) into a working computer program. Sometimes the transformation method ensures that the system conforms to the specifications. In other cases one can instead prove the correctness of a program. Applying formal methods can be expensive, but can provide greater reliability and correctness than can be guaranteed by other development methods (that use inspections and testing to ensure reliability and correctness).

- The last category of methods are called the **prototype methods**, which use prototypes of the system to clarify functional requirements and subsequently use prototypes to evolve the design of a system. Instead of focusing on upfront work, prototype methods focus on the interaction with the customer and experimenting with technology. Agile methods fall under the category of prototyping methods.

Just as there exists no “best” software process improvement approach, there is also no “best” development method. A development method needs to be suitable for a certain problem and organisational context at hand.

2.1.2 Process Frameworks

Instead of providing a group of IT professionals with a set of step-by-step guidelines, one can also give them a set of objectives that the process is supposed to fulfil. These objectives are derived by comparing the practices of companies that have a state-of-the-art development process. The recurring practices of these exemplar companies are analysed to derive their goals and underlying principles.

In a process framework these goals and principles are often arranged in levels. The idea about these levels is hat in order to be able to execute more advanced practices, one first needs to have certain basic practices and capabilities in place. It would for example not help to have a practice in place to formally link design artifacts to requirements (so
called traceability) if there is no stable set of requirements (in other words there is no requirements management in place). The idea of maturity stages in IT has been invented by Nolan [Nolan, 1973].

When we compare methods with process frameworks, one sees that process frameworks contain far less detailed prescriptions but perhaps a broader area of processes is covered. In process frameworks it does not matter how a practice is implemented, as long as the goals and underlying principles of a practice have been implemented.

Examples of process frameworks are the Capability Maturity Model (CMM) [Paulk et al., 1993], ISO-SPICE [ISO/IEC, 1998], and the Capability Maturity Model Integrated (CMMI) [CMMI Product Development Team, 2000].

2.1.3 Process Improvement Approaches

The previous two approaches are top-down approaches to software process improvements. Bottom-up improvement approaches, such as the Experience Factory [Basili et al., 1994b], suggest that feedback of individual projects should be packaged, in Experience Packages, to support the reuse of lessons learned by the entire organisation [Seaman, 1993]. In the packaging of experience information is consolidated so it can be quickly understood and used throughout the rest of the organisation. The essential difference between top-down and bottom-up processes is that with bottom-up processes one does not take exemplar processes as a norm (although nothing forbids the use of some or all of the components of a process framework) but instead takes experience of the organisation (what works and what does not work?).

The collection of metrics about the products and process of an IT organisation is seen as crucial for an organisation’s success. Measurement is either seen as a distinguishing feature of mature practices or as driving force for improvement itself. In the Capability Maturity Model [Paulk et al., 1995] measurement ensures the control and improvement of any practice, and in the Quality Improvement Paradigm [Basili et al., 1994a] measurement drives organisational change.

2.2 Process Improvement Approaches at ABN AMRO Bank

In this section we will focus on two specific improvement approaches that have been used at the ABN AMRO Bank. One is a development method (DSDM), the other is a process framework (CMM). In our case studies we will examine the effects of introducing DSDM and CMM into the organisation.

2.2.1 Capability Maturity Model

In the 1970-ties IBM was experiencing that because of the increased complexity of their software, their software systems became so difficult to develop and maintain that their software quality decreased below what IBM managers and IBM customers deemed acceptable. On the initiative of their CEO (Thoma J. Watson), IBM staff members started developing the first software process improvement process framework [Radice et al., 1985b, Radice et al., 1985a].
When the Department of Defence of the United States government started to experience similar problems but now with IT subcontractors, they requested Watts Humphrey and the Carnegie Mellon University to come up with a solution. Based on experiences at IBM, Watts Humphrey defined a five level maturity model that could be used to assess the software processes maturity of a subcontractor [Humphrey, 1989]. The assessment tool could help guard the Department of Defence from selecting an IT provider that was cheap but did not have the expertise to complete the project successfully.

This maturity model evolved into the Carnegie Mellon University Software Engineering Institute’s Capability Maturity Model [Paulk et al., 1993, Paulk et al., 1995]. Initially the model gained quick acceptance within the defense contracting agencies, as sufficient process maturity was a requisite to be allowed to participate in large governmentally projects. When the CMM gained adopters in the defense contracting sector, it became so popular that also other companies started to use the model for their software process improvement program.

The CMM model consists of five maturity levels [Paulk et al., 1993, pp. 8–9]:

**Initial** The software process is characterized as ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort.

**Repeatable** Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

**Defined** The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization’s standard software process for developing and maintaining software.

![Figure 2.2: The levels within the Capability Maturity Model [Paulk et al., 1995, p. 16]](image-url)
2.2. Process Improvement Approaches at ABN AMRO Bank

**Managed** Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled.

**Optimizing** Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.

Each of these levels, except for the Initial level, consists of a number of key process areas (KPAs), that describe a part of the software engineering process. Each KPA is described in five aspects:

**Commitment to Perform** Commitment to Perform describes required actions necessary to ensure the process is established and maintained. This aspect has to do with senior management support.

**Ability to Perform** Ability to Perform describes the necessary conditions that must be present in a project or organization to be able to implement the software process competently. Ability to Perform involves issues such as training, resources, and organizational structures.

**Activities Performed** Activities Performed describe the organisational structure (both roles and procedures) that are needed to implement a key process area.

**Measurement and Analysis** Measurement and Analysis describes the need to measure the process and analyze the measurements.

**Verifying Implementation** Verifying Implementation describes the safeguards to ensure that the process is performed in accordance to the process that has been established. These safeguards include the quality assurance roles in an organisation.

The following list presents the key process areas of CMM level 2 (the defined level) to give the reader an idea what practice areas are included in the Capability Maturity Model.

- Requirements Management
- Software Project Planning
- Software Project Tracking and Oversight
- Software Subcontract Management
- Software Quality Assurance
- Software Configuration Management

At the moment there is little empirical data available on CMM-based software process improvement, especially outside the governmental contractor domain. Galin and Avrahami [Galin and Avrahami, 2005] were only able to identify 19 empirical studies of CMM. And even those 19 studies did not discuss all factors (product quality, time-to-market, productivity and schedule risk), in their results governmental projects were over represented.
Although the results of the studies all point in the same direction, 19 studies is rather limited if one sees the current acceptance of SPI methods. This might partially have to do with the problem of collection meaningful and correct empirical data on software projects, but it may also have to do with the difficulty of demonstrating the effectiveness of a set of interrelated practices. In prior studies of maturity level theories (e.g. Nolan’s Stage hypothesis [Nolan, 1973]) there has been significant debate whether the level models are valid or not [El Emam and Goldenson, 1999]. Being of a similar family of models, the CMM model has also received its share of criticism.

2.2.2 Dynamic Systems Development Method

Not only did the ABN AMRO Bank introduce a new Software Process framework as a basis for their quality system, simultaneously the improvement program also standardized the software development life-cycle model to the Dynamic Systems Development Method [Stapleton, 2002, Stapleton, 2003], version four of the DSDM is also known as the Framework For Business Centred Development.

The DSDM provides a life-cycle model for the execution of (initially IT) projects, but has also been used to provide a life-cycle for other types of projects. Instead of providing step-by-step instructions, the DSDM describes techniques (such as facilitated workshops and prototyping) that can be used at specific moment during the life-cycle of a project.

The DSDM was created as a response to the increasing diversity of rapid application development (RAD) methods around 1993. An industrial consortium, the DSDM consortium, and especially its Technical Work-group, drafted the first version of the Dynamic Systems Development Method.

The DSDM method tries, as any RAD method, to start with the most important pieces of the software, extending that software during the project.

A DSDM project has the following phases [Stapleton, 2002]:

**Feasibility Study** During the Feasibility Study project members assess the suitability of DSDM as an approach for the project as well as the regular considerations of a feasibility study, such as a definition of the problem, technical feasibility of the problem and the predicted costs.

**Business Study** When it has been decided that the project is feasible and DSDM is an acceptable method for the project, the Business Study provides the basis for all the work to follow; affected business processes are identified as well as their information needs. During this phase also the relevant stakeholders are identified, an architecture for the system is defined and all activities are planned and management controls are installed.

**Functional Model Iteration** The Functional Model Iteration refines on refines the business requirements of the computer system using prioritized requirements lists and models of the processes.

Both the Functional Model Iteration and the Design and Build Iteration consist of cycles of four activities: identify what is to be produced, agree how and when to do it, create the product, verify that it has been produced correctly.

**Design and Build Iteration** Although most functionality has been identified during the Functional Model Iteration and built in functional prototypes, not all the testing
will have been performed nor will all the non-functional requirements have been implemented. That is the main task for the Design and Build Iteration.

**Implementation** The Implementation phase covers the cut-over from the development environment to the operational environment. This includes training the users who have not been part of the project team. Iteration of the Implementation phase is applicable when the system is being delivered to a dispersed user population over a period of time.

If, in order to save time or reduce risk, not all functionality has been implemented or not all non-functional requirement have been implemented, one can track back to the Functional Model or Design and Build Iteration to finish the work.

The technique and life-cycle of the DSDM method have been based on the assumption that the following principles are true for the project (as they usually are) [Stapleton, 2002]:

1. Active user involvement is imperative
2. DSDM teams must be empowered to make decisions
3. The focus is on frequent delivery of products
4. Fitness for business purpose is the essential criterion for acceptance of deliverables
5. Iterative and incremental development is necessary to converge on an accurate business solution

![Figure 2.3: Process overview of the Dynamic Systems Development Method](Stapleton, 2003, p. 4)
6. All changes during development are reversible
7. Requirements are baselined at a high level
8. Testing is integrated throughout the lifecycle
9. A collaborative and co-operative approach between all stakeholders is essential

2.3 Process Assessment

As we have seen in this chapter, software process improvement methods are both complex and consist of multiple interlocking practices. It is not immediately and evidently clear if a process improvement will be useful for the organisation that considers it. Debates in popular trade magazine as to which method is best usually lead to heated debates that are based more on rhetorics than on facts.

The problem is that for complex methods as software process improvement, even after one has experimented with the method in real projects it is not always clear for participants if the project was really more effective.

Because direct observations (of participants of SPI trials) and logical argumentation (in trade press) is unable to come to a decisive answer, one needs to revert to empirical methods.

Using either benchmarks (with comparisons of oneself against industry peers) or preferable using benchmarks (to compare the state of affairs before and after the SPI trials) one can gain insight into the actual benefits of SPI.

2.3.1 Software Measurement and Metrics

Some aspects of measuring the effects of SPI programs are not new. Many issues regarding the empirical validation have previously been addressed in literature regarding the implementation of successful metrics programs. The issues of metric programs (and therefore of evaluations of SPI programs) can be categorised into four classes [Jeffery and Berry, 1993]:

1. context, or overall environment of measurement effort,
2. resources to support the measurement activities,
3. process of the measurement activity, and
4. products of the measurement activity.

As most of issues 1, 2 are mostly generic for all evidence collection programs, and often outside the influence of an SPI team, we will therefore focus our attention on categories 3 and 4: the process of the measurement activity, and products of the measurement activity. As for these categories one can expect to find issues that can be influenced directly by the SPI staff and that might require solutions that are different from typical metric programs.
Chapter 3

Effects of SPI on the Productivity of Software Development

In this chapter we publish the results of a thorough empirical evaluation of a CMM-based software process improvement program that took place at the IT department of the ABN AMRO Bank. Data of 410 projects collected over a period of four years are analysed and a productivity improvement of about 20% is found.

In addition to these results we explain why the current method of analysing productivity data on software process improvement programs lacks in statistical power. The impact of a software process improvement program is most often measured using productivity indices. Analyses based on these indices often lack in statistical power. This lack in statistical power causes the analyses to miss changes in productivity, even when they are present. Instead of productivity indices, we use linear regression models and hierarchical linear models, because they possess higher statistical power to detect changes in productivity. In this chapter we demonstrate the use of linear regression models and hierarchical linear models on the data collected during the process improvement program. When analysing the statistical power of the methods using a simulation, we found that the linear regression models and hierarchical linear models possess superior sensitivity in detecting changes in productivity.

This chapter is based on a paper presented earlier at the 7th International Conference on Product Focused Software Process Improvement (PROFES 2006) [Schalken et al., 2006b] and a manuscript that has been submitted for publication [Schalken et al., 2007].

3.1 Introduction

Improvement models have been proposed by the software process improvement field to improve the management and work processes in software development. The benefits of an improvement model are not always intuitive, as each of these models focus on a wide range of interacting practices. Therefore empirical evidence is needed that demonstrates the usefulness of the improvement models.

Today there is still a dire lack of published empirical evidence on the effects of SPI in industry. Even for the Capability Maturity Model [Paulk et al., 1993], which is the
most widely used improvement model, little empirical data is available. In a recent meta
analysis on the effects of the CMM, Galin and Avrahami [Galin and Avrahami, 2005]
were only able to identify three studies that give details on productivity gains when an
organisation progresses to CMM level 2 and only twelve studies that provide details
on productivity gains when an organisation progresses to CMM level 3. The lack of
empirical evidence is odd, because improvement models have been available for a relative
long period of time and the need for empirical validation is clear.

In this study we investigate the success of a software process improvement program at
ABN AMRO Bank. In this program the Capability Maturity Model has been used as the
reference model for software process improvement and the Dynamic Systems Develop-
ment Method (DSDM) [Stapleton, 2002] as the new project management methodology.
We analysed the productivity of 410 projects during a period of four years and found a
productivity increase of 20%.

3.1.1 Ineffective Productivity Data Analysis

It is sometimes assumed that the lack of evidence on SPI programs is caused by the inef-
ficiveness of extensive measurement programs in immature organisations. We however
believe that in immature organisations the effects of SPI could just as well exist yet re-
main invisible because the data is too noisy to be analysed with statistical techniques
currently in use.

It is well known that statistical techniques are not always used appropriately in the
applied sciences. Empirical software engineering, a young discipline, is no exception
to this rule [Kitchenham et al., 2002]. It has been documented that the inappropriate
application of statistics has led software engineering researchers to believe that relations
exist, where in practice they do not exist (c.f. Rosenberg [Rosenberg, 1997]). But that
same misuse of statistics might very well have prevented other researchers from finding
relations that do exist. Cases where effects of treatments have been overlooked are not
frequently documented, because publication bias exists towards positive results [Pickard
et al., 1998].

To assist empirical software engineering researchers in the correct application and
reporting of statistics preliminary recommendations have been made [Kitchenham et al.,
2002]. Unfortunately, these recommendations remain abstract such as “Ensure that the
data do not violate the assumptions of the tests used on them” [Kitchenham et al., 2002].

Empirical studies that investigate the effects of software process improvement typi-
cally spend little or no attention to the choice of a statistical technique that is appropriate
to identify changes in productivity. Without further deliberation, these studies compute
productivity as size divided by effort. This quotient of size and effort is also referred to
as the productivity index.

For software projects, the quotient of size and effort will typically exhibit a large
variance. The large differences in project productivity, caused by the differences be-
tween these projects (of which size is the most important), are the cause of this large
variance. Given the same amount of data, the chance that a statistical test (such as the
t-test [Bhattacharyya and Johnson, 1977, p. 293]) detects an improvement is proportional
to the ratio of improvement in productivity divided by the overall variance in produc-
tivity measurement [Cohen, 1988]. Existence of large variance is therefore problematic,
because the chance that a statistical test detects an improvement decreases when variance
3.1. Introduction

Intuitively one might try to decrease the variance by including size as an additional parameter in the comparison. This is however not valid as productivity is also a composite measure of effort and size, because one then includes size in both sides of the equation. Previously, Rosenberg [Rosenberg, 1997] has explained why modelling a compound measure against one of its components can easily lead to meaningless conclusions. Another example of this mistake can be found in [El Emam et al., 2002].

As a rule, organisations have a limited amount of empirical data at their disposal to assess the effects of SPI programs. We also know that we can only expect moderate improvements in productivity from IT innovations [Brooks, 1987]. If we assume that we need to keep the required significance level constant and if we assume that we cannot influence the number of observations or the size of the improvement, our only hope lies in using statistical techniques that are economical with the data that is available. A promising approach to use the available data more economically is to decrease the variance in our productivity measurement procedure.

Most studies perform an organisation-wide analysis of their process improvement program. In larger organisations, the divisions and departments will typically use different technologies and will work on different products. We will call organisations with divisions and departments that are dissimilar heterogeneous organisations. In heterogeneous organisations, empirical data from different departments should not be combined when standard statistical methods are used, as one risks comparing chalk and cheese.

3.1.2 Research Questions

In the previous paragraphs we have shown the need for empirical validation of software process improvement methods and we explained how the inefficient application of statistical techniques can cause researchers to miss productivity improvements gained through software process improvement.

The above stated problems can be solved by providing an answer to the following three research questions:

- What is the impact of a project’s CMM level on its productivity?
- Which alternatives exist to measure changes in productivity caused by software process improvement?
- Which alternative to measure changes in productivity caused by software process improvement, is most efficient with the available data?

3.1.3 Contribution

In this chapter we apply traditional productivity index measurements as well as improved statistical techniques to investigate the success of a software process improvement program at the ABN AMRO Bank. In this program the Capability Maturity Model has been used as the reference model for software process improvement and the Dynamic Systems Development Method (DSDM) [Stapleton, 2002] as the new project management methodology. We analysed the productivity of 410 projects during a period of four years and found a productivity increase of 20%.
We explain how linear regression models [Peter et al., 1996] can be used as better alternatives to productivity indices. Linear regression models can take variable returns to scale into account. The elimination of the interference caused by variable returns to scale promises a decrease in variance of the productivity measure. This means that these methods are more likely to detect changed productivity when limited data is available.

Heterogeneous organisations, that produce software for various markets or using different technology, should not combine their productivity data when standard statistical methods are used. Hierarchical linear models [Bryk and Raudenbush, 1992] prove to be useful in such situations, as they allow data to be combined without jeopardising the validity of the investigation. This additional flexibility comes at the cost of having to estimate more parameters, which in turn will increase the required amount of data. Whether the benefits outweigh the costs needs to be determined in a case by case basis.

The use of the proposed statistical techniques resulted in a significant increase in the amount of explained variance: productivity indices explain a mere 2% of the variance in the case study data, whereas hierarchical linear models are able to explain 60% of the variance. A simulation, with data from the case study, uncovered that these techniques not only improve the explained variance, but that they also greatly improve the chance of detecting changes in productivity when the available data would have been smaller.

3.1.4 Outline

The remainder of the chapter is organised as follows: Section 3.2 discusses related work. Section 3.3 explains the research method used to choose between different statistical techniques. Section 3.4 presents an overview of statistical techniques to analyse productivity data. Section 3.5 presents the results of applying the statistical techniques on the data of an SPI case study. Finally in Section 3.6 the results of the study are discussed.

3.2 Related Work

Related work for this study consists of two bodies of literature: studies of the empirical results of software process improvement and literature of statistical techniques used in empirical software engineering.

In [Diaz and King, 2002, Diaz and Sligo, 1997, Wohlwend and Rosenbaum, 1993, Oldham et al., 1999] studies of software process improvement are described. The studies report the (solely positive) changes in productivity of software developing organisations, expressed in ratios of lines of code delivered per unit of effort (usually man months).

In their paper “Do SQA Programs Work” [Galin and Avrahami, 2005] Galin and Avrahami provide an overview of the above mentioned case studies and others that contain empirical evidence on the effects of software process improvement. They identified 22 studies relating to the effects of Capability Maturity Model based software process improvement. Of these 22 studies, only 19 contained sufficiently detailed quantitative data to allow a meta analysis of the SPI effects. The meta analysis examines the effects of CMM on error density, productivity, rework schedule time, conformance to schedule and the effectiveness of error detection.

The authors were able to locate three studies that examine the change in productivity when a software development organisation increases its maturity from CMM level
1 to CMM level 2 and ten studies that examine the change in productivity when an organisation increases its maturity from CMM level 2 to CMM level 3. On average an organisation increases its productivity by 42.3% when it matures to CMM level 2. And an organisation that increases its maturity to CMM level 3 improves its productivity by an additional 44.4%.

In the field of software process improvement little attention has been paid to the mechanics of analysing productivity data. Virtually every empirical study of the effects of software process improvement analyses productivity using productivity indices. Productivity indices express productivity as a ratio of effort (to develop the software) and size (of the delivered software) [Conte et al., 1986, chap. 5]. Unfortunately productivity indices are not efficient in analysing productivity data, as we will demonstrate in this chapter.

In the field of software cost-estimation (c.f. Fenton en Pfleeger [Fenton and Pfleeger, 1998, chap. 12]) the relationship between effort and size of the delivered software has been investigated more thoroughly. This relationship between effort, size and other parameters is modelled using economic production functions. Unfortunately the statistical techniques proposed by the cost-estimation field have not been adopted by software process improvement researchers to elucidate changes in productivity caused by software process improvement.

Software cost-estimation research has investigated the parameters influencing the required effort, such as size, as well as the functional form of this relationship, the so-called production function. Popular production functions to model the relationship between effort and size are the linear model [Nelson, 1966, Wolverton, 1974], the quadratic model, the Cobb-Douglas or log-linear model [Cobb and Douglas, 1928, Walston and Felix, 1977, Boehm, 1981] (which is based on the concept of economy of scale [Banker and Kemerer, 1989]), and the trans-log model [Christensen et al., 1971]. Different approaches have been proposed to choose which model is most appropriate to model software effort [Kitchenham, 1992, Banker et al., 1994a, Hu, 1997, Briand et al., 1999], yet there seems to be no consensus on which functional form is most appropriate to model effort as a function of size.

Related work on hierarchical linear models applied to software process improvement has not been found.

Henry [Henry et al., 1995] investigated the sensitivity of statistical techniques to detect improvements caused by SPI programs. For instance, Henry manipulated project data to artificially improve the quality of initial effort estimations. After this manipulation Henry determined how large the improvements needed to be, before the changes could be detected using conventional statistics. In our approach we probe the sensitivity of statistical techniques by selecting subsets of the original project data instead of modifying the original data.

A subset of the data that we examined in this study, has also been analysed using different statistical techniques in simultaneous study by Verhoef [Verhoef, 2005]. Verhoef demonstrates methods to audit size and cost measurements made by the organisation and demonstrates the use of time series to analyse the changes of productivity over time.

---

1We did not find any relevant results in ACM’s Portal, Springer-Verlag’s SpringerLink, Elsevier’s ScienceDirect or in IEEE Computer Society’s Digital Library that contained both a relevant statistical term (“hierarchical linear model”, “multi-level linear model”, “mixed-effects model”, “random-effects model”, “random-coefficient regression model”, or “covariance components model”) and a relevant application domain term (“spi”, “cmm”, or “software process improvement”).
irrespective of their cause (e.g. increased process maturity, changes in programming languages and tools or changes in operating systems). Conclusions from Verhoef’s study are that functional size measurements are completely consistent. Duration, costs and costs per function point are plausible when examining both their macro-and microbehaviour, using the generalised Pareto distribution and VAR models.

3.3 Research Methodology

3.3.1 Research Design

To answer our research questions, we use the experimental design of a cohort study [Cook and Campbell, 1979, p. 126]. We collected data on 410 software development projects and compared the productivity of projects that were executed in an CMM level 2 or 3 environment with projects that were executed in an environment that did not already fulfil the CMM requirements. For this comparison we used different statistical techniques.

The cohort study design has some weaknesses that create threats to the validity of the study, of which the maturation effect poses the biggest threat. Changes in the organisation occur over time that are unrelated to CMM, but that do have an effect on the productivity of the projects (c.f. [McGarry and Decker, 2002]). We found similar results in the different organisational departments that implemented SPI at different moments in time, which improves our confidence in the results.

In Sect. 3.4 we explain the different statistical techniques to analyse the impact of software process improvement in an attempt to answer the research question which alternatives to productivity indices exist to measure changes in productivity.

In this section we discuss the criteria to determine which statistical technique is most appropriate to analyse the effect of software process improvement on productivity.

3.3.2 Analysis Evaluation Criteria

Explained Variance

A first criterion that can be used to distinguish between the applicability of different statistical techniques is the amount of explained variance. The amount of explained variance is an expression of the ratio of variance that can be accounted for and the variance that cannot be account for by a statistical model. The amount explained variance is commonly expressed by the $R^2$ measure.

If only a limited amount of variance can be explained by a model, then the likelihood increases that other unknown causes influence the productivity of the organisation. If other unknown causes have a high influence on productivity, one can never be sure that the observed changes in productivity related to software process improvement are really caused by the process improvement and not by that unknown cause.

Statistical Power

A second important criterion to decide which statistical technique is superior for the comparison of productivity data, is the criterion of statistical power [Cohen, 1988,Bausell and Li, 2002]. The power of a statistical test is the chance of finding a statistically
significant difference between two or more populations. The power of a test is influenced by the effect size, sample size, significance level of the test and the type of test used. The effect size can be seen as a scale free measure to express the degree of difference of the different groups.

It is obvious that the power of a technique is influenced by the type of statistical technique that is used. It is less obvious that the effect size is also influenced by the choice of the statistical model. Conceptually effect size can be seen as the mean difference of two or more groups of data divided by their pooled variance. When we are able to reduce the variance of the data, we are effectively increasing the effect size of an analytical technique. This in effect leads to an increased statistical power of that technique.

The variance present in the data, that is not explained by group membership, can be reduced by taking covariates into account. Covariates are variables that also have an effect on the dependent variable (in our case effort), but that are not influenced by the independent variable (in our case process maturity). Previous research in the cost estimation field has demonstrated that size has a nonnegligible effect on effort. If we can reduce the influence of size on effort, we can more purely measure the effect of process maturity on productivity.

If the significance level and the sample size are kept at the invariant, the power of the test depends only on the type of test used and the effect size. In other words: the chance of finding a significant difference depends on the statistical test used and the effect size.

The effectiveness of the analytical techniques described in the previous subsections is assessed by simulation. From the original dataset samples are repeatedly drawn with a certain sample size. On these subsamples each statistical test is applied and it is determined if the test finds a significant difference. For each sample size, the frequency of finding a statistically significant result is determined for each type of test. This frequency of finding a significant difference is a measure for the statistical power of the test.

The simulation gives an indication of the chance of finding a change in productivity when the researcher has data on only a limited number of projects. If the researcher has a large number of projects the analytical technique is not of high importance, as the statistical power is also influenced by sample size. If the researcher on the other hand has access to only a limited amount of project data, as is usually the case, he or she had better make optimal use of the data in order to find significant changes in productivity.

### 3.3.3 Data Collection

To perform the evaluation of the software process improvement program, two sources of data were used: the project database and a log of the assessment results. The project database contains generic information on all projects executed in the organisation and the assessment log contains information on which domains have been assessed and what the outcomes of the CMM assessment were. From the database, data about 410 closed projects has been extracted.

From the project database the size in function points, the effort, the end date and the department in which the project was executed has been extracted for each project. To obtain the maturity of the organisation in which the project has been executed, the end date of a project was compared with the assessment date of its domain. If the end date of the project was after the assessment, the project was assumed to have been executed in an organisation with a maturity as established by the assessment (CMM level 2 or 3
if the improvement program was successful, CMM level 1 if the improvement program had not successfully introduced the new process). If the project has been executed before the assessment, the project has been considered to have been executed in an organisation with CMM maturity level 1.

The decision rule to determine the maturity of the department in which the project has been executed leads to a conservative estimate of the effects of SPI, as projects that were completed just before the assessment might very well also have benefited from an increased maturity.

Please note that before we analysed the data, we have multiplied the effort data with a random constant \((0.75 < \alpha < 1.5)\) for the sake of confidentiality of the actual productivity figures. This linear scaling of the data does not in any way affect the improvement ratios that are provided in this chapter.

### 3.4 Analysis Methods

In this section we describe three families of statistical techniques to compare the productivity of an organisation before and after a software process improvement program. These methods increase both in sophistication and in complexity. The methods we explain are: (a) productivity indices (b) regression models (c) hierarchical regression models.

#### 3.4.1 Productivity Indices

In most studies, productivity is defined as size divided by effort. Conte, Dunsmore and Shen [Conte et al., 1986, chap. 5] define productivity as: "the number of lines of source code produced per programmer-month (person-month) of effort". This leads to the following formula to calculate the productivity of a single project:

\[
l_{loc/m} = \frac{s_{loc}}{e_m}
\]  

(3.1)

where \(l_{loc/m}\) is the productivity of the project; \(s_{loc}\) is the size of the delivered software, measured in lines of code; \(e_m\) is the effort spent on developing, expressed in person months.

Analogously, we can define the current productivity of an organisation to be the average productivity of all projects that have been executed by that organisation:

\[
L_{loc/m} = \frac{1}{n_{proj}} \cdot \sum_{i=1}^{n_{proj}} l_{loc/m_i}
\]  

(3.2)

where \(L_{loc/m}\) is the productivity of the organisation, \(l_{loc/m_i}\) is the productivity of project \(i\), and \(n_{proj}\) is the number of projects executed in the organisation.

An alternative definition of organisational productivity is the weighted productivity index \(L_{loc/m}^w\). In contrast with the preceding definition, here the impact of a project on the overall organisational productivity is proportional with the project’s size. This leads to the following equation:
3.4. Analysis Methods

\[
L_{loc/m}^w = \left( \frac{\sum_{i=1}^{n_{proj}} s_{loc_i}}{n_{proj}} \right)^{-1} \cdot \sum_{i=1}^{n_{proj}} \left( l_{loc/m_i} \cdot s_{loc_i} \right)
\] (3.3)

Productivity indices are not limited to size measurements in lines of code. Size measurements of a project can also be expressed in function points [Albrecht, 1979], object points [Banker et al., 1992] or other size metrics and effort can also be expressed in person hours without loss of generality. Although function points have certain desirable properties, such as technological independence [Furey, 1997], Galin and Avrahami [Galin and Avrahami, 2005] observe that “most of the reporting organisations applied the classic lines of code (LOC) measure for productivity”. The dominance of lines of code can be explained by the ease with which this size metric can be obtained.

Productivity indices (\(L\)) can be used to measure whether the software development and software maintenance productivity has changed. One can either measure the productivity before and after the software process improvement program or the effect of an organisation’s process maturity on the productivity of its projects. If one measures the impact on productivity of the process maturity of the organisation in which a project is carried out, one can take into account that in some parts of the organisation the process improvement program might not have led to higher process maturity.

To determine if the effect of a software process improvement program is significant, the productivity indices of the project can either be compared using a t-test (two comparison groups) or an ANOVA (multiple comparison groups) [Bhattacharyya and Johnson, 1977]. The t-test and ANOVA both assume that the data have an approximately normal distribution and that all groups have the same variance. Unfortunately productivity indices follow a log-normal distribution instead of a normal distribution, which decreases the dependability of these tests.

In our study we choose for the unweighted organisational productivity index. A first argument for this decision is that unweighted indices can more easily be compared with the approaches in the following sections. The second argument is that each project instantiates the organisational project only once (a project produces a single plan, has one configuration management plan, etc.) and it is therefore fair to only count each project once.

As a size measure we used function points as a measure for size. In the remainder of the article we will use function points as the measure for size in our formulas unless noted otherwise.

This leads to a productivity index consisting of hour of effort of IT personnel (\(E_{hr}\)) per function points (\(S_{fp}\)). The effort of IT personnel includes not only programmer effort, but also the effort of requirements engineers, technical designers, architects and project management. This project index can be expressed with the following equation:

\[
\hat{L}_{hr/fp} = \frac{1}{n} \sum_{i=1}^{n} \frac{e_{hr_i}}{s_{fp_i}}
\] (3.4)

where \(\hat{L}_{hr/fp}\) is the estimated organisational productivity, \(e_{hr_i}\) is the effort of project \(i\) in hours, \(s_{fp_i}\) is the project size \(i\) in function points, and \(n\) is the number of projects executed in the organisation.
3.4.2 Regression Models

In productivity studies, changes in productivity are measured in a ratio of size divided by effort, most often lines of code per man month [Galin and Avrahami, 2005]. We contend that linear regression models are better able to express productivity, if one wishes to detect changes in productivity. Linear regression models are constructed on the same source data as productivity indices, yet they detect changes in productivity with fewer points of data.

Regression equations that model effort as a function of size are usually employed to build cost prediction models (such as COCOMO-II [Boehm et al., 2000]). Their use however is not limited to cost estimation, as regression models are also useful to compare the productivity of organisations or to determine the productivity effects of a software process improvement initiative.

The productivity of an organisation can be described by a linear regression model that expresses effort as a function of size. The first step is to choose an appropriate regression model form. The second step is to estimate the regression parameters (\( \beta_0 \ldots \beta_n \)) based on the available data. The set of regression parameters now functions as an alternative measure for productivity. Instead of comparing average cost levels, one can now compare cost models that also offer insight in which situation which process, tool or platform is most efficient.

A regression model of effort and size is appropriate for our purposes, when the regression model allows statistical inferences to be made. To allow statistical inference, the regression model must satisfy the assumptions of linear regression [Box and Cox, 1964]: (a) of structure (also called linearity of the relation), (b) constancy of error variance, (c) normality of the distributions, and (d) independence of observations.

Model form

The simplest regression model form is the simple linear regression model. It is linear because all factors have an additive effect on cost and it is simple because all variables are included as is. In contrast, other regression models could include scaled variables or interactions between variables. The equation for the simple linear regression model is as follows:

\[
e_{hr_i} = \beta_0 + \beta_1 \cdot s_{fp_i} + \epsilon_i
\]  

(3.5)

where \( e_{hr_i} \) is the effort of a project in hours, \( s_{fp_i} \) is the size of the delivered software in function points, and \( \epsilon_i \) is the unexplained, normally-distributed residual variance.

The simple linear regression model has been used as one of the earliest cost estimation models. The first software cost model ever, by Nelson [Nelson, 1966], is a simple linear model of aspects that influence cost and the first size-based cost model, by Wolverton [Wolverton, 1974], is also a simple linear regression model.

Unfortunately, in most organisations productivity data does not satisfy the assumptions of the simple linear regression model. In most organisations larger projects are less predictable and more prone to overrun their budget and schedules. This is an indication that the residuals (difference between observations and the regression model) are related to the size of a project, which is a violation of assumption (b) and can lead to invalid conclusions.
If the relation between effort and size violates one of the assumptions of the linear regression model, it is possible to transform the dependent ($E_{hr}$) and or independent ($S_{fp}$) variables before they are used in the regression equation.

Transformation of the dependent variable can reshape the distribution of the error variance into a (more) normal distribution and can improve constancy of error variance. Transformations of the independent variable can improve the fit of the regression model. As there is no consensus among researchers on which relation between effort and size is most appropriate (see Sec. 3.2), we advise the reader to select the transformation that is most appropriate to his or her set of data.

**Box-Cox transformations**

Many different transformations (e.g. square root, logarithm, square) can be applied to both the dependent variable (i.e. effort) and the independent variable (i.e. size). It is easy to miss a transformation that leads to good results, therefore a systematic search among the possible transformations is advisable.

Box and Cox have proposed a systematic procedure, the Box-Cox transformation [Box and Cox, 1964], that generalizes over a broad family of possible transformations. The Box-Cox transformation $x^{(\lambda)}$ (3.6) is parameterized by parameter $\lambda$. If $\lambda = 0$, the logarithm is taken of $x$, otherwise $x$ is raised to the power $\lambda$.

In their paper Box and Cox propose a maximum likelihood maximisation algorithm to estimate the transformation parameter $\lambda$ for dependent variables. To find optimal transformations for the independent variables or to find optimal transformation parameters for a simultaneous transformation of dependent and independent variables, the use of contour plots is proposed.

$$x^{(\lambda)} = \begin{cases} x^\lambda & \text{if } \lambda \neq 0 \\ \log(x) & \text{if } \lambda = 0 \end{cases}$$  \quad (3.6)

To select a transformation $\lambda$, one should not only observe goodness-of-fit indices (like $F$ or $R^2_{adj}$) [Matson et al., 1994], but also take diagnostic indices into account that check for normality and constance of error variance. Box and Cox [Box and Cox, 1964, Peter et al., 1996] propose Anscombe and Tukey’s indices for skewness, kurtosis, heterogeneity of variance and non-additivity [Anscombe, 1961, Anscombe and Tukey, 1963]. The proposed indices are not scale-free, which makes these indices difficult to interpret. Instead we use a correlation coefficient between the quantiles of the normal distribution and the error variance distribution [NIST/SEMATECH, 2006] ($\rho_{Q0},Q_{N(\sigma^2)}$) and a correlation coefficient between the squared error variance and the independent variable ($\rho_{R^2,S_{fp}}$) (which are based on the idea of the Breusch-Pagan test [Breusch and Pagan, 1979, Peter et al., 1996].

The Box-Cox transformation leads to a productivity model that is generic enough to comprise linear size terms, quadratic size terms and logarithmic size terms. The only production function (i.e. cost estimation function) that cannot be specified using a Box-Cox transformation is the trans-log function. Without conceptual difficulties the Box-Cox procedure can be expanded to multiple terms (e.g. linear and quadratic). The visualisation of the impact of the $\lambda$-terms would however be harder. The Box-Cox transformed regression model of size and effort can be notated as follows:
Effects of SPI on the Productivity of Software Development

\[ e_{hr_i}^{(\lambda_1)} = \beta_0 + \beta_1 \cdot s_{fp_i}^{(\lambda_2)} + \epsilon_i \]  

(3.7)

where \( e_{hr_i}^{(\lambda_1)} \) is the effort spent on project \( i \) after scaling it with Box-Cox transformation \( \lambda_1 \) and \( s_{fp_i}^{(\lambda_2)} \) is the size of the delivered software in function points, after scaling it with Box-Cox transformation \( \lambda_2 \).

Regression equations that model effort as a function of size are usually employed to build cost prediction models. Their use however is not limited to cost estimation, as regression models are also useful to compare the productivity of organisations or to determine the productivity effects of a software process improvement initiative.

**Additional model parameters**

After a suitable transformation has been found for the most important covariate (size), additional covariates can be added to the productivity model. To determine the significance of additional covariates (e.g. team size, development environment), step-wise procedures [Peter et al., 1996] can be used.

**Detecting productivity change**

The previous paragraphs explained the mechanism of describing productivity as a relationship between effort and size. This paragraph explains how this new expression of productivity is used to determine the impact of software process improvement on productivity.

When a suitable regression model has been found, the estimates of the regression parameters \( (\beta_0 \ldots n) \) define how size and effort are related. Therefore the regression parameters function as a substitute for the productivity index. If the projects are grouped into mature and immature projects, the regression parameters can be estimated for each group of projects. If the estimated regression parameters for the two groups differ, the productivity of the two groups differs and we can conclude that process maturity has an influence on productivity.

To determine if the observed differences in regression parameters have statistical significance, statistical tests (c.f. [Peter et al., 1996, chap. 2]) that test the differences in each individual regression parameter \( \beta_i \) could be used. Although the logic behind the these tests is clear, application of these tests would cause practical problems: the required multiple comparisons increases the experiment-wise error rate and not all statistical software packages have implemented tests to compare regression parameters.

To overcome these practical concerns, it is more convenient to factor the process maturity \( (C) \) of the organisation into the regression model and apply an ANOVA [Bhattacharyya and Johnson, 1977] on the augmented model:

\[ e_{hr_i}^{(\lambda_1)} = \beta_0 + \beta_1 \cdot s_{fp_i}^{(\lambda_2)} + \beta_2 \cdot c_i + \beta_3 \cdot s_{fp_i}^{(\lambda_2)} \cdot c_i + \epsilon_i \]  

(3.8)

where \( c_i \) is the maturity level (or CMM-level) of the organisation at the time when project \( i \) was executed.
3.4. Analysis Methods

If software process improvement has no effect, the regression parameters $\beta_2$ and $\beta_3$ should be equal to zero, which can be tested with ANOVA [Bhattacharyya and Johnson, 1977]. Otherwise, the size of the parameters $\beta_2$ and $\beta_3$ offer an estimate of the effect of software process improvement.

The advantage of the regression model approach is that regression models take the effect of project size on the productivity of a project into account. After all, projects can have startup costs and projects can experience (dis-)economy of scale. Ignoring the effect of project size on project productivity increases the residual, unexplained variability in the data. This increased residual variance means that larger sample sizes are needed to obtain significant results. In certain situations it is even possible that, by ignoring the effect of size on productivity, invalid conclusions are drawn (e.g. if the size of projects before and after the software process improvement changes significantly).

### Weighing of Observations

In the approach chosen we do not weigh projects according to their size of the produced software. Although this is appropriate if we want to investigate the influence of SPI in general (as each project is only a single instance of the process) it is not useful for companies that want to understand the impact on their bottom-line.

For companies, productivity is often judged by total output divided by total effort. To take this into account, one could employed weighted regression models, in which projects are weighed according to their size.

#### 3.4.3 Hierarchical Linear Models

Large organisations structure their work according to the principle of division of labour. These large organisations consist of different departments that perform different projects. These departments either specialise on the group of products they work on or specialise on the technology or skills that the department uses. We call these organisations with specialised departments heterogeneous organisations. In this section we explain why data from heterogeneous organisations cannot be adequately analysed using classical linear regression models and how hierarchical models [Lindley and Smith, 1972, Bryk and Raudenbush, 1992] can be used to analyse this data.

Linear regression models are based on the assumption that residuals in a model are independent, and therefore the observations need to be drawn from a single homogeneous pool of subjects. The characteristics of such a homogeneous pool from which the observations are drawn should have the same statistical distribution. Because the characteristics of departments in a heterogeneous organisation differ, the assumption of independent residuals does not hold.

We illustrate the problems sketched above with a hypothetical example. If we want to understand the effects of a type of fertiliser on the growth-rate of fruit, an experiment could be set up with eight plots of apples and eight plots of pears, four plots of bananas and two plots of pineapples. Half the plots are assigned with the new fertiliser (the treatment group) and half the plots are assigned with no fertiliser (the control group). At the moment the plants bear fruit, the yield of each of the plots is weighed. To determine the effect of the fertiliser (a) the yields of all plots with fertiliser could be compared with the yields of all plots without the fertiliser or (b) for each type of fruit the average yields
of plots with fertiliser could be compared with the plots grown without fertiliser. In the first approach, we literally compare apples with pears, which severely increases error variance and therefore reduces the chance we detect the effect of the fertiliser. If we on the other hand take the second approach to gauge the effect of the fertiliser, we have to make four comparisons, which increases the chance of making an error by fourfold.

When determining the effect of software process improvement on productivity, the productivity of projects that took place before and after the software process improvement initiative are compared. Unfortunately the number of projects that take place within a single department of a company is usually too small to find significant results of software process improvement. If the productivity of projects that took place in different departments is compared, effectively ‘apples’ are compared with ‘pears’. Hierarchical linear models can be used to make a single comparison of the overall effects of changes in software process (or fertiliser) and at the same time take into account that we are comparing ‘apples’ with ‘pears’. Multi-level linear models, mixed-effects models, random-effects models, random-coefficient regression modes, and covariance components models are other names for hierarchical level models. To measure the overall effects of software process improvement, hierarchical linear models should be used to take the differences of the departments into account.

Hierarchical linear models are an extension of ordinary linear regression models. In a hierarchical linear model, the regression model is split up in two components: a level 1 model and a level 2 model. Hierarchical linear models extend linear regression models by fitting a new set of regression parameters for each group of data, the department in which the project took place. For each group a different set of regression parameters $\beta_0 \ldots \beta_n$ is found. The level 2 model brings structure in the regression parameters; an overall $\gamma_{n0}$ value for a group of parameters $\beta_{nj}$ is determined, from which each group $j$ is allowed to deviate by $u_{nj}$.

The level 1 model for effort looks similar to the regression model from the previous section. Note however that the projects $i$ (the observations) are grouped according to the department $j$ in which they were made. Also note that the parameters in the level 1 regression model also have a subscript for their group $j$:

$$e_{hr_{ij}}^{(\lambda_1)} = \beta_{0j} + \beta_{1j} \cdot s_{fp_{ij}}^{(\lambda_2)} + \beta_{2j} \cdot c_{ij} + \beta_{3j} \cdot s_{fp_{ij}}^{(\lambda_2)} \cdot c_{ij} + \epsilon_{ij}$$

(3.9)

where $e_{hr_{ij}}^{(\lambda_1)}$ is the scaled effort of a project in hours. $s_{fp_{ij}}^{(\lambda_2)}$ is scaled size of effort of a project in function points, and $c_{ij}$ is the maturity level of the organisation in which the project is executed. $\epsilon_{ij}$ is the unexplained residual variance, which is normally distributed.

We also have a level 2 model for productivity, which breaks up each $\beta_{kj}$ in the level 1 model into an organisation wide parameter $\gamma_{kj}$ and a deviation $u_{kj}$ from that organisation average for each department $j$:

$$\beta_{kj} = \gamma_{k0} + u_{kj} \quad u_{kj} \sim N(0, \tau_{kk})$$

(3.10)

Bryk and Raudenbush [Bryk and Raudenbush, 1992, chap. 3] provide a conceptual explanation of how these models can be fitted and a more thorough mathematical de-
3.5 Results

Table 3.1: Productivity Indices per Maturity Level.

<table>
<thead>
<tr>
<th>Maturity</th>
<th>CMM Level</th>
<th>Productivity $L_{hr/fp}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>low</td>
<td>1</td>
<td>14.46</td>
</tr>
<tr>
<td>medium</td>
<td>2</td>
<td>12.08</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>8.50</td>
</tr>
<tr>
<td></td>
<td>2 &amp; 3</td>
<td>11.54</td>
</tr>
</tbody>
</table>

A description of how values for these parameters can be found is provided by Pinheiro and Bates [Pinheiro and Bates, 2000, chap. 2].

To determine whether software process improvement has an effect, the parameters $\gamma_{20}$ and $\gamma_{30}$ should be tested for equality with zero. The parameters $\gamma_{20}$ and $\gamma_{30}$ are related with the maturity of the organisation ($c_{ij}$) and if the parameters are equal to zero that would indicate that SPI has no effect.

The advantage of using hierarchical linear models in determining the effects of SPI in heterogeneous organisations is that by taking the department into account, the residual variance of the data is reduced, which reduces the amount of data required to make an analysis. Furthermore the usage of hierarchical linear models can guard against making erroneous conclusions. Such erroneous conclusions could be made if workload of departments that perform easy assignments increases at the expense of the workload of departments that perform difficult assignments. In such cases the productivity of the organisation would seem to have increased, whereas in reality the work has changed and no real performance increase has occurred.

3.5 Results

Bear in mind when interpreting the results, that before we analysed the data, we multiplied the effort data with a random constant ($0.75 < \alpha < 1.5$) for the sake of confidentiality of the actual productivity figures. This scaling of the does not have an impact on the improvement ratios or sensitivity measures that are provided in this section.

3.5.1 Productivity Indices

In this section we use the classical approach to determine the effects of software process improvement. Table 3.1 shows the average productivity of projects that are executed in a CMM Level 1, 2 and 3 organisation.

From Table 3.1 we can conclude that projects executed in a CMM level 2 or level 3 organisation are on average 20.19% more productive than projects that are executed in a CMM level 1 organisation. When we use a t-test to test hypothesis which states that $L_{spi}^{hr/fp}$ is not equal $L_{spi}^{-hr/fp}$, we find that have to reject $H_0$ with $p = 0.002$ ($t = 3.13$, $df = 267$). We can therefore conclude that there is a significant productivity increase after the implementation of SPI.
Although we do find statistically significant results with the classical approach, the results are not satisfactory if we look at the amount of explained variance. When we fit productivity as a function of process maturity (level 1 vs. level 2 and 3), we obtain \( R^2 = 0.02 \). This means that only 2% of the differences in productivity can be explained by process maturity.

### 3.5.2 Linear Regression Models

![Residuals of Linear Models plotted against the Fitted Values.](image)

**Figure 3.1:** Residuals of Linear Models plotted against the Fitted Values.

We use linear regression models to determine the effect of software process improvement on productivity. As explained in Sect. 3.4.2, we first need to find a suitable transformation for the regression model between effort and size.

**Table 3.2:** Diagnostic Information on Effort Regression Models.

<table>
<thead>
<tr>
<th>Formula</th>
<th>Goodness of fit</th>
<th>Constance of Variance</th>
<th>Normality</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E = \beta_0 + \beta_1 S + R )</td>
<td>( R^2 = 0.493 )</td>
<td>( X^2_{bp} = 86.07 )</td>
<td>( W^a = 0.798 )</td>
</tr>
<tr>
<td>( E = \beta_0 + \beta_1 S^2 + R )</td>
<td>( R^2 = 0.315 )</td>
<td>( X^2_{bp} = 87.29 )</td>
<td>( W^a = 0.748 )</td>
</tr>
<tr>
<td>( E = \beta_0 + \beta_1 \log(S) + R )</td>
<td>( R^2 = 0.407 )</td>
<td>( X^2_{bp} = 33.144 )</td>
<td>( W^a = 0.830 )</td>
</tr>
<tr>
<td>( \log(E) = \beta_0 + \beta_1 S + R )</td>
<td>( R^2 = 0.436 )</td>
<td>( X^2_{bp} = 27.26 )</td>
<td>( W^a = 0.981 )</td>
</tr>
<tr>
<td>( \log(E) = \beta_0 + \beta_1 S^2 + R )</td>
<td>( R^2 = 0.202 )</td>
<td>( X^2_{bp} = 54.24 )</td>
<td>( W^a = 0.988 )</td>
</tr>
<tr>
<td>( \log(E) = \beta_0 + \beta_1 \log(S) + R )</td>
<td>( R^2 = 0.576 )</td>
<td>( X^2_{bp} = 4.43 )</td>
<td>( W^a = 0.995 )</td>
</tr>
</tbody>
</table>

*Breusch-Pagan \( X^2 \) to test for dependence between predictors and residuals.

*Shapiro-Wilk W to test deviation of residuals from the normal distribution.
3.5. Results

In Table 3.2 we tabulated diagnostic information on six different combinations of transformations on both the dependent (effort) and the independent variable (size). The Shapiro-Wilk test [Shapiro and Wilk, 1965] is used to test for normality and the Breusch-Pagan test [Peter et al., 1996, p. 115] is to test the constance of variance of the residuals. The residuals of the linear model and the log-transformed model \((\log(E) = \beta_0 + \beta_1 \log(S) + R)\) are shown in Fig. 3.1. From Fig. 3.1 we can see that the residuals from the linear model are correlated with the fitted values (show heteroscedascity) and that the residuals from the logistic model are uncorrelated with the fitted values (homoscedastic). Homoscedasticity, or constance of variance, is assumed in linear regression models and therefore the logistic model is superior to the simple linear model. For goodness of fit we used the unadjusted multiple coefficient of determination \(R^2\), which tells us how much of the variation in effort can be explained by size.

If we examine the diagnostic information in Table 3.2, we can see that the log-transformed model \((\log(E) = \beta_0 + \beta_1 \log(S) + R)\) has the best characteristics; 58% of the variance is explained and its residuals are normally distributed and the variance of the residuals only has negligible relation with size.

We can also take a more general approach. In Fig. 3.2 a contour plot is shown of simultaneous Box-Cox transformations on effort and size. The contour plot is an overlay of three individual contour plots: a contour plot displaying the normality of the error variance \((\rho_{Q_n,Q_N(0,\sigma^2)})\), a contour plot showing the constance of variance \((\rho_{R^2,S_{fp}})\) and a contour plot display the Goodness of Fit \((R^2)\). An acceptable value for \(\lambda\) has an error variance that is approximately normal \(\rho_{Q_n,Q_N(0,\sigma^2)} > 0.99\) and has high constance of variance \(\rho_{R^2,S_{fp}} < 0.05\). Values for \(\lambda\) that satisfy these two constraints are coloured grey, unacceptable values are in the white area. The contour lines in Fig. 3.2 display the index for Goodness of Fit \((R^2)\). We are therefore looking for values on a grey blob that have a high \(R^2\). The transformation \(\lambda_1 = \lambda_2 = 0\) satisfies these constraints and has as an additional advantage that it is the Douglas-Cobb production function [Cobb and Douglas, 1928] that has desirable mathematical properties. It turns out that we effectively arrive at an exponential effort estimation model [Conte et al., 1986, p. 281] of the form \(\text{effort}_i = \beta_0 \ast \text{size}^{\beta_1} \ast e^{\epsilon_i}\). To determine if the maturity of an organisation has an effect on the productivity, we effectively are comparing cost-estimation models.

After further analysis we see that the log-transformed model \((\lambda_1 = 0 \text{ and } \lambda_2 = 0)\) has the best characteristics; 58% of the variance is explained and its residuals are normally distributed and the variance of the residuals only has negligible relation with size. In Fig. 3.1 the dependence of variance on size is visualized. On the left side we see the simple linear regression model and on the right side the log-scaled regression model, for which the variance is independent of size. The log-transformed model has the following equation:

\[
\exp(e_{hr_i}) = \exp(\beta_0) \cdot s_{fp_i}^{\beta_1} \cdot \exp(\epsilon_i)
\]

where \(e_{hr_i}\) is the effort of a project \(i\) in hours, \(s_{fp_i}\) is the size of project \(i\) in function points, and \(\epsilon_i\) is the unexplained residual variance, which is normally distributed.

Having selected an appropriate regression model, we continue by testing the hypothesis that process maturity influences on productivity with ANOVA (formula \(\log(E_{hr}) = \beta_0 + \beta_1 \cdot \log(S_{fp}) + \beta_2 \cdot C + \beta_3 \cdot \log(S_{fp}) \cdot C\)). The results of the ANOVA can be found in Table 3.3:
From the ANOVA we can see that both size \((\log(S_{fp}))\) and process maturity \((C)\) have a significant effect on the effort \((\log(E_{hr}))\). Furthermore we can observe that there is no interaction between size and process maturity, which means that software process improvement has a similar (positive) effect on both large and small projects. If we examine the regression coefficients, we arrive at the following relations between effort and size, which overall is a 20.86% improvement of productivity for CMM level 2 & 3 projects over CMM level 1 projects.

\[
\begin{align*}
\text{CMM level 1: } & \quad E_{hr} = 31.68 \cdot S_{fp}^{0.80} \\
\text{CMM level 2: } & \quad E_{hr} = 26.35 \cdot S_{fp}^{0.80} \\
\text{CMM level 3: } & \quad E_{hr} = 18.93 \cdot S_{fp}^{0.80}
\end{align*}
\] (3.12)
If we look at the explained variance, we see an $R^2 = 0.60$. This means that 60% of the variation in effort can be explained by process maturity and size. Although still only 2% of the variance can be explained by software process improvement (the difference between 58% and 60%), we have decreased the chance that the results can be explained in an alternative way.

### 3.5.3 Hierarchical Model Approach

In this section we examine the effects of software process improvement with hierarchical linear models. In the previous section we established that the log-scaled model best fits the data ($\log(E) = \beta_0 + \beta_1 \log(S) + \beta_2 C + R$). In a similar manner we examine the influence of domain on the regression coefficients ($\beta_1 ... \beta_3$) in Table 3.4. This table contains the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC) [Pinheiro and Bates, 2000] and the log-likelihood of each model. From Table 3.4 we see that model $\log(E) = \beta_0 j + \beta_1 \log(S) + \beta_2 C + R$ has the lowest AIC and therefore is the best balance between goodness-of-fit and number of parameters. If we compare the likelihoods with the optimal regression model from the previous section, we obtain that the hierarchical linear model is significantly better (log-likelihood ratio=22.27645, $p < .0001$). Although some other hierarchical linear models have an even lower log-likelihood, this difference is not significant.

When we perform an ANOVA on model $\log(E) = \beta_0 j + \beta_1 \log(S) + \beta_2 C + R$ to test whether process maturity has an influence on productivity, we obtain the results which are shown in Table 3.5:

So, significant effects of process maturity on productivity are not only found if we analyse the data with linear regression models, but also if we analyse the data using hierarchical linear models. As hierarchical linear models take the impact of both size and
organisation into account, we rejected that these obvious alternative explanations explain
the change in productivity instead of software process improvement. Taking organisation
into account when analysing the data increases the explained variance from 60% to 67%
\((R^2 = 0.67)\). This increases the confidence in our results.

If we examine the regression coefficients, we obtain an 23.42% overall productivity
increase for CMM level 2 & 3 organisation when compared with a CMM level 1 organ-
isation. Examining the regression coefficients leads to the following, organisation-wide
models for productivity:

\[
\begin{align*}
\text{CMM level 1:} & \quad E_{hr} = 33.09 \cdot S_{fp}^{0.82} \\
\text{CMM level 2:} & \quad E_{hr} = 26.68 \cdot S_{fp}^{0.82} \\
\text{CMM level 3:} & \quad E_{hr} = 20.40 \cdot S_{fp}^{0.82}
\end{align*}
\]

3.5.4 Power Simulation

The results of our simulation to determine the statistical power of the described statistical
techniques are described in this section. We first start a large simulation that compares the
power of the classic approach to compare productivity with the simple linear regression
model (not actually used in the case study) and the log-scaled regression model. We
conclude the section with a simulation in which the classic approach is compared with
the simple linear regression model, the logistic regression model and the hierarchical
model.

The first simulation compares the sensitivity of the t-test on productivity indices with
the sensitivity of regression analysis. From the original dataset balanced samples were
drawn, each sample included the same number of projects that has a maturity of level 1
and projects that have a maturity of level 2 or 3.

The sample size per group was varied from 5 to 100, which means that between 10
and 200 projects have been selected. For each sample size 2500 different samples were
drawn and on these samples a t-test on the difference in productivity indices, and an
ANOVA on the linear and logarithmic model was performed. For each of these test it
was determined if statistically significant results were found (at the \(\alpha < 0.1\) level)

The results of this simulation is shown in Fig. 3.3. The results clearly show that the
logarithmic model is best in obtaining statistically significant results. Because we know
the effect of CMM level is present in the total dataset, we could interpret this figure as
the chance an investigator would have to find statistically significant results if he or she
was only able to draw a small sample of projects.
3.5. Results

The second simulation compares the sensitivity of the t-test on productivity indices with the sensitivity of regression analysis and the sensitivity of the hierarchical models. From the original dataset balanced samples were drawn, each sample included the same number of projects that has a maturity of level 1 and projects that have a maturity of level 2 or 3 and the same number of projects for each group of domains.

In order to be able to draw balanced data, the domains/departments were clustered in columns, an organisational structure of the IT in which one classifies all domains in one of five columns. If the simulation was run with domains as the grouping variable, there was insufficient data to draw larger balanced samples, as certain domains only hosted 5 projects. The fact that the structuring of the domains was done by the organisation and is product-oriented gives some assurance that the results of the simulation might be generalisable to domains.

Again from this pool of data random samples were drawn. The sample size per
group was varied from 1 to 20, which means that between 10 and 200 projects have been selected. For each sample size 250 different samples were drawn and on these samples a t-test on the difference in productivity indices, and an ANOVA on the linear, logarithmic model and hierarchical model was performed to assess if the influence of the parameter $C$ was significant. For each of these test it was determined if statistically significant results were found (at the $\alpha < 0.1$ level)

The results of this simulation is shown in Fig. 3.4. The results show that the logarithmic model is somewhat better in obtaining statistically significant results, than the hierarchical model (which grouped the data by column). As the power of the two techniques appear to be similar, one can deduce that the additional assurances provided by hierarchical linear models do not come at the cost of high reductions in statistical power.
3.6 Conclusions

From the study we have found clear evidence that CMM does increase the productivity of an organisation. We found a productivity increase of 20%. More planning and more attention to management and work processes do seem to have a positive effect on the productivity of the organisation. The improvements made in this study are smaller than found in certain similar studies, but we believe that this can might be explained because in some studies small convenience samples are analysed instead of the productivity data on all projects in that organisation.

In addition we found that the classical method of comparing productivity indices has a lot of disadvantages, as only a tiny part of the variance can be explained by the maturity level. By using more sophisticated statistical techniques, linear regression models and hierarchical linear models, we gain confidence in the results of the analysis as the underlying assumptions of the analytical techniques are met and the influence of alternative explanations for the change in productivity are excluded. Linear regression models allow us to exclude the impact of project size on changing productivity and hierarchical linear models allow us to exclude the impact of the department or organisational unit in which the project takes place. It gives confidence in the results that the increases found in productivity using both statistical methods are approximately equal. In addition the more sophisticated methods promise higher statistical power, which means that with less data researchers are still able to find statistically significant changes in productivity.
Chapter 4

Effects of Facilitated Workshops on Requirements Engineering Productivity

In the previous chapter, Chap. 3, we demonstrated that increasing process maturity does lead to improved productivity of the overall software development process. In this chapter we focus on the benefits of improving the requirements elicitation process by using facilitated workshops to gather requirements.

When investigating a single aspect of the software development process, instead of performing a holistic investigation of the entire software development process, one gains more understanding of why a process has improved or deteriorated. This advantage comes with the risk that one might see improvements in a single aspect of the development process but at the same time miss deteriorations in the overall development process due to sub-optimisation.

This chapter is based on a paper presented earlier at the 8th Conference on Evaluation & Assessment in Software Engineering (EASE-2004) [Schalken et al., 2004].

4.1 Introduction

The role of facilitated workshops in the development of information systems is widely acknowledged, but apart from some anecdotal evidence little objective information about the effectiveness of facilitated workshops is available. This chapter describes a study within a single organisation in which the effectiveness of requirements engineering in projects using facilitated workshops is compared with the effectiveness of past projects that used one-on-one interviews to gather requirements instead of facilitated workshops.

In the study the duration, effort and satisfaction of 49 DSDM projects using facilitated workshops have been compared with 20 projects that used the Method/1 method, which use one-on-one interviews to gather the requirements. For small projects, Method/1’s one-on-one interviews are found to be more efficient, whereas for larger projects the efficiency of one-one-one interviews is surpassed by DSDM’s facilitated workshops. Other quantitative effects were not found, and subjective ratings of stakeholders do not indi-
cate a preference for DSDM projects either. We conjecture that these findings may be partly due to the fact that we only measured short-term effects (such as customer satisfaction immediately after a project is completed), and that the true benefits of facilitated workshops could require longitudinal studies.

**Facilitated workshops** are intensive meetings in which technical staff, end-users and management collaborate on information systems development tasks, such as project planning, requirements specification and user interface design. Facilitated workshops fit in the general tendency to increase the involvement of stakeholders in the requirements engineering process. The use of facilitated workshops in information system development has been introduced by Chuck Norris in 1977 [Rush, 1997], based on the work of Michael Doyle and David Strauss [Doyle and Strauss, 1976]. Facilitated workshops are best known for being a crucial component of Joint Application Development [Martin, 1991c, Martin, 1991a] and Participatory Design [Carmel et al., 1993], but facilitated workshops are also used in other development methodologies.

Facilitated workshops are led by a facilitator. A facilitator is a leader of workshops, trained in group dynamics, that is 'free from vested interests' [Mumford, 1993] as cited in [Macaulay, 1998]. Although not all facilitators perform the same role, some common patterns in the role of a facilitator can be distinguished.

### 4.1.1 Evaluating the Effectiveness of Facilitated Workshops

The costs of facilitating workshops are considerable. It is therefore reasonable to ask what benefits can be expected from the introduction of facilitated workshops. The costs consist of both introduction costs (training workshop facilitators and educating IS staff and end users about facilitated workshops) and operating costs (the time of both participants in the facilitated workshops and that of the workshop facilitator plus the required facilities).

In manuals on rapid application development and facilitation in the popular press many benefits of facilitated workshops have been reported. These benefits include [McConnell, 1996a]: commitment from top executives to the software planning-process, shortening requirements specification phase, eliminating features of questionable value, helping to get the requirements right the first time, helping to get the user interface right the first time, reduction of organisational infighting. Some attempts have been made in manuals and the popular press to quantify time savings, costs savings, completeness of requirements collection and satisfaction (see for example [Carmel et al., 1993, Martin, 1991a]).

Despite the claims made, little is known about the real efficacy of the workshop approach [Davidson, 1999]. In most popular studies no details are disclosed about the research methodology used and the data obtained [Carmel et al., 1993], which makes the assessment of the validity problematic at best. This lack of objective data on facilitated workshops might be explained because facilitation of workshops has been developed outside the academic world [Carmel et al., 1993] and because most organisations lack the required data on systems development to allow evaluation of facilitated workshops [Davidson, 1999]. More academic effort has been put into the evaluation of facilitation in the context of group support tools (see for example [Damian et al., 2001]). Facilitated workshops supported by tools and focusing on analysis models (such as ER-diagrams and UML diagrams) have been termed Rapid JAD by Martin [Martin, 1991c]. The debate
whether facilitated workshops focusing on models and tools are always an improvement is still open [Davidson, 1999].

4.1.2 Research Questions and Results

This study attempts to add to the body of knowledge about facilitated workshops by evaluating the effectiveness of facilitated workshops, by asking the following three questions:

1. What is the advantage of facilitated workshops in terms of the quality of the requirements when compared to one-on-one interviews?

2. What is the advantage of facilitated workshops in terms of calendar time when compared to one-on-one interviews?

3. What is the advantage of facilitated workshops in terms of effort (in requirement engineering) versus quality (of the requirements) when compared to one-on-one interviews?

Our results only show a significant effect of facilitated workshops for larger projects. For smaller projects, the added effort needed to conduct facilitated workshops seems not to outweigh possible other positive effects. We did not find the expected positive effect of facilitated workshops on stakeholder satisfaction.

4.2 Related work

Our study focuses on the use of facilitated workshops in the context of traditional software development. Studies on the use of facilitation in the context of meetings supported by Computer Supported Cooperative Work tools or Integrated CASE tools, as discussed in Sect. 4.1, are not taken into consideration.

The first scholarly study we are aware of is the work performed in 1977 by Unger and Walker [Unger and Walker, 1977]. This study reports a case study on an operating systems course, in which the students partially implemented an operating system as a practical assignment for the course. The course, in this form, has been offered four times. In two of the courses, the students were supported in their communications by a professional facilitator schooled in group dynamics. The two groups that were assisted by a professional facilitator were approximately twice as productive (based on the number of lines of code delivered and the amount of effort expended) when compared with the groups that did not receive any facilitation.

This study covered the full development life-cycle and not just the requirements engineering phase. Since the students had to come up with the requirements (lacking an external sponsor) the problems faced during the project also included requirements analysis. The effort spent by the students during the practical assignment has not been formally tracked, but instead estimated based on the study load of the students. This makes the measurement of the expanded time less reliable. Another drawback of this study is that it is performed on college students and therefore the external validity of the study is not evident.

In 1999 Davidson, observing the lack of transparent, objective evaluations of the benefits of using Joint Application Development to gather requirements specifying systems
properties, performed a study into the efficacy of JAD as a software process improvement method [Davidson, 1999]. As the organisations investigated by Davidson did not collect statistical data on development time, costs and errors, no quantitative evaluation of the benefits were possible. Instead, Davidson conducted 34 interviews with facilitators and managers (covering 20 distinctive projects) to investigate the users’ evaluations of the JAD method. This study reports that only 10% of the informants believed the requirements had been specified faster using JAD and none of the informants believed the total system development time had been reduced.

The strongest point of Davidson’s study is that it does not only take productivity into account, but also the less tangible factor of user satisfaction, which is a derived measure for the quality of the process. The study also attempts to explain why the benefits of JAD have not been fully reached in the projects examined. Unfortunately no objective, quantified measurements are available to determine whether development time and costs have dropped or not. As this study contradicts previous studies that claim higher productivity, it would have been useful to be able to determine if the development time and costs really remained the same or were only believed to remain the same. This is why objective, quantified metrics are needed in method evaluations. In the study described in this chapter we therefore use objective, quantified measurements to objectively assess the schedule and productivity effects.

Hubbard, Schroeder and Mead performed a comparison between facilitator-driven requirements collection processes and conventional interview methods [Hubbard et al., 2000]. In this study the researchers compare eight projects using facilitator-driven requirements collection with four projects using the conventional, unstructured interview technique. For each of the projects the yield (number of requirements), effort and duration was registered. Based on an analysis of the data, the researchers found that the effort to obtain the same amount of function points was reduced by a factor of 2.8 for facilitator-driven requirements collection and the duration per function point has decreased by a factor of 9.8 (both results are significant at the 0.05 \( \alpha \)-level).

Our study is similar to the one performed by Hubbard et al. We pay attention to not only the quantitative indicators of the performance of the project, but also to subjective information. By repeating a quantitative study into the effects of facilitator-driver requirements collection, confidence is built that the result obtained by Hubbard, Schroeder and Mead will also be obtained in a different context.

### 4.3 Research Methodology

#### 4.3.1 Research Context

In Sect. 1.2, we have explained the overall software process improvement program that has been implemented at ABN AMRO Bank. Part of the improvement program consisted of the introduction of the Dynamic Systems Development Methodology (DSDM) [Stapleton, 2002] in the IT department. Facilitated workshops are a core technique of DSDM to reach decisions in a team for, amongst others, requirements engineering [Stapleton, 2002, pp. 71–72].

In the organisation a policy stated that projects must start by gathering requirements using facilitated workshops in the Business Study phase and the Functional Model Iteration phase (see Sect. 1.2 for more information on the phases of DSDM). The organisation
4.3. Research Methodology

Technical Design

Functional Design

Project Proposal

Quick Scan

... Aftercare

High-Level Requirements Phases

Low-Level Requirements Phases

Feasibility Study

Business Study

Functional Model Iteration

Design & Build Iteration

Dynamic Systems Development Method (DSDM)

Figure 4.1: Mapping Method/1 phases to DSDM phases

has a number of trained workshop facilitators, who are available for these workshops (more on the facilitation process in use can be found in [CGEY, 2002, van Krugten-Elgersma and Hoogenboom, 2000]).

4.3.2 Comparing different Development Cycles

To investigate the effects of facilitated workshops projects performed with the Method/1 Custom Systems Development method must be comparable to ones that are run with the new DSDM method. A mapping of the two methods to a more generic model is therefore required to allow meaningful comparisons between the old and new projects (see Fig. 4.1). Based on the intent of the phases and the intent of the deliverables of the phases, the Quick Scan and Project Proposal phases of Method/1 have been mapped to the Feasibility Study and Business Study phases of DSDM. Furthermore the Functional Design phase of Method/1 has been mapped to the Functional Model Iteration phase of DSDM. This way two requirements engineering phases are created: a high-level requirements engineering phase and a detailed-requirements engineering phase.

4.3.3 Data Collection

Two sources of evidence are used to assess the merits of facilitated workshops: the required effort and the duration of the low-level requirements engineering phase and the size (in NESMA function points [Nesma, 1997]) is extracted from the organisation’s project administration database. The satisfaction of the project leaders, customers and IT employees about the project is determined from the organisation’s project evaluation database.

To assess whether projects that employed facilitated workshops are really more efficient in terms of required effort, the organisations’ project database has been used. The organisation has implemented DSDM gradually into the organisation. As a consequence, Method/1 and DSDM were used next to each other for a period of over two
years, the period to which our data refer. All DSDM projects using facilitated workshops have been compared to Method/1 projects that used the traditional one-on-one interview technique to gather project requirements. Since the terminology and phase names in these two methods are not equal, the mapping, outlined in Sect. 4.3.2, has been used to compare DSDM projects with Method/1 projects. A comparison of both high-level and low-level requirement specification phases was planned. Upon examining the project database comparing high-level requirements specification phases proved infeasible, as a large proportion of projects at the organisation share the high level requirements phase between multiple small implementation projects. Attributing the effort of the collective high level requirements engineering phase to the individual development projects would require a lot of speculation. Therefore only attention is paid to the low-level requirement specification phase.

The only reliable effort data in the project database concerns that of IT personnel. So we only take those into account in our effort equations below. In particular, effort spent on requirements engineering by non-IT personnel, such as business representatives, is not taken into consideration.

In order to gain insight into the subjective opinions regarding the use of the DSDM method as compared to Method/1 projects, we also used a database with the evaluations and lessons learned about past projects. At the end of each project a mandatory evaluation of the project is conducted using an electronic tool. The evaluation consists of both a list of closed questions that are rated on a scale from 1 to 10 and of a list of open questions regarding the project. The project manager, the customer and the IT employees all participate in the evaluation of projects. For this research closed questions from the evaluation form have been selected that deal with subjective ratings of the requirements clarity and with subjective ratings that deal with overall project satisfaction. For the latter category only the project managers and customers rated the project.

The data from the project administration database and the evaluation database have been extracted and converted into a format suitable for statistical analysis using S-Plus version 6.0.

4.4 Discussion of results

In this section we discuss the analysis of effort, duration and satisfaction in both types of projects. Effort concerns the number of hours spent by IT personnel on requirements activities. Duration concerns the lead time of the requirements phases, as recorded in the project database. Satisfaction concerns the subjective ratings of the stakeholders regarding project results and requirements clarity.

4.4.1 Effort

Comparing the hours of effort spent in low-level requirements analysis for both DSDM projects with facilitated workshops and Method/1 projects that did not use facilitated workshops at first does not yield much difference. In Fig. 4.2(a) a scatter plot is given of hours spent on requirements engineering per function point. Although some of the Method/1 projects perform considerably worse than the DSDM projects, no obvious pattern can be discerned.
4.4. Discussion of results

If we however plot the productivity in requirements engineering, in terms of hours spent by ICT personnel per function point against the size of the projects, a pattern does appear. When we examine only the small software development projects some projects using facilitated workshops perform worse than their traditional Method/1 counterparts. When we examine large projects the opposite can be observed: here Method/1 projects under-perform while facilitated workshops are more productive.

To determine whether small projects are really better off without facilitated workshops and large project are really better off with facilitated workshops, the statistical significance of the pattern is tested with an ANOVA analysis. If the effect caused by the project size (in function points) on the effectiveness of the requirements engineering technique (facilitated workshops or one-on-one interviews) cannot be explained by chance alone, the interaction between the factors technique and size should be significant.

The dependent variable productivity is severely left-skewed. To obtain an approximately normally distributed variable, the dependent variable has been transformed with an square root transformation. The transformed variable has been used in the ANOVA analysis.

When testing the interaction effect between requirements engineering technique and project size with an ANOVA test, the interaction effect (technique:size in Table 4.1) turns out to be significant well beyond the 0.05 $\alpha$-level. The first line (labelled ‘technique’), reflects the influence of the requirements engineering method on total effort. The second
Effects of Facilitated Workshops on Requirements Engineering Productivity

Figure 4.3: Effect of requirements engineering technique on required calendar time for requirements engineering.

Using regression analysis, the turning point where Method/1 projects become less productive than DSDM projects with facilitated workshops can be determined. For our set of projects, the regression lines intersect at 171.4 function points. This means that requirements engineering using Method/1 is more productive for projects smaller than the threshold of 171.4 function points.

Table 4.1: ANOVA analysis of the effects of technique and size on requirements engineering effort per function point

<table>
<thead>
<tr>
<th></th>
<th>df</th>
<th>SSE</th>
<th>MSE</th>
<th>F</th>
<th>p(F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>technique</td>
<td>1</td>
<td>0.06</td>
<td>0.06</td>
<td>0.11</td>
<td>0.7444</td>
</tr>
<tr>
<td>technique:size</td>
<td>1</td>
<td>3.25</td>
<td>3.25</td>
<td>6.19</td>
<td>0.0154</td>
</tr>
<tr>
<td>residuals</td>
<td>64</td>
<td>33.58</td>
<td>0.52</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.4.2 Duration

To compare the difference in duration in low-level requirements analysis for both DSDM projects with facilitated workshops and Method/1 projects that did not use facilitated workshops the date of the earliest activity in the low-level requirements engineering phase has been subtracted from the date on which the last activity in the low-level requirements engineering phase took place. This yields the duration of the low-level requirements engineering phase in days.

Examining the scatter plots in Figs. 4.3(a) and 4.3(b) reveals that no significant differences occur between the duration of projects that used facilitated workshops and those that did not.

An ANOVA analysis of the data (see Table 4.2) confirms that the factor technique does not have a significant influence on the duration of requirements engineering and the interaction between requirements engineering technique and size (as we did see for effort in Sect. 4.4.1) is also absent. As the dependent variable duration per function point is somewhat left-skewed, the variable has been transformed with an square root transformation before it has been used in the ANOVA analysis. An ANOVA analysis on the non-transformed variable yields similar results.

<table>
<thead>
<tr>
<th></th>
<th>df</th>
<th>SSE</th>
<th>MSE</th>
<th>F</th>
<th>p(F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>technique</td>
<td>1</td>
<td>0.37</td>
<td>0.37</td>
<td>3.55</td>
<td>0.0645</td>
</tr>
<tr>
<td>technique:size</td>
<td>1</td>
<td>0.07</td>
<td>0.07</td>
<td>0.65</td>
<td>0.4216</td>
</tr>
<tr>
<td>residuals</td>
<td>60</td>
<td>6.19</td>
<td>0.10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.4.3 Satisfaction

To determine if the requirements engineering technique (one-on-one interviews or facilitated workshops) influences the satisfaction of either the customer, project manager or IT employee, information from the project evaluation database has been collected for the projects that have also been used in the above analyses.

The evaluation database contains both quantitative and qualitative information about the perceived success of the project, from the viewpoint of each of the stakeholders. At the end of a project, the project manager and the customer are allowed to rate, amongst others, their satisfaction about the results of the project and about the requirements clarity. IT employees rate their satisfaction about the requirements clarity only. The set of questions asked and the categories of persons concerned are listed in Table 4.3. Unfortunately not for all projects an evaluation has been conducted and in some of the projects not the full set of questions has been answered.

The results of the MANOVA analysis of the (combined) set of questions relating to satisfaction are given in Table 4.5. In the same vein, the results with respect to requirements clarity are listed in Table 4.4.
Table 4.3: Project evaluation questions

<table>
<thead>
<tr>
<th>Question</th>
<th>Project manager</th>
<th>Customer</th>
<th>IT personnel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1: Functionality is delivered according to agreements</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Q2: The delivered results are according to my expectations</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Q3: The delivered results meet the quality requirements as agreed beforehand</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Q4: The requirements were clear</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

It appears stakeholders are more satisfied about the results of Method/1 projects than they are about the results of DSDM projects. This is contrary to what we expected. A closer inspection of the individual questions as listed in Table 4.3 gives a potential explanation. For both Q1 and Q3, Method/1 projects score higher than DSDM projects, and the project leader gives higher rates than the customer. For question Q2, the project leader and customer rate Method/1 projects equally high. However, project leaders rate DSDM projects (slightly) higher than Method/1 projects on this question, while customers rate DSDM projects quite a bit lower on this question. We hypothesise the following:

- project leaders do not expect too much from yet another method imposed by senior management. They’ve seen it all before.
- customers on the other hand do expect a positive effect from their involvement in the workshops.
- the net effect of both quality and functionality of DSDM projects is perceived to be a bit lower than for Method/1 projects.
- the mismatch between expectations and actual achievements results in lower scores for Q2, especially so for the customer.

Table 4.4 clearly reveals that different stakeholders do not rate requirements clarity differently for Method/1 and DSDM projects.

Table 4.4: ANOVA analysis of satisfaction about requirements clarity

<table>
<thead>
<tr>
<th></th>
<th>df</th>
<th>SSE</th>
<th>MSE</th>
<th>F</th>
<th>p(F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>viewpoint</td>
<td>2</td>
<td>3.76</td>
<td>1.88</td>
<td>1.83</td>
<td>0.1659</td>
</tr>
<tr>
<td>technique</td>
<td>1</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
<td>0.8902</td>
</tr>
<tr>
<td>viewpoint:technique</td>
<td>2</td>
<td>1.51</td>
<td>0.75</td>
<td>0.74</td>
<td>0.4813</td>
</tr>
<tr>
<td>residuals</td>
<td>94</td>
<td>96.52</td>
<td>1.03</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 4.5: MANOVA analysis of stakeholder satisfaction regarding project results

<table>
<thead>
<tr>
<th></th>
<th>df</th>
<th>Pillai Trace</th>
<th>approx. F</th>
<th>num df</th>
<th>den df</th>
<th>p(F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>viewpoint</td>
<td>1</td>
<td>0.19</td>
<td>4.18</td>
<td>3</td>
<td>53</td>
<td>0.0099</td>
</tr>
<tr>
<td>technique</td>
<td>1</td>
<td>0.07</td>
<td>1.30</td>
<td>3</td>
<td>53</td>
<td>0.2847</td>
</tr>
<tr>
<td>viewpoint:technique</td>
<td>1</td>
<td>0.32</td>
<td>8.30</td>
<td>3</td>
<td>53</td>
<td>0.0001</td>
</tr>
<tr>
<td>residuals</td>
<td>55</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.4.4 Threats to Validity

In this section we will discuss the potential threats to the validity of our study. For this discussing, we use the framework presented by Cook & Campbell [Cook and Campbell, 1979, p. 37–39] which distinguishes four categories of validity: internal validity, statistical validity, external validity and construct validity. As the statistical validity of the study has been discussed in Sects. 4.4.1, 4.4.2 and 4.4.3 this issue will not be repeated over here.

Threats to the internal validity of a study are the uncontrolled nuisance or background factors that could have an effect on the outcomes of the study and therefore present an alternative explication to the observed effect. In our study the internal validity of the study can be jeopardised by factors that influence the the required effort or duration of the project or that influence the satisfaction of the employees.

The amount of experience in the requirements gathering technique (facilitated workshops or one-on-one interviews) might be a confounding factor, as the IT personnel had more experience in one-on-one interviews than they had in facilitated workshops. To minimise this effect the organisation provided a three day course to all IT personnel to learn to use the new project methodology and facilitated workshops. Besides training, all new projects have been supported by consultants that had extensive prior experience using facilitated workshops. Although this might not have eliminated the total impact of difference in experience using a certain requirements gathering technique, these measures did reduce their impact.

To prevent negative effects on project effort, duration or satisfaction caused by incapable or inexperienced workshop facilitators, the organisation created a pool of workshop facilitators. Before employees were allowed to enter the facilitators pool, they were either to possess previous facilitation experience or to have followed a standard course for workshop facilitators [CGEY, 2002]. This reduced the effect of the facilitator’s experience on the outcome of the facilitated workshop.

After a small number of trial projects, which have been excluded from this study, the organisation has implemented the DSDM methodology in a department by department way. The project employees could not choose to use either DSDM and facilitated workshops or Method/1 and one-on-one interviews. As soon as a department implemented DSDM all new projects were to use DSDM. This measure makes it unlikely that motivation of project staff, the suitability of facilitated workshops for the project or other project characteristics had a significant influence on the relation between requirements gathering technique and project outcomes (in terms of required effort, duration and satisfaction).
Threats to the external validity of a study reduce the generalisability of the study’s outcomes to domains outside the scope of study. This study has been performed in just a single organisation. The organisation develops applications in a very wide range of programming languages, platforms and operating systems. As different programming languages, platforms and operating systems have been used during the study, the results should be applicable to different technological environments.

The organisation is however a very large internal IT department that is part of an even larger overall organisation. Both the IT department and the surrounding business environment can be characterised as being a professional bureaucracies. Further studies are needed to determine if the effects of the requirements gathering technique found in this study will also be found in medium to small IT departments, in less bureaucratic environments and in companies for whom IT is not a service but a core business.

A last threat to the validly of the study is posed by the question whether efficiency is the right measure to determine the benefits of a certain requirements gathering technique. This issue touches the construct validity of the study. One could argue that a good requirements gathering technique should not strive for efficiency, but instead should strive for effectiveness. Instead of eliciting the most requirements per unit of effort, one should perhaps aim to elicit the most requirements of high value or high quality per unit of effort.

Unfortunately it is extremely hard to measure the quality of a requirement. To judge the value or quality of a requirement one would need to measure the benefits of a system and the evolution of that system during its use. This would require an extensive, longitudinal study of system development projects.

4.5 Conclusion

Care must be taken not to draw too bold conclusions. The data analysis shows that neither of the two requirements engineering techniques is more effective in every situation. The significant interaction effect between the requirements engineering technique used and the size of the software project indicates that for large projects facilitated workshops offer greater productivity, whereas for smaller projects the one-on-one requirements engineering interviews appear to be more productive. This need not come as a big surprise. Facilitated workshops incur some overhead (employing a facilitator and organising the workshops) that do not outweigh their benefits for small projects.

One reason for not finding significant positive effects may be due to the fact that we only have data on the short-term effects of facilitated workshops, as perceived by various stakeholders. The effect of getting a better hold of the real requirements only shows after, and sometimes long after, the system has been made operational. It is only then that we see a plethora of change requests for ill-conceived systems. Next to that, facilitated workshops may have additional benefits that we did not measure, such as an improved mutual understanding between stakeholders. This is similar to what can be observed for other participatory activities, such as design reviews.

We therefore advise other researchers that plan to investigate the benefits of different requirements engineering techniques to take at least the following factors into account:

- the number of stakeholders involved.
- the number of requirements changed during requirements engineering.
• the number of conflicting requirements.
Chapter 5

Exploring Information in Project Evaluation Databases

In the previous two chapters we have limited the analysis to available quantitative information. In this chapter we demonstrate how to analyse sources of data that are of qualitative nature. By not limiting ourselves to quantitative data sources, we more information at hand to monitor and guide a software process improvement programs.

An example of qualitative data about software projects are post-mortem project reviews. Post-mortem reviews are mostly recorded in plain text. This makes it difficult to derive useful overall findings from a set of such post-mortem reviews.

We have developed a five-step method to transform the qualitative, natural language type information present in those reports into quantitative information. This quantitative information can be analyzed statistically and related to other types of quantitative project-specific information. In this chapter we discuss the method, and show the results of applying it to two case studies.

This chapter is partly based on a paper presented earlier at the 11th European Software Process Improvement Conference (EuroSPI 2004) [Schalken et al., 2004], of which an extended version appeared in the journal Software Process: Improvement and Practice [Schalken et al., 2006a]. The second case study has been performed in cooperation with Torgeir Dingsøyr, Nils Brede Moe and Tor Stålhane (from SINTEF ICT and the Norwegian University of Science and Technology – NTNU) and has been accepted for publication at the 14th European Software Process Improvement Conference (EuroSPI 2007) [Dingsøyr et al., 2007].

5.1 Introduction

A central issue in knowledge management and software process improvement is to learn from experience. In software engineering, most experience is gathered in projects, which makes project experience a prime source for learning. This is why most software project management methods recommend that projects be evaluated. Experience from completed projects can be collected through postmortem reviews [Dingsøyr, 2005] or project retrospectives [Kerth, 2001]. Many companies conduct postmortem reviews, but we have
found few companies that analyze the outcome of several reviews to facilitate learning on an organizational level.

Although the advice to perform project evaluations is sound, most project management methods offer no guidance as to how to analyze the data collected. This study fills this gap and describes an exploratory data analysis method to extract useful information from qualitative post-mortem project evaluation reports using an approach that is based on Grounded Theory [Glaser and Strauss, 1967, Strauss and Corbin, 1990].

A common factor in knowledge management and software process improvement is to learn from past successes and failures in order to improve future software development. Experience Factory [Basili et al., 1994a] has been a central term in focusing organizational learning on improving software development processes.

Two kinds of project evaluations can be distinguished [Jurison, 1999]: intermediate project evaluations that take place periodically during the course of the project, and post-mortem project evaluations that take place at the end of a project, when the actual task of the project has been finished.

The objectives of these evaluations differ. Intermediate evaluations are used by senior management to periodically assess whether the project’s goals and objectives are still relevant [Jurison, 1999] and to monitor project risks [McConnell, 1996b]. Post-mortem project evaluations on the other hand have the objective “to evaluate past experience and develop lessons learned for the benefit of future projects” [Jurison, 1999]. The context of this study is project post-mortem reviews as a means to develop lessons learned for future projects.

The insights gained in the course of the project are made explicit and are recorded in the post-mortem project evaluation report. This evaluation can be useful to the people that have been directly involved with the project, but also to the organization at large. People directly involved in the project may gain an understanding of the factors that attributed to and/or undermined the project’s success. One of the strengths of post-mortem project evaluations is that they force people to reflect on their past work and at the same time the evaluations also provide feedback from other team members. These lessons learned can also be used outside the group of people directly involved in the project, to reuse improvements and to avoid pitfalls in future projects throughout the organization. There are few empirical studies addressing the organization at large [Dingsøyr and Conradi, 2002].

Garvin defines a learning organization as “an organization skilled at creating, acquiring, and transferring knowledge, and at modifying its behaviour to reflect new knowledge and insight” [Garvin, 1993] Huber gives advice on what managers can do to make their organizations more “learning” [Huber, 1991]:

- Learn from experience - systematically capture, store, interpret and distribute relevant experience gathered from projects; and also to investigate new ideas by carrying out experiments.

- Use a computer-based organizational memory - to capture knowledge obtained from experts to spread it throughout the organization.

One way to collect experience from projects is to perform postmortem reviews [Birk et al., 2002, Dingsøyr, 2005]. By a postmortem review, we mean a collective learning activity which can be organized for projects either when they end a phase or are terminated. The main motivation is to reflect on what happened in the project in order to
5.1. Introduction

improve future practice – for the individuals that have participated in the project and for the organization as a whole. The tangible outcome of a meeting is a postmortem report.

Researchers in organizational learning sometimes use the term “reflective practice”, which can be defined as “the practice of periodically stepping back to ponder on the meaning to self and others in one’s immediate environment about what has recently transpired. It illuminates what has been experienced by both self and others, providing a basis for future action” [Raelin, 2001]. This involves uncovering and making explicit the results of planning, observation and achieved practice. It can lead to understanding of experiences that have been overlooked in practice.

Before insights from project evaluations can be used by the rest of the organization, they first need to be packaged for reuse [Seaman, 1993]. For the organization as a whole post-mortem project reports usually contain too much project specific details. Therefore, the information in post-mortem project reports needs to be consolidated before it can be quickly understood and used throughout the rest of the organization. The consolidation of project evaluations is usually based on software metrics, since software metrics allow for easy consolidation.

There is a difference between the body of knowledge regarding the elicitation of knowledge during a project postmortem review and the body regarding the analysis of postmortem reviews to package this knowledge for the rest of the organization. An overview of the elicitation process, in which tacit knowledge is externalized into explicit knowledge [Nonaka and Takeuchi, 1995] in project postmortem reviews is given in [Dingsøyr, 2005]. The packaging of this tacit knowledge in project postmortem reviews, so that it can be internalized by employees in other parts of the organization, is the topic of this study.

Software metrics, both objective and subjective, can be used to evaluate projects. The advantage of objective metrics is that they do not require the judgment of an expert. However, their formal definitions usually require strict adherence to a measurement procedure and frequently require a lot of data to be collected and aggregated in order to measure the attribute. Without the required data no measurement for the attribute is possible, which explains why for many projects not all potentially useful objective metrics are available. Subjective measures on the other hand require no strict adherence to measurement rules, the judgment of an expert suffices. This explains the higher availability of subjective measurements of projects [Wohlin and Andrews, 2001].

Even when resorting to subjective measurements for project attributes the distillation of knowledge from past experience is not easy. Without careful up-front [McIver and Carmines, 1981] design of the subjective metrics, chances are that the scales of the subjective measurements are meaningless. On top of that potentially interesting project attributes are often missing from the metrics database. This leaves the analyst with missing data and measurements that are meaningless.

To solve the stated problems we propose a new method to explore potentially interesting relations present in project evaluations. Instead of limiting ourselves to just the project metrics database we propose to use the natural language post-mortem project reports as an additional source of data. Using concept hierarchy trees we are able to recode the qualitative information in the post-mortem project reports into quantitative information. This quantitative information can be analyzed statistically to discover correlations between factors. This proposed method has been tested extensively in a case study involving 55 projects at the internal IT department of the ABN AMRO Bank, followed
up by a reduced case study at a small company that produces embedded systems for satellites and receivers.

The remainder of this chapter is structured as follows. In Sect. 5.2 we discuss some related work in this area, and in Sect. 5.3 we present our analysis method. Sections 5.4 and 5.5 contain the case studies in which we applied the method to a substantial set of real-world post-mortem review reports. We end with our conclusions in Sect. 5.6.

5.2 Related Work

Most work on organizational learning or knowledge management in software engineering address technical systems for distributing experience in an organization [Dingsøyr and Conradi, 2002]. There is little work on the effects of gathering experience on software development issues over time.

Wohlin and Andrews [Wohlin and Andrews, 2001] have developed a method to evaluate development projects using subjective metrics about the characteristics of a project, collected using a questionnaire that can even be conducted at the end of the project. They have created a predictive model for certain success indicators, based on subjective metrics of project factors. Our work differs from their work in that our approach does not even require the collection of subjective measurements at the end of the project. Instead, we extract subjective metrics from qualitative data as found in post-mortem review reports.

As our method places lower demands on the required information, the method proposed in this chapter might be applicable to an even wider range of projects than Wohlin and Andrews’ method. On the other hand, as our data has less structure, our method results in a large percentage of missing data, which limits the analysis that can be performed on the data (regression model building and principal component analysis are not feasible when our method is used).

Damele et al. have developed a method to investigate the root causes of failures during development [Damele et al., 1996]. Their method uses questionnaires to obtain quantitative information on failures, which are analyzed using correlation analysis and Ishikawa diagrams. Their method differs from ours in that it uses Ishikawa diagrams to present the results of the analysis and we use the diagrams similar to Ishikawa diagrams as an intermediate structuring technique.

In [van der Raadt et al., 2004] we used Grounded Theory to interpret and analyze data from a large set of semi-structured interviews with practitioners in the area of software architecture. The Grounded Theory method is a qualitative approach to inductively distill theory from a dataset [Glaser and Strauss, 1967] [Strauss and Corbin, 1990]. This approach is not meant to test an existing hypothesis, but provides a method for emerging a theory from collected data. The basic idea of Grounded Theory is to read and reread some textual database, and iteratively ’discover’ and label a set of concepts and their interrelationships. In the research described in this chapter, we apply a method related to Grounded Theory when populating the concept hierarchy trees.

5.3 Method

Before we can describe the analysis method and insights the method attempts to spot, we first need to introduce some definitions. These definitions follow the definitions given by
Table 5.1: Analysis process to discover relations in post-mortem reports.

<table>
<thead>
<tr>
<th>Process steps</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Identify success factors</strong></td>
<td>Identify the factors that determine the success of a project in the eyes of the stakeholders.</td>
</tr>
<tr>
<td><strong>Select project evaluations</strong></td>
<td>Select project evaluations for further analysis. To obtain meaningful results, one might select projects with extreme values along certain success factors.</td>
</tr>
<tr>
<td><strong>Identify project factors</strong></td>
<td>Identify repeating patterns in project factors by screening a subset of the selected projects (from step Select project evaluations). These repeating patterns will be structured using a concept hierarchy tree.</td>
</tr>
<tr>
<td><strong>Interpret project evaluations</strong></td>
<td>Read and interpret all project evaluations (from step Identify project factors) using the concept hierarchy tree created in the previous step. After the interpretation, the project is evaluated on the project factors that are present in the concept hierarchy tree.</td>
</tr>
<tr>
<td><strong>Analyze correlations</strong></td>
<td>Analyze the correlations between project factors and success factors. Sort through the correlations between the project factors to find interesting results.</td>
</tr>
</tbody>
</table>

Wohlin and Andrews [Wohlin and Andrews, 2001]. A factor is a general term for project aspects we would like to study. Factors are either project factors or success factors. Project factors describe the status, quality, or certain characteristics of a project (e.g. as the testing tool used and team morale), and their value can either be determined prior to starting the project or during the execution of a project. Success factors capture an aspect of the outcome of a project (e.g. the timeliness of a project).

The method described below attempts to expose the effects of project factors on the success factors of a project. The method could try, for example, to discover the effect of using a certain programming language (which is a project factor) on the productivity of those projects (which is a success factor). Our method for discovering insights in natural language post-mortem evaluations consists of the steps listed in Table 5.1.

### 5.3.1 Identify Success Factors

To determine whether a project is a success or a failure, one needs to know what the important aspects (or factors) are of a project in the eyes of the stakeholders. Examples of success factors are timeliness, productivity and stability. Success factors can both be measured objectively and subjectively. The success factors we used in our case study are listed in Table 5.3.

In our study we have assumed that quantitative data is available for the key success factors metrics that have been defined. If the organization does not have quantitative data on the key success factors, one can let experts rank the projects based on the success factor (effectively measuring the success factor on an ordinal scale). If no consensus exists on the key success factors, one could start a separate analysis phase using step 3
and 4 to obtain quantitative assessments of the outcomes of a project.

5.3.2 Selection of Project Evaluations

Within the scope of an analysis, it usually will not be feasible to analyze all projects of which a post-mortem project evaluation is available. Therefore a sample of the available project evaluations needs to be drawn. Both random sampling and theoretical sampling can be used to select a sample of project evaluation reports. With random sampling chance determines if a postmortem report is included in the sample. With theoretical sampling a report is not drawn blindly based on chance, but instead those reports are selected that offer the greatest opportunity of discovering relevant project factors. Although theoretical sampling has its own disadvantages, we prefer to use theoretical sampling over random sampling, as it increases our chances of finding noteworthy results.

As we use up-front theoretical sampling of project postmortem reviews, we need to have a method to select promising postmortem reviews. To achieve this, we first stratify the projects based on success factors identified in the previous step. The stratification process selects a proportion of projects that score high or low on the success factor and another proportion of projects that score average on the success factor. The stratification selects a disproportionately large group of projects that are an extreme case for one of the selected success factors, as these projects yield most information.

In the stratification process projects with the most extreme values on certain dimensions are found by selecting those projects that deviate more than $X$ standard deviations from the average for that dimension. $X$ is chosen such that a manageable number of projects are selected.

The stratification should in this case not lead to different conclusions than an analysis of a random sample, since stratification does not disrupt the coefficients in a regression equation model as long as the other assumptions of the regression model are not violated [Allison, 2002].

5.3.3 Identify Project Factors

The open questions in project evaluations do not have numerical answers. We thus have to look for a transformation from the natural language texts available to numerical scores on project factors, which can subsequently be analyzed. We use concept hierarchy trees to bring structure in the remarks in the project evaluations to find the underlying project factors in the answers to the open questions. The project factors that are identified as relevant in this step, will be used in the next analysis step to code all selected project evaluations.

Concept hierarchy trees, which have a great resemblance with fish-bone or Ishikawa diagrams [Ishikawa, 1984], organize individual remarks made in project postmortem reviews. The difference between Ishikawa diagrams and concept hierarchy trees is that Ishikawa diagrams focus on the effect of project factors on a single specified success factor, whereas concept hierarchy trees merely organize project factors that have an influence on any of the identified success factors. Concept hierarchy trees do not indicate what effect the project factors have on the success factor. Remarks that identify similar project factors are located near each other in the tree, whereas remarks that refer to different project factors are placed in separate positions in the concept tree. To synthesize
the concept hierarchy tree, we take a sample (say, 30%) from the set of selected project evaluations. The content of these selected project postmortem reviews is used to populate the concept hierarchy tree. To synthesize the concept hierarchy tree it is usually not required to examine all the project evaluations that will be analyzed in the subsequent analysis step, as we expect relevant project factors to occur frequently.

To synthesize a concept hierarchy tree based on a set of selected project postmortem reviews, one starts by breaking the review text into lines or paragraphs each describing a single project factor that influences project outcomes. In our case study review, the text originates from the open questions listed in Table 5.2. The mapping of lines to project factors is not always one-on-one, sometimes multiple lines describe a single project factor, sometimes a single sentence can describe multiple project factors. This process of breaking the text into single topic remarks is called open coding in Grounded Theory. We should be careful not to include descriptions of success factors in the concept hierarchy tree, as we do not want to confuse cause and effect.

The result of breaking the postmortem review up into single topic remarks is a large list of descriptions of project factors. This list likely contains duplicate entries for a single project factor and contains both overly broad remarks and very project specific remarks regarding the project factors. To eliminate duplicates in the list of project factors and to obtain a list of project factors of a roughly similar level of abstraction, the individual remarks need to be organized. To do so, the individual remarks are placed into a concept hierarchy tree in such a way that similar remarks end up near one another (share more ancestors in the concept hierarchy tree). This organization of remarks by comparing individual remarks and grouping similar remarks together is called comparative analysis and the whole process of hierarchically ordering remarks is called axial coding in Grounded Theory.

To facilitate the ordering of the remarks in our case study we started with a predetermined set of four main categories: processes, tools, people, and environment, see Fig. 5.1. Our top-level structure was derived from a classification from Balanced Score Card [Kaplan and Norton, 1996] schemes used elsewhere within the company. If no such structure is available up front, it can be obtained as a byproduct of a Grounded Theory-like investigation. Researchers who are familiar with Grounded Theory may have noted that our main classification of remarks into project factors and success factors and our

![Figure 5.1: Example of concept hierarchy tree containing project characteristics.](image-url)
subsequent classification of remarks in four main categories differs from the paradigm model [Strauss and Corbin, 1990, p. 99] suggested by Strauss and Corbin. The rationale for this difference is that our classification scheme is specialized for software engineering research and that the absence of intermediate constructs (such as causal conditions that cause a phenomenon) makes the statistical analysis of the gathered data easier. Models including intermediate constructs require complex factor analytical analyses, whereas with our approach correlation coefficients suffice.

After all remarks have been placed in the concept hierarchy tree, nodes of conceptually similar remarks are replaced by a node containing a keyword or a short statement that describes the underlying project factor. To avoid ending up with overly specific or overly broad concepts we have used the following rule of thumb: each keyword must be observed in at least 20% and at most 50% of the project postmortem reviews under study. Overly specific or overly broad concepts can hamper the subsequent analysis steps as we will only be able to determine the effect of the project factor on the success factors if the presence/absence of project factors varies. Only discriminating concepts are interesting as we can’t learn much from the observation that, say, “user involvement is a crucial factor” if this remark is made for almost all projects.

The keywords from the adjusted concept hierarchy tree describe the answers to the open questions in a project postmortem review. Since these keywords classify the answers to the questions, and not the questions themselves, one should not expect a one-to-one relation between the open questions in the postmortem report and the keywords in the concept hierarchy tree. For reasons of space (we distinguished over 80 categories in the case study), the resulting list of categories is not included in the text. Figure 5.1 contains a few typical sub categories (such as: test tooling used) and subsubcategories (such as: cooperation in IT team) we found during the case study.

5.3.4 Interpret Project Evaluations

After the keywords and patterns have been distilled from the project evaluations, the subjective interpretation can start. During the subjective interpretation step, selected project evaluations are read by the researcher. This researcher interprets in which categories from the concept hierarchy tree the remarks from the evaluation fit. Next, it is determined whether the project scores positive or negative on this category. For example, one of the categories we distinguished is change management. A project evaluation report may for instance contain phrases that pertain to the notion of change management. From these phrases, the researcher may deduce that change management has been implemented well for the project, or that problems with respect to change management occurred during the project. Using Likert scaling, we transform natural language text in the answers to open questions into numerical information. For each project, the scores of each category are included in a spreadsheet, together with information from other sources on project aspects. This leaves us with a concise numerical characterization of each project.

5.3.5 Analyze Correlations

The interpretation of the post-mortem project evaluations yields a spreadsheet with a quantitative encoding of the information in the project evaluation database. As said above, this spreadsheet may be coupled with other numerical information from the project
administration database. Using a statistical package, we may next determine correlations between the project evaluations on the one hand, and other quantitative information on the other hand. In our case study, we had quantitative information on productivity, conformance to budget and schedule, and satisfaction of different sets of stakeholders.

The matrix that results from the subjective interpretation of the project evaluations unfortunately has a high number of variables in relation to the number of observations. Normally we would use a technique such as principal component analysis to reduce the number of variables. However in our case a large percentage of the data is missing, which makes principal component analysis infeasible.

Instead of first reducing the number of variables in the data matrix, we directly measure the correlation between the project factors (the independent variables) and the success factors (the dependent variables). This leads to a matrix of correlations between project characteristics and success indicators. We used Kendall’s $\tau_B$ [Liebetrau, 1983] measure for the correlation, since this measure is suited for ordinal data. We use pair-wise deletion when encountering missing data, instead of list-wise deletion [Allison, 2002], to make optimal use of the available data.

The correlation coefficients in the matrix are not all based on the same number of observations. Correlation coefficients that are based on a larger number of observations offer more certainty that the observed correlation is really present in the underlying population. This certainty is shown in the level of significance of the correlation coefficient, which can be calculated by statistical packages [Siegel, 1956].

Rather than examining the correlation coefficients for all identified project factors, we may opt to restrict ourselves to observing the most significant coefficients. The sheer amount of numbers in the correlation matrix can distract attention away from the most influential project factors. Several selection criteria can be used to reduce the size of the correlation matrix.

The most straightforward selection criterion for rows in the correlation matrix is to select only those rows that contain a single correlation coefficient that has the highest significance level. An alternative to this criterion is to look for rows that contain the highest or lowest single correlation coefficient and not to look at the significance of the coefficient. More advanced selection criteria can look at the overall effect of a project factor on all success factors. To achieve this one can either select those rows that have the maximum squared average correlation coefficient or that have the highest average significance level for the correlation coefficients. Even more sophisticated criteria can take weight factors for each success factor into account to average the overall correlation coefficients or significance levels.

Note that the statistical significance observed in this type of analysis often is not very high, due to the small sample sizes. As we make multiple comparisons we should apply a Bonferroni or Sidak correction to compensate for the multiple comparisons if we want to use the technique as a confirmatory instead of an exploratory technique. As the statistical significance of the results is rather low, we need to have a theory for the correlations observed, in order to derive useful information. Correlation by itself does not imply the direction of the causality.
5.4 Case Study: ABN AMRO Bank

5.4.1 Context of Case Study

The first case study has been performed within an internal Information Technology department of the ABN AMRO Bank N.V. (a large financial institution). For more background information about the organisation, please refer to Sect. 1.2.

Bottlenecks

The organization has developed its own post-mortem project evaluation method. The evaluation method consists of an online administered questionnaire composed of both open and closed questions. In the evaluation process three groups of stakeholders are addressed: the customer who has commissioned the project, the IT personnel that participated in the project and the involved middle management of the IT department.

Although the organization has invested large amounts of time in both developing the original evaluation method and evaluating the projects themselves, the organization was unfortunately not able to fully benefit from the lessons learned that lie hidden in the more than 600 evaluation reports that are stored in the project evaluation database.

The organization used the complete information in the project evaluation only as feedback to the project leader responsible for the project. The organization outside the project used the project evaluations only as Balanced Score Card indicators of satisfaction of the commissioning customer and the satisfaction of the IT employees. These overall, consolidated satisfaction ratings made it hard to pinpoint what is going wrong when e.g. the employee satisfaction drops.

The inability to use the project evaluation database can be attributed to four major reasons:

- The database system containing the evaluation database was not available electronically. Manual copying of the data from the database to a statistical tools was required. This made analysis a labour-intensive task.

- As the evaluation method included open questions, some of the answers contained textual answers instead of quantitative data. Textual data is inherently harder to analyze than quantitative data.

- The wording and grouping of the closed questions was inadequate. The grouping of the questions, that was derived from the Balanced Score Card items, was such that many of the categories measured simultaneously different concepts, which makes the interpretation of the average on a category infeasible.

- As the individual answers on closed questions contribute to the average customer, employee or management satisfaction, one cannot state that the scores on individual questions are independent from the satisfaction measurements. These scale-subsample dependencies make the interpretation of correlation coefficients difficult at least.

The low quality of the closed questions and their clustering combined with the problem of scale-scale correlations led to the decision to extract project characteristics from the answers to the open questions, using the method outlined in the previous section.
Table 5.2: Case study ABN AMRO Bank: Open questions in project evaluation questionnaire.

<table>
<thead>
<tr>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>• What are the 3 most positive points of the project?</td>
</tr>
<tr>
<td>Explain them.</td>
</tr>
<tr>
<td>• What are the 3 most important learning points of the project?</td>
</tr>
<tr>
<td>Explain them.</td>
</tr>
<tr>
<td>• Can you give 3 suggestions by which the project could have been carried</td>
</tr>
<tr>
<td>out (even) better?</td>
</tr>
<tr>
<td>• Was there sufficient input documentation at the beginning of the</td>
</tr>
<tr>
<td>functional design phase?</td>
</tr>
<tr>
<td>Which inputs were not available? Indicate the reasons.</td>
</tr>
<tr>
<td>• Which testing tools were used?</td>
</tr>
<tr>
<td>What were the advantages and disadvantages?</td>
</tr>
<tr>
<td>• Was test ware available?</td>
</tr>
<tr>
<td>If not, what were the reasons?</td>
</tr>
<tr>
<td>• Which configuration management tools were used?</td>
</tr>
<tr>
<td>What were the advantages and disadvantages?</td>
</tr>
</tbody>
</table>

Analyzing the open questions had as an added advantage that the answers from every respondent were available, which gave insight into the degree to which the stakeholders in the project agreed on certain issues.

5.4.2 Data Collection

Evaluation Reports

At the end of each development project a mandatory evaluation cycle is initiated by the IT project office of the organization. Upon request of the project office the project leader invites involved stakeholders by e-mail to fill out the evaluation questionnaire. When a sufficient number of stakeholders has filled out the questionnaire, the project leader activates an evaluation consolidation routine in the evaluation program, which anonymises the responses and calculates averages of all the closed questions.

The evaluation questionnaire contains both open and closed questions. The open questions in the questionnaire are listed in Table 5.2. As there are over 150 closed questions in the questionnaire, only the categories used to group the closed questions are included in this study. The categories of the closed questions are: Time Management, Risk Management, Project results, Project Task (Work), Organization, Work environment, Quality/scope, Project management, and Information, Project. The categories of the closed questions originate from a Balanced Score Card [Kaplan and Norton, 1996] initiative that has been conducted at the organization.
Selection Criteria

For the analysis of the project information database 55 project evaluations have been selected out of a database containing over 600 project evaluations. The selection of projects included ‘normal projects’, projects with a specific programming environment, and projects that deviated on: productivity, conformance to budget or conformance to schedule. For the deviant projects, a selection of projects with an equal ratio of over-performing and under-performing projects has been made. The exact distribution of the number of projects on the selection criteria is given in Table 5.3.

5.4.3 Results

The result of the analysis steps performed in the case study can be found in Table 5.4. The table contains both the Kendall’s $\tau_\beta$ correlation coefficients between project factor and success factor, as well as the $p$-values of those correlation coefficients. The correlation coefficients indicate if there is a strong positive (1), or negative (-1) relation between the factors, or no relation (0). The $p$-value indicates the strength of the evidence of the relation between the factors, varying from 0 (indicating very strong evidence) to 1 (indicating very weak evidence).

To reduce the correlation matrix we have sorted the factors with respect to the highest overall level of significance. Having sorted the project factors we selected the top 20% from this list.

5.4.4 Discussion

The analysis has given us a lot of insight into the quality and organization of the set of closed questions used so far, and suggested a number of additional closed questions to ask. For each of the project factors that was in the selected top 20% of most relevant project factors a question has been added to the updated project evaluation questionnaire, so that, for future evaluations, more quantitative information will be directly available.

At a concrete level, the study showed some interesting, albeit weak, relations between
project characteristics and success indicators. For example, high productivity occurs frequently when there is a good cooperation within the team, and when the infrastructure architecture is elaborated well. One remarkable conclusion from the matrix is that all project management activities (i.e. creating a schedule, planning the activities and checking the progress of a project against its planning) seem to have a negative effect on the productivity of a project. At the same time project management activities do have the expected positive effect on keeping a project on schedule and within budget.

The relations that have been discovered by analysing the project post-mortem database need further study though, as the relations are weak and (because many comparisons were made) not statistically significant.

In the next chapter we will examine the conclusion validity of the results, because we are interested to see if the relations, which we have found during the study, are stable relations that are representative for the entire organisation.

A second validity issue with this study has to do with the content validity of the results. In research that categorises concepts using Grounded Theory, the validity of the end results is conditional on the validity of the axial coding phase. The validity of the outcomes of the axial coding is hard to assess. One way to assess the validity of this phase would be to perform the axial coding phase twice, each time by a different researcher. The resulting concept trees from the axial coding phase could be compared for similarity. Unfortunately no suitable concept tree comparison metrics are available to measure the similarity of concept trees.
Table 5.4: Case study: Results of the correlation analysis on the evaluation matrix.

<table>
<thead>
<tr>
<th>Factor name</th>
<th>Productivity</th>
<th>Budget</th>
<th>Schedule</th>
<th>Duration</th>
<th>Management</th>
<th>Employee</th>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conformance to</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>change management</td>
<td>0.05</td>
<td>-0.18</td>
<td>-0.20</td>
<td>0.20</td>
<td>0.33</td>
<td>0.51</td>
<td>0.40</td>
</tr>
<tr>
<td>project management</td>
<td>-0.13</td>
<td>0.12</td>
<td>0.39</td>
<td>-0.26</td>
<td>0.39</td>
<td>0.45</td>
<td>0.10</td>
</tr>
<tr>
<td>quality planning</td>
<td>-0.16</td>
<td>0.13</td>
<td>0.34</td>
<td>-0.20</td>
<td>0.43</td>
<td>0.27</td>
<td>0.10</td>
</tr>
<tr>
<td>quality schedule</td>
<td>-0.41</td>
<td>0.24</td>
<td>0.23</td>
<td>-0.06</td>
<td>0.10</td>
<td>0.29</td>
<td>-0.19</td>
</tr>
<tr>
<td>project control</td>
<td>-0.28</td>
<td>0.17</td>
<td>0.29</td>
<td>-0.34</td>
<td>0.08</td>
<td>0.39</td>
<td>-0.33</td>
</tr>
<tr>
<td>test-ware reused</td>
<td>-0.11</td>
<td>0.50</td>
<td>0.53</td>
<td>-0.38</td>
<td>-0.17</td>
<td>0.39</td>
<td>0.20</td>
</tr>
<tr>
<td>quality infrastructure architecture</td>
<td>0.52</td>
<td>0.37</td>
<td>-0.33</td>
<td>0.50</td>
<td>0.39</td>
<td>0.05</td>
<td>0.43</td>
</tr>
<tr>
<td>communication efficiency</td>
<td>0.01</td>
<td>-0.36</td>
<td>-0.26</td>
<td>0.30</td>
<td>0.16</td>
<td>0.33</td>
<td>0.25</td>
</tr>
<tr>
<td>cooperation</td>
<td>0.08</td>
<td>-0.20</td>
<td>0.25</td>
<td>-0.26</td>
<td>0.22</td>
<td>0.40</td>
<td>0.27</td>
</tr>
<tr>
<td>cooperation within IT</td>
<td>0.39</td>
<td>-0.13</td>
<td>0.46</td>
<td>-0.24</td>
<td>0.59</td>
<td>0.20</td>
<td>0.44</td>
</tr>
<tr>
<td>appropriateness team size</td>
<td>0.12</td>
<td>-0.51</td>
<td>-0.93</td>
<td>0.84</td>
<td>-0.14</td>
<td>-1.00</td>
<td>-1.00</td>
</tr>
<tr>
<td>team stability</td>
<td>-0.58</td>
<td>0.14</td>
<td>-1.00</td>
<td>0.50</td>
<td>-0.36</td>
<td>0.36</td>
<td>0.18</td>
</tr>
<tr>
<td>team stability organization</td>
<td>0.25</td>
<td>0.36</td>
<td>-0.13</td>
<td>-0.61</td>
<td>-0.27</td>
<td>-0.57</td>
<td></td>
</tr>
<tr>
<td>test-tool expediter used</td>
<td>-0.27</td>
<td>0.29</td>
<td>0.58</td>
<td>-0.34</td>
<td>-0.03</td>
<td>0.11</td>
<td>-0.11</td>
</tr>
</tbody>
</table>
5.5 Case Study: Kongsberg Spacetec

The second case study reported here is an exploratory case study [Yin, 2003] of twelve projects in Kongsberg Spacetec AS, a medium-size software company. In this organisation, we examined what we can learn from analyzing postmortem review reports of twelve projects in a medium-size software company. The company was selected because of prior cooperation in software process improvement projects.

5.5.1 Context of Case Study

Kongsberg Spacetec AS ("Spacetec") of Norway is one of the leading producers of receiving stations for data from meteorological and Earth observation satellites. Since the company was founded in 1984, its products have been delivered to a number of clients around the world, with a current export share of 85%. Spacetec has expertise in electronics, software development and applications. 80% of the 60 employees in the company have a master's degree in physics or computer science.

At the start in 1984 the main task of the company was engineering through customer specific projects, and the main customer was the European Space Agency. Because of this, the ESA PSS-05 software engineering standards were adopted. During the 1990s the market situation changed, and a new kind of customer became increasingly important. These customers were not interested in how the product was developed or how the quality assurance was performed. Instead of providing detailed requirements specifications they expected off-the-shelf products that could be delivered at short notice. This made it necessary to develop generic products through internally financed and managed projects. This new way of doing projects, and the fact that several of their big projects had problems, motivated Spacetec to focus on learning from experience. The company has conducted postmortem reviews since 2000.

5.5.2 Data Collection

Selection Criteria

The data used in this study are collected from twelve software development projects which were finished between 2000 and 2005 at Spacetec. The projects that are analyzed are not a random sample of the company’s projects, but projects singled out because they had cost overruns (8 to 155 percent), see Table 5.5.

During the qualitative analysis, we found out that the projects selected for postmortem review shared five main characteristics. All postmortem reports recorded negative experiences related to lack of knowledge, people effects, process effects, deliverables and management.

Data Sources

We have collected information from three data sources: postmortem review reports, a questionnaire-based evaluation of the projects and memos from a workshop with company participants where the projects were discussed. This workshop discussed findings from our analysis of the two first sources, but has only been used for validation purposes. We briefly describe the first two sources:
Table 5.5: Projects selected for postmortem review.

<table>
<thead>
<tr>
<th>No.</th>
<th>Overrun (%)</th>
<th>Project size</th>
<th>Duration</th>
<th>Report</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>99</td>
<td>large</td>
<td>3y</td>
<td>short</td>
</tr>
<tr>
<td>2</td>
<td>31</td>
<td>large</td>
<td>3y</td>
<td>short</td>
</tr>
<tr>
<td>3</td>
<td>155</td>
<td>large</td>
<td>0.5y</td>
<td>short</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>large</td>
<td>0.5y</td>
<td>long</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>large</td>
<td>1y</td>
<td>short</td>
</tr>
<tr>
<td>6</td>
<td>100</td>
<td>large</td>
<td>1y</td>
<td>short</td>
</tr>
<tr>
<td>7</td>
<td>114</td>
<td>medium</td>
<td>1y</td>
<td>short</td>
</tr>
<tr>
<td>8</td>
<td>85</td>
<td>large</td>
<td>1.5y</td>
<td>short</td>
</tr>
<tr>
<td>9</td>
<td>18</td>
<td>large</td>
<td>3y</td>
<td>long</td>
</tr>
<tr>
<td>10</td>
<td>23</td>
<td>large</td>
<td>2y</td>
<td>short</td>
</tr>
<tr>
<td>11</td>
<td>79</td>
<td>medium</td>
<td>1y</td>
<td>long</td>
</tr>
<tr>
<td>12</td>
<td>79</td>
<td>large</td>
<td>4y</td>
<td>short</td>
</tr>
</tbody>
</table>

Postmortem review reports

Three of the reports were written by researchers participating in carrying out the review, while nine were written by the company’s quality department. In Table 5.5, the project overrun is given as a percentage, size is either large (>5000 h) or medium (<5000, >1000 h), duration in years and we have indicated whether we have an extensive (long) or brief (short) postmortem report. The postmortems were carried out after project completion for 10 projects, and after finishing the main part of the project for the remaining two projects.

Three of the postmortem reports were long reports written by researchers (17-23 pages). The researchers used the following, postmortem review process [Birk et al., 2002]:

- Use the KJ [Scupin, 1997] process for brainstorming to identify what went well and what went wrong in the project.
- Root cause analyses [Straker, 1995] to identify the root causes for the most important reasons for success and for failures.
- Prioritize improvement actions based on the results from the root cause analysis.
- Write a postmortem report, summing up all important points. In addition, the meetings were taped and transcribed as part of the report.
- The report was reviewed by all participants and misunderstandings were corrected.

Nine reports were written by the company’s quality department. They wrote short reports (3-8 pages) and their process differed in that they:

- Only collected the negative experiences, because of the project sample and time limitation.
• Did not tape the meeting and later make a transcript.
• Did not circulate the postmortem report for commenting and to correct possible misunderstandings.

The postmortem process that collected the least information will decide which information we can later use in our analysis. Thus, we can for instance only use information on what went wrong for the projects in our analysis. When we bear in mind that we already have selected only projects with cost overruns, we see that we have a strong focus on the negative aspects of the company and their projects.

**Questionnaire-based evaluation**

This was sent to two members of the quality department as well as the person responsible for all software projects, and the person responsible for the software products. This was done in order to get an opinion on the project quality as perceived from these roles. We asked them to rank the projects according to the following factors: Strategic importance, Customer satisfaction, Software quality and Software productivity. In addition, we asked them to indicate what they thought was most important in the project: Quality, productivity or customer satisfaction. We also gathered information on the project size, duration and project cost overrun.

**5.5.3 Data Analysis**

To analyze the data from the postmortem reports, we chose to: 1. code the reports into a predefined set of project factor categories and 2. analyze the most occurring factors through a bottom-up qualitative analysis, inspired by grounded theory [Strauss and Corbin, 1998] . We describe these two steps in the following:

**Step 1:** To code the data from the postmortem reviews, we used a predefined framework inspired by McConnel [McConnell, 1996b], which covers most topics that are relevant in a postmortem review. We placed each topic from the review in the framework. The first two levels of the coding framework are given in Table 5.6:

We coded all negative project factors from the postmortem reviews by the categories listed in the coding framework (axial coding). Each review was coded by two researchers independently, and we discussed disagreements until we reached consensus on the coding.

This coding resulted in a matrix with project factors and occurrences in projects. We combined this matrix with success factors from the quality department and from the questionnaire-based evaluation.

**Step 2:** For factors that happened in more than nine projects in our sample, we did a bottom-up analysis of the text by importing the text into the NVivo tool\(^1\) for analysis of qualitative data and used open coding. Based on the researchers’ experience and knowledge, both of software development in general and of this special company, each of the main categories were split up into five to ten new categories. During the coding process, some of the items in the postmortem reports were moved from one main category to another.

\(^1\)A tool for analysis of qualitative data, available from QSR International.
5.5.4 Results

We now present the key findings from our exploratory study. First we present findings from the qualitative analysis and then the quantitative analysis:

5.5.5 Qualitative Analysis

The five categories that were coded in almost all projects (the five factors on the right in Table 5.6) were analyzed in detail by a qualitative analysis. The categories were “People effects”, “Deliverables”, “Management”, “Knowledge” and “Process effects”. In the following, we discuss what subcategories we found in these main categories.

In the material that was coded as “People effects”, we found the subcategory “lack of technical skills” to be present in five projects. Further, “people unavailable” was a negative issue in four projects, inexperienced project participants in two and also inexperienced project manager in two projects.

An analysis of the category “Deliverables” revealed that the product quality received a negative evaluation by the customer in two projects, and by the company itself in three projects – two projects that had not got a negative customer evaluation. In one project, this was described as “system not ready for delivery”. Also, seven projects mention customer relations as a negative issue related to the deliverables, like “the customer expects to get a lot for free”.

The category coded as “Management” was split into “inadequate initial project plan-
5.5. Case Study: Kongsberg Spacetec

ning” which occurred in six projects. An example of a statement related to this was “not planned for unforeseen expenses and work”. “Bad estimation process” also occurred in six projects. An example statement of this is “risk not taken into account when estimating”. The subcategories “missing or inadequate priorities” and “inadequate project control” occurred in five projects, “inadequate project management” and “inadequate risk analysis” in four projects, “inadequate contract” in three projects. “Process not followed” occurred in two projects.

A lack of “Knowledge” in the projects was mainly related to project management knowledge. “We lack knowledge on planning” was a statement in one report. This subcategory was found in six of the eleven projects. Knowledge related to technology was seen as a problem in four of the eleven projects, for example “little experience with antenna installation”. Lack of knowledge of the customer was seen as a problem in only one project.

For the “Process effects”, we found four subcategories. Process effects related to requirements was mentioned in four projects, related to project management in three projects, external relations and resources in two projects and design, low priority and unclear process were negative issues in one project.

5.5.6 Quantitative Analysis

To gain an understanding of the impact of project factors on the failure of software development projects in the company, we first qualitatively analyzed the information in the postmortem reports.

This qualitative information was then combined with quantitative information, which was obtained separately from the company. As the company had no formal metrics program in place, we relied on subjective measures to get an insight into the quantitative performance of each project. From the development manager, product manager and the QA staff (2 employees), we obtained rankings and ratings of the projects on project focus, strategic importance, customer satisfaction, software quality, and productivity.

As it is a well known fact that subjective ratings are less reliable than objectively collected metrics [Straker, 1995], we started by statistically analyzing the validity of each subjective rating.

A postmortem collects data on many project factors. If we are going to combine these data and, in addition, combine them with other data that the company collects, we need to use every opportunity to check their quality. Important points to check are for instance whether a participant always records the same information in the same way “intra-rater reliability” and whether different participants record the same information in the same way “inter-rater reliability”. If the data that are supposed to agree really do, it increase our confidence in the results, thus increasing the confidence we can have in them and the value they will have when we use them in a decision.

As a basis for this we have used two analysis methods: Kendall’s $\tau_B$, which measures inter-rater reliability, and the Krippendorff’s $\alpha$ [Krippendorff, 1980, Krippendorff, 2006], which is a measure of the agreement between two or more classification schemes or classifiers – the intra-rater reliability.

To understand the impact of project characteristics (the project factors) on the failure of projects (as indicated by the success factors), we need to do more than merely collect data. We can gain understanding by studying the regularities in absent project factors and
the resulting values for the success factors of these projects.

To study the regularities, we use $R^2$ to construct a matrix of correlation coefficients between project factors and success factors. Because the success factors are measured on an ordinal scale measurement and not on an interval scale, we use Kendall’s $\tau$ correlation coefficients instead of the more commonly used Pearson product-moment correlation coefficients.

The correlations in Table 5.8 are based on the factors, as reported by the Quality Assurance staff. Only correlations which are significant at the 5% level are indicated.

We looked at the data from the postmortems for the following success factors:

- Project focus – what was the main aim or goal for this project?
- The satisfaction score – how satisfied were the customer?
- The productivity score – how efficient were the teams when working at the project?
- The quality score – what was the product’s quality?

The project focus factor was left out since this measure had neither intra-rater nor inter-rater reliability (c.f. Chap. 6).

## 5.5.7 Discussion

In this study, our main research question is: what can we learn from analyzing post-mortem reports that have accumulated over time? We discuss our research question through our more detailed questions in the following:

**What characterizes the projects selected for postmortem review?**

In Sect. 5.5.2 we noted that all projects selected for postmortem review shared five main characteristics (negative experiences related to lack of knowledge, people effects, process effects, deliverables and management).
Table 5.8: Correlation table based on factors reported by the Quality Assurance staff.

<table>
<thead>
<tr>
<th>Factor</th>
<th>Productivity</th>
<th>Overrun</th>
<th>Quality</th>
<th>Satisfaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Knowledge</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1.1 Project management</td>
<td>-0.57</td>
<td></td>
<td>0.49</td>
<td></td>
</tr>
<tr>
<td>1.2 Cooperation</td>
<td>0.58</td>
<td>-0.57</td>
<td></td>
<td>0.49</td>
</tr>
<tr>
<td>1.3 Commitment</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4 Team stability</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1 Management process</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1.1 Inadequate initial project planning</td>
<td>0.64</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1.2 Inadequate contract</td>
<td>-0.42</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1.3 Missing or inadequate priorities</td>
<td>-0.52</td>
<td>-0.57</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1.4 Inadequate project control</td>
<td>0.46</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.2 Subcontractor management</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.3 Requirements engineering</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.6 QA</td>
<td>0.69</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.1 Validation and verification</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.1 Software design</td>
<td>-0.51</td>
<td></td>
<td>-0.54</td>
<td></td>
</tr>
<tr>
<td>5.3 Hardware components</td>
<td>-0.50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A.1 Process Outcomes</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.1.1 Low priority</td>
<td>-0.45</td>
<td></td>
<td>-0.50</td>
<td></td>
</tr>
<tr>
<td>4.1.2 Design</td>
<td>-0.45</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A.2 Deliverables</td>
<td>-0.68</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.1.1 Internal product quality</td>
<td>0.77</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.1.2 Customer relations</td>
<td>-0.45</td>
<td></td>
<td>-0.62</td>
<td></td>
</tr>
<tr>
<td>B.1 Process</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B.2 QA</td>
<td></td>
<td></td>
<td></td>
<td>-0.70</td>
</tr>
<tr>
<td>B.3 People</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.1.1 Lack of technical skills</td>
<td>-0.46</td>
<td></td>
<td>0.52</td>
<td></td>
</tr>
<tr>
<td>4.1.2 Inexperienced project participants</td>
<td>0.52</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.1.3 Inexperienced project manager</td>
<td>0.45</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B.4 Tooling</td>
<td>-0.58</td>
<td></td>
<td>0.46</td>
<td></td>
</tr>
</tbody>
</table>

If we are even more precise and focus on the projects that have the largest cost overruns, we can identify what characterizes these projects. According to the statistical analyses on the relation between project factors and cost overrun, the following characteristics/failures lead to the highest cost overrun: inadequate initial project planning, inadequate quality assurance, insufficient validation and verification, poor design and code quality (as noted by internal product quality judged by the workshop or the company’s management) and inexperienced project participants.

This can be an important finding in order for the company to focus it’s software process improvement initiatives.

Do we get similar results when analyzing the data with different perspectives? If not, how can this be explained?

There are notable differences between the results of the quantitative analysis and the workshop. Part of this difference might be explained by the fact that the data for the quantitative analysis originated from management, whereas the input for workshop came from both management and developers.
Which challenges should be considered when analyzing postmortem data (from an academic perspective and from an industry perspective)?

Having observed the discrepancies in point of view between different stakeholders with respect to project success (such as quality and customer satisfaction) it helps to more clearly define the key success indicators of a project. This will help both in achieving the desired results and in analyzing these results afterwards. If at all possible, we should define objective measurement procedures for quality, productivity and customer satisfaction.

Adding different point of view to an analysis helps in gauging the reliability of the results and is a worthwhile exercise for practitioners. The application of a statistical analysis helps to structure the discussion about fail factors of projects, so that the available insights, both quantitative and qualitative, can be integrated.

Having a larger sample of projects, and including both positive and negative project factors from the projects increases the reliability of the findings. Also, projects that do not have a cost overrun represent a learning opportunity, and should hold interesting data for analysis.

As for carrying out the analysis, short reports are difficult to code in retrospect. Doing qualitative and quantitative analysis the way we have performed it is too time-consuming to apply in real-life settings. Industry wanting to learn from postmortem reviews need to combine the knowledge documented in the reports with existing domain knowledge in a more efficient manner, for example in a workshop setting where key points from each review is identified, or axial coding of the reports are discussed. Lightweight statistical analysis can also be appropriate.

Further, the points that were occurring in many projects are candidate issues for more thorough discussions in subsequent postmortem reviews.

5.6 Conclusions

We have presented a method to analyze qualitative, natural language information as often encountered in post-mortem project reports. The method has five steps: identify success factors, select project evaluations, identify project factors, interpret project evaluations, and finally analyze correlations.

This method provides a structured way to deal with qualitative information such as present in post-mortem project reviews. This information can next be related to other, relevant quantitative information that is available. This quantitative information can originate from the company’s project database, from ratings preformed by well-informed experts, or from other dependable source that shed a light on the performance of an IT project.

For both companies, we have identified some issues that the employees that participated in workshops were unaware of. We have also found that some issues identified in the postmortem reports were no longer relevant. This emphasizes the importance of using multiple data sources in software process improvement research.

Note that the statistical significance observed in this type of analysis varies, due to the exploratory nature of the analysis. So we need to have a theory for the correlations observed or confirm the results using experiments. Correlation by itself does not imply causality.
In the second case study, qualitative and quantitative findings indicate different issues in the IT department. We also found that there was little agreement on many of the project success factors between the involved experts. Disagreements between experts with respect to the level of success of a project, made statistical analysis challenging.

The analysis of the post-mortem reports also provided feedback about the process of post-mortem analysis, which was useful for the evaluation process itself. In the first case study we used the experience gained from this analysis to improve closed questionnaires that are used to evaluate and rate software projects. In the second case study we were able to provide advice on where to focus the evaluation effort.

We have found that analysis of postmortem data gives participants new insights into the critical factors of their companies projects. However, broad exploratory analyses, such as the ones we performed, come with a cost that might be too high for small-sized software companies, because reliable results can only be obtained if many projects are analysed.
Chapter 6

Validating Information in Project Evaluation Databases

In the previous chapter, Chap. 5, we proposed a method to analyse the content of project post-mortem evaluation reports. The analysis was based on a five-step method inspired on grounded theory. In this chapter we validate the approach taken in Chap. 5, with mixed results.

The validation is done using two different triangulation methods. For the ABN AMRO Bank case we collected data on 109 new software projects in the same organisation. Using this new information we tried to see if we could obtain the same results. Effectively we are using the logic behind test-retest reliability, and we assume temporal stability of the relationships observed in the previous case study.

For the Kongsberg Spacetec case, we applied a different validation approach. Instead of obtaining a second set of project evaluations, we instead asked an expert panel to validate the relationships found during the case study. Instead of using data on new projects, we are effectively soliciting different information on the same projects and hope to find convergence between the results.

This chapter is partly based on work that is described in a paper that has been submitted for publication [Schalken and van Vliet, 2007] and partly based on a study that has been performed in cooperation with Torgeir Dingsøyr, Nils Brede Moe and Tor Stålhane (from SINTEF ICT and the Norwegian University of Science and Technology – NTNU) and has been accepted for publication at the 14th European Software Process Improvement Conference (EuroSPI 2007) [Dingsøyr et al., 2007].

6.1 Introduction

Post-mortem project reviews provide a feedback process to organisations. Without this feedback, organisations are destined to repeat the mistakes of their past. Usually it is clear how this feedback can be used by the project team members, but there is no consensus on how these lessons can be used by the organisation as a whole. The goal of the analysis method was to package the lessons and insights that lay buried in a stack of post-mortem reports.
Efficient investigations in the elusive domain of software development requires us to take both quantitative and qualitative evidence into account. Quantitative investigations take only a limited number of factors into account, as the required effort and therefore cost of collecting quantitative evidence is high.

Although quite a few critical success factors for software have been identified, there are still a plethora of causes for the success or more often the failure of a software project. If we limit ourselves to investigating just the causes that can be detected by the collected quantitative evidence, we run the risk of missing important causes of failure.

Qualitative data can also be used to discover hypotheses that explain the efficiency of an IT department. Because these hypotheses are based on qualitative data instead of mere gut feeling, they are grounded in reality. And because qualitative data is easier to collect, one can take a broad range of issues into account.

As investigators, we were curious to know whether the correlations between project factors and success factors, as calculated in Table 5.4 and Table 5.8 (both in Chap. 5), bear any resemblance on the real state of practice within the company. Unfortunately there is no such independent, objective data about the relationship between project factors (i.e. the causes of the problems) and success factors (e.g. productivity and satisfaction). Lacking objective data that can be used to verify the correlation matrix, we take a triangulation approach.

For the first study (at ABN AMRO Bank), we take the outcomes of the analysis of the five-step post-mortem analysis method that has been introduced in the previous paragraph. Based on the top 20 most influential success indicators from the previous case study, a post-mortem evaluation questionnaire was constructed. We analyse the results of this questionnaire, that has been administered in the same organisation in which the case study took place to confirm the results we found previously.

For the second study (at Kongsberg Spacetec), we check the consistency of the ratings of the different managers with respect to the success factors and we compare the outcomes of the statistical analysis with a liberal qualitative analysis and with the opinions of a panel of experts.

6.1.1 Threats to Validity

The described method to find correlations between project factors and success factors is by nature an exploratory analysis technique to find starting points for software process improvements. Before actual software process improvements activities are undertaken, the correlations found should be verified in a conformational study.

Having explained that exploratory analysis techniques should not be used to gather evidence, but instead to find starting points for new research, it is still useful to examine which sources of noise and bias could potentially influence the results of the analysis. These sources of noise and bias can lead to spurious correlations where in reality there exist none. Understanding how the validity of a study can be impaired is useful in trying to prevent these problems from occurring.

In this chapter we will only discuss the method specific threats to the validity of outcomes of studies using the method of analysis described the previous chapter. Generic threats to the validity of an experiment will not be discussed as the subject is too broad to describe in a single chapter. There are however excellent books that explain generic threats to the validity of a study, see for example the seminal work by Cook and Campbell...
6.1. Introduction

[Cook and Campbell, 1979].

**Representation Condition of Success Factors**

During our analysis we use indicators for the success of a project, the so-called success factors, to determine if a project factor has a positive or negative influence on the outcome of a project.

Sometimes the indicators used to measure the success of a project, are objective, well-defined, appropriate and tamper-proof. Examples of this kind measures are total development costs, certified by an independent accountant, or developed function points, as counted by an independent consultant.

More often, the indicators for success are not 100% tamper-proof. In these cases people could be tempted to deliberately manipulate these measures, especially as the organisation creates strong incentives for people not to let their project fail (or at least not to let their project look like a failure).

Certain other success indicators, such as customer satisfaction, are less well defined. Other success indicator call for a subjective assessment of the productivity (when the organisation lacks objective data on costs and delivered size).

The outcome of the analysis depends on the reliability on the indicators for project success. Therefore invalid success indicators are a threat to the validity of the analysis.

**Representation Condition of Evaluation Reports**

One of the strengths of (partially) open-ended project postmortem evaluation processes is their ability to discern success or failure conditions that were not anticipated during the design of the development process and the evaluation process. Being able to detect unanticipated sources of failure could help in preventing unforeseen causes of problems to be ignored until they cause irreparable damage to the software product of project.

Open-ended evaluation processes are however a double-edged sword. While the process allows unanticipated remarks, open-ended reviews depend on the subjective evaluation of the project’s success and failure source by the individual project members. Subjective evaluations in general are known to be less dependable than objective evaluations. Subjective evaluations are less crisp and therefore open to manipulation, especially if the organisation creates incentives for certain outcomes of the analysis. The quality of the analysis that is based on these open-ended reviews can only be as good as the quality of project postmortem reports. If the project postmortem reports do not offer an accurate view of the project reality, the result of the analysis will also not reflect that reality.

Sometimes project postmortem reviews do not reflect reality because the project members do not have a good overall picture of the entire project or because the project members have a vested interest in not accurately depicting the project. Most project members are only involved in part of the project or their tasks might only involve one aspect of the entire project. From the point of view of a single project member certain aspects of the project might not seem relevant or productive (for example change control or requirements management) because the benefits of those actions are not for the person involved, but are none-the-less useful for other stakeholders in the project. This could lead to project postmortem reviews indicating problems when in reality there exist none. Another potential source of disturbance is that for certain project members it could be interesting to not give an accurate description of the project. Even if the project manager
knows that the planning was mediocre and caused downstream problems in the project, he manager might not mention this fact if he feels that this revelation could have negative consequences for him. Conflicts of interest of the participants of a project could cause biased findings in the final analysis.

As project postmortem reviews are the only source of information, there is no external oracle that can be used to validate the contents of the project postmortem report. Although no external validation source exists, there do exist internal sources that can be used to validate the findings. If other project members agree upon the findings of the project evaluation, this corroborates the validity of the findings. Especially if the participants of the project postmortem review have different/conflicting interests, this will decrease the chance of manipulation of the review findings. Still, the dependence on subjective interpretations of the project reality remains a weak point in the entire technique.

**Validity of Identified Project Factors**

Not only the project members can cause noise and bias in the findings of the study, also the researchers can introduce bias in the findings of the study. Two points of concern are the identification of project factors and the interpretation of project evaluations. If the identification of project factors is flawed, the resulting concept hierarchy tree will be invalid, causing problems in interpreting and coding the individual project postmortem reports during the interpret project evaluations step. Even with a solid concept hierarchy tree the researchers can misunderstand project postmortem reports or make mistakes in coding the reports for later analysis.

Although the exact form of the concept hierarchy tree is not of direct influence on the final analysis results, it does have a major impact on the subsequent interpretation of the project evaluations. Therefore it is crucial that the concept hierarchy tree is a useful representation of the project factors and that all remarks from the open coding phase can be placed in the concept hierarchy tree with ease.

To gain insight in the quality of the concept hierarchy tree, two researchers can independently create that tree and later compare them with each other to see where there are points that require clarification. To formally compare the resulting concept hierarchy trees of the two researchers, one can use mathematical techniques used in systematic biology to compare taxonomy trees. For more information we refer to [Penny and Hendy, 1985].

**Validity of Project Evaluation Interpretation**

Even with a proper concept hierarchy tree researcher can and will misunderstand project postmortem reports. This will lead to these mistakes being encoded into the data matrix, which will lead to invalid results of the correlation analysis.

There are two strategies that can be used to eliminate this problem. The first method involves letting the original project members code the project postmortem review under the supervision of the researcher. The project member uses the supplied concept hierarchy tree to classify and code the postmortem report. As the project member has a deeper understanding of the project, it is less likely that the statements in the project postmortem report are misunderstood. This approach has been used in our case study, where a project support officer from the organisation in which the study took place, assisted in coding the project postmortem reviews.
A second strategy to eliminate this problem is to let two researchers code each post-mortem review and to compare the interpretations of each project postmortem review. If two or more researchers arrive at the same conclusion, this strengthens the results. As full overlap in interpretation of each postmortem report is unlikely, Cohen’s $\kappa$ [Cohen, 1960] coefficient can be calculated to determine the degree of agreement in interpretation of both researchers. This coefficient gives an impression of how valid the analysis results will be.

6.2 Case Study: ABN AMRO Bank

6.2.1 Context of Case Study

The first case study has been performed within an internal Information Technology department of the ABN AMRO Bank N.V. (a large financial institution). For more background information about the organisation, please refer to Sect. 1.2.

6.2.2 Data Collection

The method we proposed in Chap. 5 has been tested in a case study involving 55 projects at the internal IT department of the ABN AMRO Bank. At the end of the study we noted that the results were only of an exploratory nature and should be used as a starting point for further study, not as a departure point for process improvement. We made this remark because of the low level of certainty that is provided by the analysis method.

In this section we report a study in which we repeated, within the same organisation, a similar analysis based on a new sample of 109 projects.

Data set from Previous Study

The organisation has developed its own post-mortem project evaluation method. The evaluation method consists of an on-line administered questionnaire composed of both open and closed questions. In the evaluation process three groups of stakeholders are addressed: the customer who has commissioned the project, the IT personnel that participated in the project and the involved middle management of the IT department.

At the end of each development project a mandatory evaluation cycle is initiated by the IT project office of the organisation. Upon request of the project office the project leader invites involved stakeholders by e-mail to fill out the evaluation questionnaire. When a sufficient number of stakeholders has filled out the questionnaire, the project leader activates an evaluation consolidation routine in the evaluation program, which anonymises the responses and calculates averages of all the closed questions.

For the analysis of the project information database 55 project evaluations have been selected out of a database containing over 600 project evaluations. The selection of projects included 'normal projects', projects with a specific programming environment, and projects that deviated on: productivity, conformance to budget or conformance to schedule. For the deviant projects, an equal ratio of over-performing and under-performing projects has been selected.

The resulting correlation matrix of the above analysis, which can be found in Tab. 5.4 (page 78), is input for this validation step.
Data set for Confirmatory Study

Based on insights from the prior study, the organisation reorganised their evaluation process. It became clear to the company, that the evaluation of employee, management and customer satisfaction based on the then-used set of questions did not provide enough insights into the development process to justify the costs to collect this data.

During the first study, the organisation had already planned, to reorganise their project evaluation process. Based on their own insights and the results of the first study, the organisation chose a new web-based evaluation tool, called Survey Monkey. With the implementation of this tool the organisation also decided to no longer compute employee, management and customer satisfaction based on the set of questions used previously. Instead, the organisation decided to systematically collect information on project factors that turned out to have a large influence on the key success factors of a project.

The new items on the evaluation questionnaire, that could be rated on a 1 to 5 Likert scale are:

- change management
- project management
- quality planning
- quality schedule
- project control
- test-ware reuse
- the quality of the infrastructure architecture
- the efficiency of the communication
- the cooperation
- the cooperation with-in IT
- the appropriateness of the team size
- the stability of the team
- the stability of the team organisation
- the use of test tool expediter

Changing the questions that are used to calculate the employee, management and customer satisfaction has as a drawback that it is no longer possible to compare previous satisfaction scores with current satisfaction scores. From the company’s perspective this was the right choice however, as the interpretation of the satisfaction scores has been difficult in the past.

An improvement of the questionnaire is that scores are no longer aggregated per stakeholder group (i.e. employee, management and customer), but instead the individual scores are retained. This makes it possible to see how many people assigned a certain judgement to a certain question. This makes it possible to see how much agreement exists on the value of a certain project factor.
Selection Criteria

We collected all data from the new evaluation database, which contained information on 111 projects, and collected the corresponding information from the project administration. On 109 projects the project administration contained sufficient project data for the project to be allowed into the analysis. The remaining two projects had to be excluded because of poor data quality or data completeness in the project administration.

6.2.3 Results

Confirmatory results

We applied an analysis, as described in Chap. 5, to the new data set. The new data set contains information on 109 projects. Unfortunately we were unable to compute customer, employee and management satisfaction, as the organisation decided, with good reasons, to stop measuring these attributes in the way did measured these attributes last time. The results of the analysis, when applied to the new data set, leads to Tab. 6.1.

In the calculation of the correlation matrix, shown in Tab. 6.1, we had to make two changes to the calculation procedure. The first change is a change in the manner in which the values for the project factors is derived, the second change is a change in the way the individual correlation coefficients are calculated.

Before we were able to compute correlations between the project factors and the success factors, we had to obtain the scores for these factors for each project. The method of collecting data on the success factors has remained the same (the same formulae were used to compute productivity, conformance to budget and schedule and duration).

To obtain scores for the project factors, we had to take a different approach from the one used in the previous study. In the previous study, we read, interpreted and subsequently scored each project on a number of factors. This scoring led to a unique single score. In the current study we did not have access to the individual post-mortem reports. Instead we used the scores for the project factors, as assigned by the different stakeholders.

To calculate regression coefficients between the project and success factors, we no longer have a single score for the project factors to correlate the success factor with. The scores for project factors, which have been assigned by the different stakeholders, are not a single score for a project, but instead a set of scores.

To calculate the correlation, we could have taken a simple measure of centrality (e.g. the mean or median) of the set of project factor scores. This method of computation would however not take into account that certain sets of scores have a high level of variance and some sets of scores have a low level of variance. The amount of variance in a set of scores is inversely related to the amount of agreement between the stakeholders on some aspect of a project.

To weigh in the knowledge regarding the agreement between the stakeholders, we calculated a weighted correlation coefficient, where the weight of the observation has an inverted relationship with its variance. To obtain the weighted correlation coefficient, we first performed an ordinary, weighted regression analysis, and then transformed the $\beta$ coefficient of the regression equation into a correlation correlation, that is similar to the Pearson product-moment coefficient.
We used the following formula to calculate the correlation coefficient from the regression coefficient:

\[ r_{x, y} = \frac{S_{x, y}}{S_x \times S_y} = \frac{S_{x, y}}{S_x^2} \times \frac{1}{S_y} = \beta_1 \times \frac{1}{S_y} \quad (6.1) \]

To assess the validity of the analysis method for project post-mortem analyses, we need a measure for comparison between the relationships of the old and the new correlation matrix. Unfortunately, because the Kendall’s \( \tau_B \) correlation coefficient and the Pearson product-moment coefficient do not provide the same values for a data set, we cannot compare the exact values in the matrix but only their direction.

First we tried to compare the correspondence in signs of corresponding items in the two correlation matrices. In 61% of the cases, we find that the sign of the correlation coefficients for corresponding entries are equal (meaning that we found either a positive or negative correlation in both studies). We tested if this match is significantly better than could be expected based on chance (50%), but did not find this difference to be significant (a binomial test yields \( p = 0.13 \)).
This binomial approach does not do justice to the fact that some correlation coefficients that are almost equal (e.g., -0.03 and 0.02) yet have a different sign. In a second iteration we therefore compared the old correlation matrix with the new correlation matrix, by calculating a Pearson product-moment coefficient between the correlation coefficients in both matrices. We obtain a $\rho = 0.26$, with $p = 0.06$. We can interpret this result in the following manner: there is a small, but significant correlation between the two correlation matrices. This means that the analysis method for project post-mortem analyses has predictive power in discerning relations between project and success factors.

This predictive power is sometimes called convergent validity [Campbell and Fiske, 1959]. But as Campbell and Fiske indicated, convergent validity is only the first step to construct validity, divergent validity is also required. Divergent validity means that relationships that are not expected (based on theory or hypotheses) should also not be found in practice. In other words the measurement device should not detect things that are not present.

Unfortunately to measure divergent validity, we would also need to collect data on factors deemed irrelevant (to check their irrelevance), which would defeat the purpose of the screening of project factors. So although the validity of the post-mortem analysis method has been examined, it cannot be fully demonstrated by the collected data.

Regarding the use of correlation coefficient to judge the convergent validity of the post-mortem analysis method, we would like add the following: If we were in a position to calculate the correlation matrices with the same correlation coefficients, we were able to used stronger validity statistics, such as Cohen’s $\kappa$ [Cohen, 1960] or Krippendorff’s $\alpha$ [Krippendorff, 1980].

### 6.3 Case Study: Kongsberg Spacetec

In this second study, we validate the exploratory case study on twelve projects of Kongsberg Spacetec AS, which described in the preceding chapter.

#### 6.3.1 Context of Case Study

Kongsberg Spacetec AS (“Spacetec”) of Norway is one of the leading producers of receiving stations for data from meteorological and Earth observation satellites. Since the company was founded in 1984, its products have been delivered to a number of clients around the world, with a current export share of 85%. Spacetec has expertise in electronics, software development and applications. 80% of the 60 employees in the company have a master’s degree in physics or computer science.

For more background information about the organisation, please refer to Sect. 5.5.

#### 6.3.2 Data Collection

**Selection Criteria**

The data used in this study are collected from twelve software development projects which were finished between 2000 and 2005 at Spacetec. The projects that are analyzed are not a random sample of the company’s projects, but projects singled out because they had cost overruns (8 to 155 percent), see Table 6.2.
Table 6.2: Projects selected for postmortem review.

<table>
<thead>
<tr>
<th>No.</th>
<th>Overrun (%)</th>
<th>Project size</th>
<th>Duration</th>
<th>Report</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>99</td>
<td>large</td>
<td>3y</td>
<td>short</td>
</tr>
<tr>
<td>2</td>
<td>31</td>
<td>large</td>
<td>3y</td>
<td>short</td>
</tr>
<tr>
<td>3</td>
<td>155</td>
<td>large</td>
<td>0.5y</td>
<td>short</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>large</td>
<td>0.5y</td>
<td>long</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>large</td>
<td>1y</td>
<td>short</td>
</tr>
<tr>
<td>6</td>
<td>100</td>
<td>large</td>
<td>1y</td>
<td>short</td>
</tr>
<tr>
<td>7</td>
<td>114</td>
<td>medium</td>
<td>1y</td>
<td>short</td>
</tr>
<tr>
<td>8</td>
<td>85</td>
<td>large</td>
<td>1.5y</td>
<td>short</td>
</tr>
<tr>
<td>9</td>
<td>18</td>
<td>large</td>
<td>3y</td>
<td>long</td>
</tr>
<tr>
<td>10</td>
<td>23</td>
<td>large</td>
<td>2y</td>
<td>short</td>
</tr>
<tr>
<td>11</td>
<td>79</td>
<td>medium</td>
<td>1y</td>
<td>long</td>
</tr>
<tr>
<td>12</td>
<td>79</td>
<td>large</td>
<td>4y</td>
<td>short</td>
</tr>
</tbody>
</table>

During the qualitative analysis, we found out that the projects selected for postmortem review shared five main characteristics. All postmortem reports recorded negative experiences related to lack of knowledge, people effects, process effects, deliverables and management.

Data Sources

We have collected information from three data sources: postmortem review reports, a questionnaire-based evaluation of the projects and memos from a workshop with company participants, where the projects were discussed. We first briefly describe the last data source, the previous sources have been discussed in the preceding chapter.

Workshop

This was done with five persons from the company, who were either from the quality department, or project managers from the projects who also had participated in one or more projects as developers. All participants had participated in one or more postmortem reviews on the selected projects.

In the workshop we asked them to express which events or “project factors” they thought occurred most frequently in the projects under study, and which factors would correlate with productivity, overrun, quality and customer satisfaction. Further, we asked them what they thought would be the dominant factors within a classification framework for analysis: “knowledge”, “management”, “deliverables”, “people effects” and “process effects”. Finally, we asked each participant to comment on the correlations between causes and effects found in a statistical analysis on project factors and success factors.
6.3.3 Results

6.3.4 Quantitative Analysis

As investigators, we were curious to know whether the correlations between project factors and success factors, as calculated in Table 5.8, bear any resemblance on the real state of practice within the company. Unfortunately there is no such independent, objective data about the relationship between project factors (i.e. the causes of the problems) and success factors (e.g. productivity and satisfaction). Lacking objective data that can be used to verify the correlation matrix, we take a triangulation approach.

In a workshop at the company, we asked developers and managers to give an independent assessment of the impact of project factors on success factors. We used the results of this workshop to see how well the answers generated by the objective, quantitative approach matched the subjective opinions from the people involved. This comparison leads to a ranking of correlations vs workshop scores, which we compare using correlations.

When we look at the customer satisfaction score the workshop votes for which project factors that are important in order to develop a product that satisfies the customer and compare this to the correlations, we find that factors like management process and requirements engineering both are considered to have a high importance but do not correlate with the customer satisfaction at all.

If we instead look at the productivity score we observe that the workshop votes for which project factors that are important in order to get a high productivity, we find that factors identified by the correlation matrix and the factors identified by the developers have a Kendall’s $\tau_B$ of -0.5.

Lastly, we look at the quality score. When we look at the developers’ votes for which project factors that are important in order to develop a high quality product and compare this to the correlations, we find that the factor identified by the correlation matrix and the factors identified by the developers have a Kendall’s $\tau_B$ of -0.3.

We see from this analysis that only for the success factor productivity the insights of the correlation table match the insights from the workshop. This might be explained by the fact that productivity is a reliable measure (high intra- and inter-rater reliability), whereas satisfaction and quality measures are unreliable.

When we look at the customer satisfaction score we find a $\tau_B$ of -0.6, which indicates that satisfaction is an intra-rater reliable score but we find a Krippendorff’s $\alpha$ of -0.3, indicating a low inter-rater reliability.

If we instead look at the productivity score we find a $\tau_B$ of -0.5, which indicates that satisfaction is an intra-rater reliable score and a Krippendorff’s $\alpha$ of -0.8, indicating a high inter-rater reliability.

Lastly, we look at the quality score. Here we find a $\tau_B$ of -0.8, which indicates that satisfaction is an intra-rater reliable score but a Krippendorff’s $\alpha$ of -0.3, again indicating a low inter-rater reliability.

Qualitative Analysis

When we asked the participant in the workshop to indicate which events (or project factors) they thought would occur most frequently, they ranked them as shown in Table 6.3, together with the occurrence taken from the postmortem reports. Some factors that occurred frequently in the reports matched the belief amongst the participants: process
Table 6.3: Reported and believed ranking of factors for the selected projects

<table>
<thead>
<tr>
<th>Event</th>
<th>Report ranking</th>
<th>Workshop ranking</th>
<th>Rank difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process effects</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Knowledge</td>
<td>1</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>Management</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>People effects</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Deliverables</td>
<td>4</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>QA effects</td>
<td>6</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>Process outcome</td>
<td>7</td>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>Software design</td>
<td>7</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>Tech design</td>
<td>9</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Requirements engineering</td>
<td>9</td>
<td>13</td>
<td>5</td>
</tr>
<tr>
<td>Tools effects</td>
<td>11</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>QA</td>
<td>11</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>Validation process</td>
<td>11</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>Team stability</td>
<td>11</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>Commitment</td>
<td>11</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>Cooperation</td>
<td>11</td>
<td>4</td>
<td>7</td>
</tr>
</tbody>
</table>

effects and management and deliverables were among the top in both ratings. However, process outcome, cooperation, team stability and validation process were factors that were believed to be fairly frequent, but only seldom appeared in the reports.

We also asked the participants to comment on the subgrouping of the most frequently occurring factors. The participants had a few comments, but generally thought the subgroups were reasonable.

The workshop participants commented that the large difference for “knowledge” was that the postmortem reports were written at a time when there were many new employees in the company. Another comment was that developing software for space applications, there is always new technology involved, which means that there must always be time allocated for learning. At the time of the workshop “Knowledge” was not seen as a problem anymore, but as a constant challenge in all new projects.

### 6.3.5 Discussion

**Do information sources provide consistent information about the projects? If not, how can this be explained?**

We compared the results from the qualitative analysis with perceptions of the workshop participants. The following project factors had a short distance in ranking between reports and workshop (2 or less):

- Process effects
- Deliverables
6.3. Case Study: Kongsberg Spacetec

- Tech design
- Management
- Tool effects
- QA
- Commitment

The following factors occurred frequently in the reports, but were not ranked high in the workshop:

- Knowledge
- QA effects

The following factors occurred infrequently in the reports, but were ranked high in the workshop:

- Validation process
- Team stability
- Cooperation

An error source for the data collection is the ability of the participants in the workshop to remember the projects in question – the workshop was carried out up to six years after the projects finished. Another error source is the workshop participants knowledge of all projects, some did only have detailed knowledge about a subset of the projects.

As for the quantitative data, except for productivity, where Krippendorff’s $\alpha = 0.76$, the other subjective ratings on success factors (quality and customer satisfaction) shows that the data are unreliable. For quality and customer satisfaction, the ratings differ wildly between the different observers. This difference in ratings, or lack in inter-rater agreeableness, means that the measurements should not be used.

The lack of agreement might be caused by the lack of clear and shared definitions of quality and customer satisfaction. Productivity on the other hand seems to be a straightforward concept for most people involved.

**Do we get similar results when analyzing the data with different perspectives? If not, how can this be explained?**

There are notable differences between the results of the quantitative analysis and the workshop. Part of this difference might be explained by the fact that the data for the quantitative analysis originated from management, whereas the input for workshop came from both management and developers.

An additional explanation might be that the reliability of the source data might have played a role in the accuracy of the statistical analysis. There is much more agreement between workshop participant and statistical results with regard to productivity (which has the highest reliability) than with regard to cost overrun, quality and customer satisfaction.
Which challenges should be considered when analyzing postmortem data (from an academic perspective and from an industry perspective)?

Having observed the discrepancies in point of view between different stakeholders with respect to project success (such as quality and customer satisfaction) it helps to more clearly define the key success indicators of a project. This will help both in achieving the desired results and in analyzing these results afterwards. If at all possible, we should define objective measurement procedures for quality, productivity and customer satisfaction.

Adding different point of view to an analysis helps in gauging the reliability of the results and is a worthwhile exercise for practitioners. The application of a statistical analysis helps to structure the discussion about fail factors of projects, so that the available insights, both quantitative and qualitative, can be integrated.

Having a larger sample of projects, and including both positive and negative project factors from the projects increases the reliability of the findings. Also, projects that do not have a cost overrun represent a learning opportunity, and should hold interesting data for analysis.

### 6.4 Conclusions

The framework we proposed in this chapter helps to structure the investigation into critical success and fail factors for an organisation. The framework guides data collection processes that gather evidence on the causes of success or failure of a project. By the subsequent gathering of ever more stronger data, one can build reliable theories regarding the effectiveness of the organisation and at the same time prevent the collection of excessive amounts of data that would bring with it excessive costs.

In the first validation study, executed at ABN AMRO Bank N.V., we provided some preliminary evidence that the post-mortem analysis method can provide useful, but not perfect information.

In the second validation study, executed at Kongsberg Spacetec AS, we have analyzed twelve postmortem review reports from a medium-size software company in a qualitative and quantitative analysis, focusing on negative experiences. In addition, we have gathered opinions on the projects analyzed through a questionnaire and through a workshop discussion. Qualitative and quantitative findings indicate different characteristics. We have also found that there was little agreement on project success factors, which made statistical analysis challenging.

Still we were able to identify some issues for the organisation that employees who participated in workshops were not aware of. We have also found that some issues identified in the postmortem reports were no longer relevant. This emphasizes the importance on multiple data sources in software process improvement.

Overall we can conclude that the analysis of richer sources of evidence, such as postmortem data can give new insights into the success and fail factors of projects in an organisation. The method proposed in the previous chapter can help to identify factors that have an impact on the outcomes of a project. To obtain these results one needs to perform a large, labour intensive analysis of a large number of projects. It would be beneficial if further research could identify ways to decrease the effort required for the analysis.
Chapter 7

Measurement of Infrastructural Projects

In the previous chapters, we have used productivity indices and models to assess the efficiency of the software development process. The productivity indices and models used were all based on functional size metrics derived by function point analysis [Albrecht, 1979, Albrecht and Gaffney, 1983, Garmus and Herron, 2002], more specific the Dutch variant of function point analysis, standardised by the NESMA [Nesma, 1997]. The productivity indices and models express the relationship between the functional size and effort, which is a measure for efficiency.

Although function point analysis has become an industry standard and has many positive aspects [Furey, 1997], it is not without flaws [Kitchenham, 1997] nor is it the only measure for functional size (e.g. object points [Banker et al., 1992, Banker et al., 1994b] and function bang [DeMarco, 1982]). More important the functional perspective of the software size is not the only perspective one can use. Alternative perspectives on the size of a project are the length of the software code (i.e. the well known lines of code [Park, 1992]) and the size of the required technical infrastructure (i.e. the required databases, operating systems and middleware). The alternatives to functional size measurement are not necessarily better (c.f. Jones [Jones, 1986] for a description of problems with counting lines of code), but together they offer a more balanced view of the outcomes of a project.

For the effort required to deploy and configure the required technical infrastructure in a software project is everything but negligible. It is therefore odd that there are no standard metrics to express the size of the technical IT infrastructure task. In this chapter we describe a metric that can be used to measure the size of projects that install and configure COTS stand-alone software, firmware and hardware components. We call these IT infrastructure, as these components often form the foundation of the information system that is built on top of it. The proposed metric promises to be a viable instrument to assess the effectiveness and efficiency of IT infrastructure projects.

This chapter is based on a paper presented earlier at the 17th Conference on Advanced Information Systems Engineering (CAiSE 2005) [Schalken et al., 2005].
7.1 Introduction

Organizations no longer create software intensive systems from scratch. The use of pre-existing software components, not created by the organizations themselves, becomes ever more prevalent in the creation of large software systems [Abts et al., 2000, Morisio et al., 2002]. These pre-existing components are often called *commercial-of-the-shelf components* (COTS components) in software engineering literature. In this chapter we however prefer to use the term *non-developmental items* (NDIs) [Carney and Long, 2000] for these pre-existing components.

The integration of NDI components that are packaged as stand-alone programs differs significantly from traditional software development. Many software engineering metrics that have been applied to software development (such as function points [Albrecht, 1979, Albrecht and Gaffney, 1983], lines of code [Park, 1992], object points [Banker et al., 1994b], or bang metrics [DeMarco, 1982]) cannot be applied to projects that integrate these NDI components into larger systems. After all most effort in infrastructural IT projects is not in programming a system, but in installing and configuring the system. In this chapter we propose a new software metric to measure the size of projects that integrate stand-alone NDI components into software-intensive systems.

The metric we propose is not only applicable to the integration of stand-alone non-developmental software components, but also to the integration of firmware\(^1\) and hardware components into software-intensive systems. We use the term *Information Technology infrastructure* (IT infrastructure) to refer to NDI hardware, firmware and stand-alone software components, as these components often form the foundation of the information system that is built on top of it.

Examples of IT infrastructure projects are: operating system upgrades, installations of a database system, deployment of new desktop computers and memory upgrades of servers.

Up until now no size metric in the area of IT infrastructure development has received broad acceptance by industry. Although no standard size metric for IT infrastructure exists, it does not mean that there is no need for such a metric. On the contrary, the need for such a size metric is increasing. Empirical knowledge of NDI-based systems is still at an early stage of maturity [Basili and Boehm, 2001]. Processes are quite different from traditional projects [Morisio et al., 2000] and project estimation and tracking are less effective for NDI-based development [Morisio et al., 2000]. This is problematic as the use of NDI components is becoming ever more prevalent [Abts et al., 2000, Morisio et al., 2002]. The metric we propose in this chapter uses information part of which only becomes available late in a project. Consequently, it’s intended use is to assess and validate the effectiveness and efficiency of projects, rather than upfront cost estimation.

7.1.1 IT Infrastructure Defined

The absence of consensus on the meaning of COTS within the academic community [Carney and Long, 2000, Morisio and Torchiano, 2002] necessitates a definition of the related concept IT infrastructure in this section. The definition of IT infrastructure marks which projects can and which projects cannot be measured with the new size metric.

\(^{1}\) As firmware has remarkable similarity with software (with respect to its configuration), everywhere were software is written, on should read software/firmware unless explicitly stated otherwise.
7.1. Introduction

The term IT infrastructure has been inspired by the term technical infrastructure [Stensrud and Myrtveit, 2003]. In this chapter the following definition of *IT infrastructure development* will be used: “the deployment, installation, connection and configuration of both new and upgraded, non-developmental hardware, firmware and stand-alone software components”.

The development of IT infrastructure is concerned with pre-existing hardware and software components that have not been developed by the organizational unit that installs these components. In software engineering literature a pre-existing software component is often called a *commercial-of-the-shelf component* (COTS component). In this chapter we however prefer to use the term *non-developmental item* (NDI) [Carney and Long, 2000], as the term COTS implies that the component comes from a commercial vendor. In this chapter the term NDI refers not only to software components, but also to firmware and hardware components.

Non-developmental software components can be packaged in several ways [Morisio and Torchiano, 2002], either as source code, static libraries, dynamic libraries, binary components or stand-alone programs. The type of packaging also has a direct impact on how these components are integrated and delivered to the customer. NDI components that are provided in source code or library form usually require programming to integrate the components into the software system under construction. These kinds of NDI components are usually delivered as inseparable subcomponents of the system under construction. On the other hand NDI components that are provided as stand-alone programs usually require little or no programming, but require so much the more configuration. These NDI components are usually not delivered as an inseparable system, but instead the components need to be installed separately or are installed by the installation program as separate, stand-alone components or programs.

The development of IT infrastructure not only entails the installation and configuration of stand-alone software components, but also the deployment, connection and configuration of hardware components. As the choice of software components is not independent hardware components and because the integration of the components is frequently performed in a single project, we have chosen to measure the size of software, firmware and hardware components using a single, composite size metric.

The development of IT infrastructure consists of *deploying* the hardware (placing the physical components), *installing* the stand-alone software (loading the software from the installation medium into the target hardware), *connecting* the hardware (installing sub-modules and wiring the hardware) and *configuring* the software (setting and testing the configurable parameters and settings).

7.1.2 Related Work

This section describes related work in the field of IT infrastructure size metrics and algorithmic cost estimation models for IT infrastructure. Cost estimation models are included in the overview, as they can be seen as size models that do not only take the inherent problem size into account, but also the capability of the environment to deal with the complexity at hand. In this article we focus solely on the costs of labor for the installation and configuration of the IT infrastructure. Selection effort, hardware costs, such as discussed in [Ardagna et al., 2004], and license costs lie outside the scope of this chapter.

There are two cost estimation methods that are commonly used in practice: *IT in-
**Infrastructure service costing** and **IT infrastructure product costing**. The IT infrastructure component is either viewed as a service whose costs are of a recurring nature or as a component that is delivered to an organization as product or a one-off service delivery. Examples of IT infrastructure services are CPU cycles on a mainframe and network ports. Examples of IT infrastructure products are configured servers and software upgrades.

The most crude approach to IT infrastructure costing consists of amortizing the total costs generated by a class of IT services or IT products by total amount of service units or products delivered.

A more sophisticated approach to IT infrastructure service costing is offered by Van Gorkom [van Gorkom, 2002] in the form of Service Point Analysis. The Service Level Agreement is decomposed into Service Level Agreement components. Based on the Service Level Agreement components standardized cost estimates can be made.

Another more sophisticated approach to IT infrastructure product sizing is the SYSPOINT method [Raghavan and Achanta, 2004]. The IT infrastructure product to be provided is divided into primitive components (servers, workstations, printers, LAN’s, WAN’s, server applications and client applications). Based on the relative complexity and count of the primitive components, tables can be used to calculate the total size of a project in SYSPOINTs.

COCOTS [Abts and Boehm, 1997,Boehm et al., 1999] is an advanced cost-estimation model for NDI software, based on the COCOMO suite of cost estimation models. The COCOTS model allows the estimation of not only the implementation of the system, but also the selection costs and modification costs. The COCOTS tailoring model estimates the implementation and configuration costs of a system, based on parameter specification, script writing, reports & gui, security and the availability of tailoring tools. Each of the factors is measured on a five-point scale.

The last method discussed in this section is data envelopment analysis, which can be applied to measure the relative efficiency in creating IT infrastructure [Stensrud and Myrtevit, 2003]. Data envelopment analysis allows the efficiency of projects to be compared on a variety of output measures simultaneously. In that sense it is not a cost estimation or size measurement procedure, but the method does shine a light on the project’s productivity. The method solves the problem that IT infrastructure can have multiple outputs (e.g. servers can have connected users, storage space and processing speeds as output measures).

Amortizing product or service costs, Service Point Analysis and the SYSPOINT method share the drawback that estimates can only be made for IT infrastructural systems that consist of known infrastructural product or service components. The costs for IT infrastructural systems that contain novel product components or service components cannot be estimated.

Data envelopment analysis can only analyze the implementation efficiency of projects that implement similar products or services. Different IT infrastructural projects will yield very different primitive outputs. Compare the efficiency in providing network throughput with the efficiency of providing storage space (both measured in gigabytes). This explains why comparisons are only possible between projects that have similar end results, comparing throughput with storage of data is of course not meaningful.

Our method measures the size of IT infrastructure on a continuous, interval scale. It is reasonably precise, whereas COCOTS measures each attribute on a rough 5-point scale. The metric allows different types of IT infrastructural products to be compared
to each other and does not depend on the existence of a list of known IT infrastructure components.

### 7.1.3 Structure of Chapter

Having discussed the necessity of IT infrastructure metrics and the definition of IT infrastructure, the remainder of this chapter is structured as follows: In Sect. 7.2 the metaphor that guided the design of the infrastructure metric is presented together with the formal definition of the metric. Section 7.3 discusses the calibration of the measurement formulas presented in Sect. 7.2. Section 7.4 provides the results of the preliminary calibration and validation of the metric during the feasibility study. Section 7.5 describes the conceptual validation of the proposed metric for IT infrastructure. And the last section wraps up the chapter with some concluding remarks and further work.

### 7.2 Infrastructure Effort Points

In this section we present our metric to measure the size of IT infrastructure projects, the **Infrastructure Effort Point** or IEP for short. First the the underlying principles that guided the design of the size metric are explained. Following the theory design of the metric a detailed description of Infrastructure Effort Points and its measurement procedure is given.

Infrastructure Effort Points only measure the size of the installation and configuration effort of the IT infrastructure. The NDI component selection effort, training effort, hardware costs, and license costs cannot be measured using this metric.

#### 7.2.1 Theory Design

In this section we explain the design of the size metric for IT infrastructure using a metaphor. A metaphor helps to create a common understanding [Robinson and Sharp, 2003]. It can be used to explain the underlying principles and assumptions of a measurement procedure.

The metaphor gives an intuitive justification for a metric. For example function point analysis counts the variety and complexity of the data records that are received, stored,
transmitted and presented by an information system. Function point analysis is able to measure the size of different classes of information systems by abstracting each information system to a manipulator of flows of information. This is the guiding metaphor of function point analysis. It is based on the assumption that the complexity of an information system is equal to the complexity of its information flows.

Infrastructure Effort Point analysis considers the development of IT infrastructure to consist of two activities: wiring and placing hardware boxes during the deployment and connection of hardware components and the manipulation of configuration settings during the configuration of software components and configurable hardware.

The guiding metaphor for Infrastructure Effort Points is based on the following three assumptions:

1. Infrastructural IT projects are composed of a hardware and a software component.

2. The effort of the hardware component of the project depends on the number of hardware boxes that need to be installed and the number of connections that need to be made.

3. The effort of the software component of the project depends on the number of configuration parameters that need to be set.

### 7.2.2 Hardware Effort Points

Two distinct tasks in IT infrastructure projects have been identified: tasks related to hardware and tasks related to software. Verner and Tate [Verner and Tate, 1992] argue that different types of system components can have different size equations. In this section we identify the size drivers and size equations that are applicable to the deployment and connection of hardware components.

A bottom-up size model, as the Infrastructure Effort Points, consists of a number of size drivers and one or more size equations. A size driver is “any countable, measurable, or assessable element, construct, or factor thought to be related to the size” of a component [Verner and Tate, 1992]. The size drivers form the input to a size equation that combines the different counts on a specific size driver into a single size measurement.

For the hardware side of the Infrastructure Effort Point equation we distinguish three major size drivers: main components, subcomponents and connections. Each of the

<table>
<thead>
<tr>
<th>Group</th>
<th>Size driver</th>
<th>Symbol</th>
<th>Unit of measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>main components</td>
<td>number of components</td>
<td>$c_i$</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>average weight</td>
<td>$c_i^w$</td>
<td>kilo</td>
</tr>
<tr>
<td></td>
<td>installation or removal</td>
<td>$c_i^a$</td>
<td>{installed, removed}</td>
</tr>
<tr>
<td>subcomponents</td>
<td>number of subcomponents</td>
<td>$s_{i,j}$</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>average number of connections</td>
<td>$s_{i,j}^c$</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>installation or removal</td>
<td>$s_{i,j}^a$</td>
<td>{installed, removed}</td>
</tr>
<tr>
<td>external connections</td>
<td>number of connections</td>
<td>$w_{i,j}$</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>average length</td>
<td>$w_{i,j}^l$</td>
<td>meter</td>
</tr>
<tr>
<td></td>
<td>installation or removal</td>
<td>$w_{i,j}^a$</td>
<td>{installed, removed}</td>
</tr>
</tbody>
</table>

Table 7.1: Atomic measurements for IEP of hardware installation.
major size drivers has associated minor size drivers that can be used to fine-tune the size equations in the future. Although technically there is no difference between a major and a minor size driver, practically we expect the major size drivers to have greater influence on the size of the IT infrastructure project.

The first major size driver is the number of main components \( c_i \) of a certain type of hardware that has been installed or removed. Main components are those pieces of hardware that are considered to form a functional whole by their average end users. E.g. an end user will identify a scanner as a main component, whereas the separate automatic document feeder for the scanner is seen as a subcomponent, as the automatic document feeder cannot be seen as an independent machine that offers functionality on its own. Associated minor size drivers are the average weight of the main component \( c^w_i \) and whether the components were installed or removed \( c^a_i \).

The second major size driver is the number of subcomponents \( s_{ij} \) of a certain type that have been installed or removed from main component \( c_i \). (The index \( i \) refers to the main component to which the subcomponents are attached, the index \( j \) refers to this particular group of subcomponents.) Subcomponents that have not been installed or removed, but instead were already assembled with the main component should not be counted. Minor size drivers that are associated with the size driver number of subcomponents are the average number of connections between the subcomponent and the main component and other subcomponents \( s^c_{ij} \) and whether the subcomponents were installed or removed \( s^a_{ij} \).

The third and last major size driver for the hardware side is the number of connections \( w_{ij} \) between the outside world and main component \( c_i \). The connections size driver considers all physical connections (wires) between the main component and the outside world, but not the mutual connections between subcomponents and connections between subcomponents and the main component as these have already been counted (in \( s^c_{ij} \)). Examples of connections are the power cable and the network cable of a personal computer, but not the keyboard cord. Associated minor size drivers are the average length of the connection \( w^l_{ij} \) and whether the connections were installed or removed \( w^a_{ij} \).

These three major size drivers and their associated minor size drivers form the input for the size equation that combines the measurements on the individual size drivers, see Fig. 7.1 for a schematic overview. The size equations consist of a model combined with calibrated model parameters. The equation model states which size drivers need to be taken into account and in which manner. E.g. a size model might state that the size of a task is equal to the number of main hardware components multiplied by a constant plus the number of connections multiplied by a constant, i.e. size \( s_{i}^{hw} = \theta_1 \cdot c_i + \theta_2 \cdot w_{ij} \). The exact size equation is determined during the calibration process in which the most appropriate values for the constants in the equation are determined using empirical data.

The calibration of a size model, once the empirical data has been collected, is conceptually easy, albeit computer-intensive (more information on the calibration process can be found in Sect. 7.3). The selection of the appropriate model is more complicated. Apart from the selection of an appropriate form for the equation (linear, logarithmic or quadratic) one needs to be careful to select the right amount of size drivers for the model. Too few size drivers makes the size equation perform poorly, as insufficient information is taken into account. Too many size drivers creates the risk of over-fitting the size model, causing the size equation not to capture the real hardware size but instead some random patterns that exist in the observed data. The risk of over-fitting the model is increased
Table 7.2: Atomic measurements for IEP of software configuration.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Metric</th>
<th>Symbol</th>
<th>Unit of measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration parameters</td>
<td>number of parameters</td>
<td>$p^i_1$</td>
<td>-</td>
</tr>
<tr>
<td>parameter type</td>
<td></td>
<td>$p^v_i$</td>
<td>{text, number, boolean, binary}</td>
</tr>
<tr>
<td>input type</td>
<td></td>
<td>$p^i_1$</td>
<td>{gui, text interface, configuration file, configuration database, script file, dip-switch/jumper, other}</td>
</tr>
<tr>
<td>configuration group</td>
<td>number of parameters</td>
<td>$g^i_{ij}$</td>
<td>-</td>
</tr>
<tr>
<td>parameter type</td>
<td></td>
<td>$g^v_{ij}$</td>
<td>see above.</td>
</tr>
<tr>
<td>input type</td>
<td></td>
<td>$g^i_{ij}$</td>
<td>see above.</td>
</tr>
</tbody>
</table>

when many size drivers are included relative to the amount of empirical data.

For the first empirical validation we propose to use only a single size driver to prevent over-fitting the data. When more data becomes available more complex size models can be examined. The most important hardware size driver is the number of main components. We therefore propose to use the following simple formula to calculate the size of the hardware part $s^{\text{hw}}$.

$$ s^{\text{hw}} = \sum_{i=1}^{\theta^{\text{hw}}_1} c_i $$

7.2.3 Software Effort Points

For the software side of the Infrastructure Effort Point equation we should be able to apply the method to all possible situations, for the software side of the Infrastructure Effort Point equation we differentiate between two usage scenarios. In the first scenario the software is configured for the first or second time in the organisation (configuration engineering), whereas in the second scenario the organisation has a degree of experience in configuring the software (configuration roll-out). In a large deployment of software within an organisation one usually starts with configuration engineering and when all the required information about configuring the software has been gathered the project proceeds with configuration roll-out.

The difference between a first-time configuration and a repeat configuration is the familiarity with the software product. During a first installation one needs to examine all configuration parameters to determine which parameters require adjustment. When one is familiar with a system one knows which parameters require adjustment and which factory settings are already correct.

For the software side of the Infrastructure Effort Point equation we distinguish one or two major size drivers. With configuration roll-out projects the major size driver is the number of configuration parameters. In recognising the effort required to determine which parameters to change, configuration-engineering has a second major size driver the total amount parameters in a group of parameter settings, that measures all available configuration parameters.
7.3. Calibration Process

The major size driver for software configuration tasks is the number of configuration parameters $p_i$ that require modification. During the installation of a software component the software is loaded from the installation medium to the target execution platform and simultaneously the settings of the components are set. The type of settings that determine the behaviour of a component can vary, broadly from installation directories, subsystem selection, user account creation, user settings to script files. Associated minor size drivers are: the type of values a parameter can store $p_{ij}^v$ and the input method that is required to change the parameter value $p_{ij}^t$. The type of parameter values has an influence on the effect size of the task, because binary strings are much harder to enter and test compared to boolean parameters. The input method also has influence on the configuration size, as for example configuration using a gui is easier than configuring a system using a configuration file.

The other major size driver for configuration engineering tasks is the number of configuration parameters in the configuration group $g_{ij}$ that belong to the configuration parameter $p_i$. As configuring a system with only a few parameters is easier as configuring a system with a large number of parameters, the number of available configuration parameters also needs to be taken into account into the size equation. Therefore, all existing configuration parameters, that are seen by the IT team during installation and configuration are counted. The associated minor size drivers are the same as those described for the size driver number of configuration parameters.

To calculate the size for roll-out configuration tasks, the following model formula can be used:

$$s^{sw} = \theta_{1}^{sw} \cdot p_i$$

To calculate the size for configuration-engineering tasks, the following model formula can be used:

$$s^{sw} = \sum_{i=1}^{\theta_{1}^{sw}} p_i + \sum_{j=1}^{a} (\theta_{2}^{sw} \cdot g_{ij})$$

7.2.4 Infrastructure Effort Points

Having explained both the hardware and software part of the Infrastructure Effort Point measurement, we are ready to combine these two measurements into the overall Infrastructure Effort Point measurement.

To obtain the (total) Infrastructure Effort Point measurement the Hardware Effort Points and the Software Effort Points are added. To prevent one of the two measurements overruling the other measurement, a scaling factor $\theta_{scale}$ is used. This leads to the following equation:

$$s^{hw} = s^{hw} + \theta_{scale} \cdot s^{sw}$$

7.3 Calibration Process

The purpose of the calibration is to find the IEP model parameters for an IEP size model. Together with the size model, the IEP model parameters make up the estimation equation.
of the Infrastructure Effort Points.

A good size metric should correlate well with development effort. We therefore define an optimal set of IEP model parameters to be those parameters that allow a cost estimation model to make the best possible prediction/explanation of the costs of a project. The advantage of using a cost estimation model as an intermediate to correlate size with development effort are twofold. First the cost estimation model can take phenomena as diseconomy of scale and schedule compression into account. The second advantage is that once the model has been calibrated using empirical data, other people can more easily recalibrate the model to their specific environment; the only need to recalibrate the estimation parameters.

To find the optimal calibration of the measurement model, empirical data about the infrastructural size drivers and the development effort is needed. The optimization process consists of the cyclic process of improving/modifying the initial IEP model parameters and consequently calculating error of the estimation algorithm (compared with the historical performance), see Fig. 7.2.

### 7.4 Feasibility Study

Having described the definition of the Infrastructure Effort Points and its calibration process, we describe the results of a preliminary feasibility study conducted to test the practicality of the data collection and its associated calibration process.

To test the data collection and calibration process, we collected measurements and effort data of nine projects that were collected in a controlled environment. The projects consisted of both hardware and software infrastructure projects. The aggregated results can be seen in Table 7.3

The conclusion of the feasibility study is that it is possible to collect the required data in an efficient manner that does not disrupt the IT infrastructure project itself. All that is required is some degree of discipline in recording the steps during the project. Based on the data we are able to obtain a preliminary calibration of the data, however as this calibration is based only on nine projects it is not statistically significant.

---

2The full dataset of the preliminary data collection is available in Microsoft Access-form, from the following address: [http://www.cs.vu.nl/reflection/infra-metrics/](http://www.cs.vu.nl/reflection/infra-metrics/)
7.5 Measurement Validation

Software metrics need to be validated to ensure that they measure what they purport to measure [Schneidewind, 1992]. The validation of the metric should check whether the metric is valid and/or correct.

The validity of a software metric refers to its ability to provide measurements that can be used by practitioners in the context in which the metric was gathered. The correctness of a software metric refers to generic “laws” that govern measurements in general. An example of a correctness requirement is that addition of two measurement values should lead to a meaningful total. This requirement and other many correctness requirements are codified in measurement theory (see e.g. ).

As correctness requirements on software metrics have been extensively discussed in the academic literature (e.g. [Fenton and Pfleeger, 1998, chap. 3]) they will not be discussed in more detail in this chapter. In following section we pay more attention to the validity of the Infrastructure Effort Point metric. The validity requirements that are discussed in this section are: the focus aspect, the objectivity aspect, the timeliness aspect, the granularity aspect, and the genericity aspect.

7.5.1 Validity Requirements

The focus aspect of a software size metric determines whether the emphasis of the size of an IT solution lies on the size of the problem to be solved with an IT solution or on the size of the IT to create the solution. Certain complex function requirements on a system might take little implementation effort given the right tools whereas certain apparently simple functional problems might require a large amount of implementation effort because tool support is lacking. Metrics with a value focus pay more attention to the size of the problem and less on the size of the IT required to solve the problem. Metrics with a work focus pay more attention to the size of the IT solution as compared to the problem.

The IT infrastructure size metric has a work focus, because typical infrastructural IT projects are not free to choose which infrastructure to implement to support the functional requirements of the user. Lacking influence on the choice and availability of suitable IT infrastructure, it would not be fair to hold projects accountable for their implementation
efficiency.

The **objectivity aspect** of a software size metric dictates whether the determination of the size of a software can be based partly on human judgment or can only be based on rules that can be interpreted in only a single manner. **Objective metrics** require only the application of clear rules and require no human judgment. **Subjective metrics** on the other hand do require human judgment and interpretation before a size can be determined. C.f. lines of codes [Park, 1992] which can be counted automatically by a computer (a fully objective metric) with function points [Albrecht and Gaffney, 1983] (a partially subjective metric) which require a human function point counter to interpret the requirements specification. Size metrics should be as objective as possible, as subjectivity leaves room for disagreements on the real functional size and causes inaccuracies (e.g. [Kemerer, 1993]). The objectivity aspect of a metric is not a crisp distinction between objective and subjective, but a wide spectrum of nuances is possible.

The IT infrastructure size metric is an objective measure of the size of an IT infrastructural task. The determination of which hardware components are main components and which are sub-components does involve some subjective judgement. The same holds for the determination of the number of parameters in a parameter group, as the boundaries of the parameter group are not always very clear.

The **timeliness aspect** of a software size metric determines in which phase of the project one needs to be able to determine the size. Size measurements require information about a project or piece of software that becomes available in the course of the project. Certain information is available already at an early stage of a project whereas other information only becomes available near the end of the project.

The IT infrastructure size metric is meant to be used for assessment and evaluation of methods and practices. This usually takes place at the end of a project, in contrast to practices such as estimation and project tracking that take place at an earlier phase of the project. As assessment and evaluation take place at the end of a project, it is not problematic if some of the required data only becomes available near the end of a project.

The **granularity aspect** of a software size metric refers to the aggregation level to which the metric is applied. The efficiency can be determined of a single project, of all work related to a single product type or of all work in an organizational unit. E.g. the efficiency of single network installation project can be measured (project-level granularity) or the efficiency of all networking operations in an organization based on cost per network point can be measured (organizational-level granularity). Between project-level and organizational-level metrics lie the **product-level** metrics that measure the efficiency of implementing a single type of product (e.g. a Linux server).

The IT infrastructure size metric needs to have a project-level granularity to explain in which contexts methods do work and in which contexts they do not work. Metrics with an organizational-level granularity would obscure the reasons why certain processes do or do not work.

The **genericity aspect** of a software size metric indicates how broad the applicability of the metric should be. **General-purpose** metrics can be applied to a broad range of software products, whereas **special-purpose** metrics can only applied to a limited range of software products.

The IT infrastructure size metric needs to be a general-purpose metric. IT infrastructure includes a broad area of products. One can only meaningfully compare efficiency rates that are based on the same size measurements. It would therefore be beneficial
if most IT infrastructure can be measured with the metric, allowing comparisons of the applicability of processes in different infrastructural domains.

7.6 Conclusions

In this chapter we discussed the design and principles of the Infrastructure Effort Point metric for IT infrastructure projects. The metric is an objective metric that can be used to measure the size of IT infrastructure projects. It outperforms other existing size metrics for IT infrastructure in genericity and objectivity.

Infrastructure Effort Points can help to assess the effectiveness of processes and techniques, making it a valuable tool for process improvement for organizations that deliver IT infrastructure.

The Infrastructure Effort Point metric is unsuitable for the upfront estimation of a project’s schedule or costs. The required information to feed into the size equation is available only at a late stage of the project. However with the aid of additional techniques (e.g. PROBE [Humphrey, 1994, pp. 109–134]) it might well be possible to fruitfully use the metric for estimation as well.

The results look very promising, but some work still needs to be done. First, data about IT infrastructural project containing at least the IEP size drivers and the effort consumed by the project needs to be collected. This collected database will be used to calibrate the parameters in the Infrastructure Effort Point model and to analyse how good Infrastructure Effort Points correlate with the real expended effort.
Chapter 8

Influence of Research Methodology on Research Outcomes

For the state-of-practice in software engineering to improve, first academia and industry need to gain a deeper insight into the working of software processes and the use of software engineering tools. Secondly, industry needs to apply the insights, methods and tools generated by software engineering research.

Software engineering researchers will need to empirically investigate the companies are involved in software development, in order to gain insight into the actual process of software development and to enable industrial software engineers to decide which tool or method will work in their context. Only this kind of research can provide sound evidence about the effectiveness of methods and tools in practice.

Collecting data in real projects is difficult, whether this is to lay bare the dynamics of software development or to demonstrate the effectiveness of a method or tool. Collecting objective data (metrics) to measure the attainment of goals of a software organisation is already hard. Defining useful metrics to measure the causes of the (failure) to fulfil those organisational goals is even more difficult, as the diversity of potential causes makes their measurement illusive. In this article we describe a method to select useful software metrics based on findings from qualitative research.

This chapter is based on a paper presented at the 12th Doctoral Consortium of the Conference on Advanced Information Systems Engineering (DC CAiSE 2005) [Schalken et al., 2005] and on a manuscript that has been submitted for publication [Schalken and van Vliet, 2007].

8.1 Introduction

In software engineering, there is a need for better tools and methods to develop software systems. However, industry is slow to pick up improvements suggested by academia. Redwine and Riddle show in their study that on average acceptance of software technology by industry takes 10 to 15 years [Redwine and Riddle, 1985].
In their model of software technology maturation Redwine and Riddle state that substantial evidence on the value and applicability of a technology is needed before it will be accepted by industry [Redwine and Riddle, 1985]. In the software engineering community there is a growing awareness for the need of empirical validation of the effectiveness of proposed methods and tools. Without proper empirical validation one is unable to assess the effectiveness of the proposed tools and methods in a real-life setting. The problem in the software engineering discipline is that there is no accepted standard for what constitute suitable methods to gather the required evidence [Shaw, 2003] to convince both academic software engineering researchers and software engineers in practice. Although practitioners rarely directly question the evidence supporting a method or technique, the continuing religious wars regarding tools and methods could well be a consequence of the lack of solid evidence on the effectiveness of the proposed methods.

The collection of data (such as metrics) about the products and processes of an IT organisation, has become more and more a prerequisite for relevant scientific inquiry. It has also long been recognised as a crucial for an organisation’s success, as measurement is either seen as a distinguishing feature of mature practices or as driving force for improvement itself. In the Capability Maturity Model [Paulk et al., 1995] measurement ensures the control and improvement of any practice, and in the Quality Improvement Paradigm [Basili et al., 1994a] measurement drives organisational change.

Despite the perceived value of software measurement, many organisations and researchers have troubles to successfully collect and use software metrics. Although the majority of published case studies regarding software measurement present success stories, it is known that failures of a program tend to be kept silent in industry. This leads to a publication bias towards positive results. Briand, Differding and Rombach stated that “[d]espite significant progress in the last 15 years, implementing a successful program for software development is still a challenging undertaking” [Briand et al., 1997]. This statement is supported by studies regarding the successfulness of metrics programs. One study [Hetzel, 1990, cited in [Daskalantonakis, 1992]] showed that as few as 10% of the industry classified metrics programs as positive and having created enthusiasm, another study [Rubin, 1991, cited in [Daskalantonakis, 1992]] showed that two out of three metrics initiatives are terminated before two years have passed.

To improve the success of an IT metrics program, researchers have proposed numerous suggestions with respect to the implementation of a software metrics program [Gopal et al., 2002, Hall and Fenton, 1997, Briand et al., 1997] and the focus of the metrics program [Niessink and van Vliet, 1999].

Most advice proposes a goal-directed approach to the collection of software metrics. The goal-directed approach, made popular by the Goal Question Metric approach [Basili et al., 1994b, van Solingen and Berghout, 1999], structures the collection of data around the goals of the organisation. Although this approach is successful in diagnosing whether the process operates efficiently and/or effective, it is less successful in diagnosing the causes of their (mal-)performance.

8.1.1 Evidence-Based Software Engineering

Historically the emphasis of software engineering research has been on the construction of new tools and the invention of new development methods, not on the empirical evaluation of methods and tools in complex, industrial settings. The social sciences have long
struggled to find acceptable methods to perform research in complex settings and have developed a substantial body of knowledge regarding the methods of scientific enquiry. However, even in the empirically well-established disciplines such as social sciences, there is controversy over what methods of scientific enquiry are acceptable. On top of that, it is not always straightforward to apply research methods of the social sciences on software engineering research (see for example [Miller, 2004, Jørgenson and Sjøberg, 2004]).

Barbara Kitchenham et al. have suggested that the application of the evidence-based practice paradigm to software engineering offers an opportunity to improve the quality of current empirical research and to improve the diffusion of research results to practitioners [Kitchenham et al., 2004]. Evidence-based software engineering has been inspired by evidence-based medicine, which has had substantial success in achieving the goal of transferring research results to practice in the medical domain.

Unfortunately, there are a few serious problems related to the application of the evidence-based practice paradigm to software engineering. One of the most significant problems has to do with the inability to consistently use randomized, double-blind experiments within empirical software engineering research.

During the design of the holistic studies of the efficiency benefits of SPI (c.f. Chap. 3) and the design of study of the benefits of facilitated workshops (c.f. Chap. 4), questions arose about the relative merits of different research strategies (such as experiment, survey, archival analysis, history, or case study [Yin, 2003, p. 5]).

When choosing research methods, it seems as if one always needs to make a trade-off between relevance and rigor [Davison et al., 2004]. Interpretative case studies [Klein and Myers, 1999] and action research [Davison et al., 2004] and other interpretative approaches can give deep, relevant insights into what happens during a software engineering project. These research strategies are however often criticized for lacking sufficient scientific rigor (even with the guiding principles described by Klein & Meyers [Klein and Myers, 1999] and Davidson et al. [Davison et al., 2004]).

Experiments (and to a lesser degree quasi-experiments) on the other hand excel in scientific rigor, but the results of experiments with objective measurements often merely prove what is already known to be true by practitioners in the field. And when surprising results do appear, software engineering experiments often do not provide sufficient information to validate or explain all the results [Karahasanovic et al., 2005].

At the same time, choosing research methods is also a trade-off between internal and external validity [Cook and Campbell, 1979]. When using interpretive research, one can be rather confident that the obtained data really is related to the intended constructs (construct validity) and the deeper insights gained through interpretative research will usually make clear to the researcher if the theory will be applicable to other organizations. This advantage should be weighed against the disadvantages of subjectivism (other researchers can have a different interpretation of the same events and perhaps even the same data) and the risk of inferring causal relations where in reality none exist; the human mind is not good at distinguishing causal relationships and random correlations, one always runs the risk that the observed phenomenon could not be explained by chance alone.

Experiments and quasi-experiments with objective measures are however also not without problems. The measurement definitions can be contradictory and often be explained in different ways (e.g., a book over 100 pages is needed to define a standard for consistently counting lines of code), leading to differences in observed data, just because
the measurement procedures differed. This is even more problematic when comparing
data between companies (such as in meta-analysis or benchmarks) as differences in mea-
surement procedures will be even larger.

Another severe problem with experiments and quasi-experiments is that one cannot
always be sure that the right operationalization of the variables is chosen. E.g., is de-
veloped function points per hour a good measure of productivity or should quality also be
taken into account?

A procedure to improve the validity of experiments is to control as many environ-
mental factors as possible and to use random double-blind assignment (in which both re-
searcher and subject do not know to which treatment group they are assigned). Although
this method works well for evidence-based practice in certain medical disciplines, soft-
ware engineering is a skill-based profession in which the professionals will have to carry
out the assignment. This makes double blinding impossible, the software engineer will
always know what method he is practicing (which could influence the results).

The skill-based aspect also makes comparison between treatments more complicated;
a pill can only be applied in a single manner, whereas the application of a skill can
differ vastly between individuals [Wentem et al., 2003]. And with complex, com-
posite methods (such as Extreme Programming) not all organizations will employ the full
method [Williams et al., 2004].

8.1.2 Epistemological Frameworks

As no silver bullet has (yet) been found to solve all research methodological difficulties
for software engineering research, the researcher has to choose a research approach that is
appropriate for his or her problem at hand. The research approach has to include at least
the research strategy (such as experiment, survey, archival analysis, history or case study
[Yin, 2003, p. 5]), the research design (which experimental subjects are selected and how
are treatments assigned to the subjects), data collection procedures (what to measure
and how to measure it) and analysis methods (such as regression analysis, ANOVA or
grounded theory [Glaser and Strauss, 1967]).

The research approach has to be adapted to the properties of the research problem at
hand: the reason for performing the study and the context in which the research will be
performed. The reason for performing the study, also called the research motive, has an
impact on the research question. If a research project is undertaken as a critique on de-
velopment practices the research question will most likely differ from a research project
that is motivated to improve the state of development practice. Different research motives
will result in different research questions for the same problem domain. These different
research questions will in turn lead to different research approaches to investigate similar
phenomena. Furthermore, properties of the context in which the research project will be
performed will have a profound impact on the research approach taken. Performing long
case studies with students in an academic setting might prove to be difficult, whereas in
industry experiments might be harder to perform. Therefore the research approach must
be suitable for the context of the study.

The research approach chosen by the researcher is not only influenced by the prop-
erties of the research problem, but also by the researcher’s notion of knowledge. “Epis-
temological assumptions decide what is to count as acceptable truth by specifying the
criteria and process of assessing truth claims. …Methodological assumptions indicate
the research methods deemed appropriate for the gathering of valid evidence.” [Chua, 1986].

It would be beneficial to develop frameworks to document how epistemological assumptions, methodological assumptions and properties of the research problem at hand have guided the selection of a research approach. Most likely software engineering researchers have reached implicit inter-subjective agreements regarding the selection of research approaches. Descriptive frameworks could be useful to make this consensus explicit, so that the consensus can be translated into methodological guidelines for beginning software engineering researchers.

8.1.3 Incremental Approach

An incremental research methodology could offer a viable alternative to a single research method that solves every researchers methodological problem. When we accept that insights into real software development projects cannot be found when we limit ourselves to restricted laboratory experiments, we can look for methods that will help understand data gathered in the illusive domain of (non-experimental) field research.

We propose a framework to guide non-experimental investigations that can give insights into the reasons of software engineering processes (in-)efficiency and effectiveness. The framework guides the collection and interpretation of both qualitative and quantitative evidence regarding software engineering processes. The framework consists of two independent cycles of qualitative investigation and quantitative research. On a higher level, the cycles of qualitative and quantitative investigation are interconnected and together the cycles provide a reliable and efficient process of evidence collection that can be used to improve the software engineering process.

Efficient investigations in the elusive domain of software development require us to take both quantitative and qualitative evidence into account. As the required effort and therefore cost of collecting quantitative evidence is high, quantitative investigations take only a limited number of factors into account.

Although a lot has been learned about the success factors for software projects, there are still a lot of elusive causes for the success or failure of a software project. If we limit ourselves to investigating just the causes that can be detected by the collected quantitative evidence, we run the risk of missing important causes of failure.

Qualitative data can also be used to discover hypotheses that explain the efficiency of an IT department. Because these hypotheses are based on qualitative data instead of mere gut feeling, they are grounded in reality. And because qualitative data is easier to collect, one can take a broad range of issues into account.

The qualitative cycle of investigation leads to insights into the engineering process. The qualitative cycle can be based on a variety of rich information sources that are not necessarily irrefutable, but at least credible. The analysis of this qualitative data leads to presumed causes of organisational success (e.g., indications that the use of test-tooling leads to better quality or indications that changes in project management lead to slipping schedules).

Sometimes the insights generated by the qualitative cycle of investigation will warrant immediate corrective action. More often the assumptions about the software engineering process will not be deemed solid enough to justify direct action. In this case the assumptions can be tested by collecting additional data with a more rigorous approach.
This focused collection of more rigorous evidence—such as surveys, product and process metrics—and subsequent analysis takes place in the quantitative cycle of investigation. The more rigorous evidence obtained by the investigation can then be analyzed to confirm or reject the hypothesized causes for project failure or success. The quantitative cycle resembles normal metric programs, but (in contrast to normal metric programs) questions are raised based on preliminary evidence instead of instinct or gut feelings.

In the remainder of this chapter we provide details on the research framework based on the qualitative and quantitative cycle and we illustrate this research framework by applying it to the study of project post-mortem evaluations.

8.1.4 Outline

The remainder of this chapter is structured as follows: in Sect. 8.2 we explain why collected software metrics are so hard to interpret and why it is so difficult to apply the insights gained by the collection of data. In Sect. 8.3, we explain how qualitative data can be used at the start of an investigation to generate grounded hypotheses and how one can strengthen the validity of the hypotheses with the collection of additional software metrics. Finally, in Sect. 8.5 we end with some concluding remarks.

8.2 Rationale

Although “the importance of measurement and its role in better management practices is widely acknowledged” [Abran and Moore, 2004, p. 8-6], it is often unclear which metrics should be collected. In this section we explain the specific problems associated with the collection of suitable metrics and explain why less formal evidence (e.g. post-mortem analysis) can be a valuable source of information to steer the measurement process.

8.2.1 Decisions Based on Evidence

Rational decisions should be the basis for the management of software development and maintenance, especially when the decisions do not have impact on a single project but on the organisation, processes or tooling of all development and maintenance projects. Rational decision making should be fact-based, meaning that these decisions should be based on the best available evidence from both the organisation itself and from external sources (such as academia and professional associations).

Most actions of managers of software development and maintenance that affect the entire organisation and that require rational decision making can be classified into three categories: 1. problem definition, in which the area of management attention is set. Problem definition involves the selection of organisational units or process areas whose performance is seen as unsatisfactory or whose performance leaves room for significant improvement. The problem definition is fact-based when empirical evidence (from either benchmarks or assessments) is used to substantiate the under performance of an organisational unit or process area. 2. solution selection, in which the solution (or solutions) to the perceived problem is selected from all available alternatives. Examples of potential solutions are methodologies, work processes, tools and changes in the organisation. Fact-based solution selection should start with the collection and analysis of evidence and insights from external sources (academic, professional or otherwise) on the effects to be
expected of the proposed solution, such as a reduced defect-rate and higher productivity. Once the collection of data has been completed, the selection of the solution should be based on the expected effects of the solution. 3. *solution evaluation*, in situations where the stakes are high or when the evidence on potential solutions is inconclusive, management can choose to evaluate a solution. Solutions can be evaluated after a field trial or after a full implementation. Fact-based evaluations are based on collection and subsequent analysis of data on the effects of a solution. Most commonly data from before the solution (or the so-called baseline situation) is compared with data with the solution in place.

In essence this is an application of the Plan-Do-Check-Act cycle [Shewhart, 1939] to the software development and maintenance process.

In this chapter we will mainly focus on rational decision making in the categories problem definition and solution evaluation. For rational solution selection, evidence is not gathered from the own organisation, but instead from external sources. The gathering, analysis and application of prior research to a practitioner’s situation falls outside the scope of this chapter. The interested reader is referred to [Kitchenham et al., 2004] for more information on this type of decision making.

### 8.2.2 Different Forms of Data

Empirical evidence is data about the organisation that can be used to support or reject statements regarding the existence of problems (or opportunities for improvement) and statements regarding the effectiveness of a solution. This data can come in different forms. For the interpretation of this data it is useful to differentiate data among two axis: quantitative data vs. qualitative data and data on project factors vs. data on success factors.

Evidence about software development and maintenance differs in its level of rigour and its objectivity. Evidence can be rigorous, quantitative evidence or of a more informal, qualitative nature. Although no crisp boundaries can be drawn, we identify four classes on the continuum that ranges from qualitative to quantitative evidence:

1. as *metrics*, numerical data that has been obtained through rigidly-defined, objective measurement procedures (e.g. function point counts, defect counts). In Chaps. 3 and 4 this kind of data is used to calculate the productivity gains.

2. as *ratings* (or called subjective metrics), judgements by either experts or stakeholders on a specific part of the project or product (e.g. perceived timeliness and perceived quality on a five-point scale). In Chap. 4 this kind of data is used to decide whether facilitated workshops increase customer satisfaction and in Chap. 6 this data is used to validate the analyses from Chap. 5.

3. as *semi-structured information*, information that has been captured in documents which contain answers to open-ended questions (e.g. project post-mortem documents, risk logs). The exploratory data analysis in Chap. 5 is based on semi-structured information.

4. as *unstructured information*, information that has been captured in documents that have no common structure (e.g. email messages).
The term factor is a general term for aspects of projects, tasks or processes we could study. The collection of information on a factor would lead to data. Factors can be distinguished between project factors and success factors. Definitions of these terms can be found in Chap. 5.

### 8.2.3 Metrics

The collection of metrics (and to a lesser degree ratings) is the gold standard for evidence. It has become the gold standard, because of the high validity of the evidence and the ease with which the evidence can be interpreted, once it has been collected.

Unfortunately the collection of metrics and ratings on even a few project and success factors requires a significant amount of effort. This makes the collection of metrics on a broad range of project and success factors infeasible. Organisations that have the maturity to collect metrics typically collect a small set of metrics that covers only a part of the development process. Tom DeMarco advocated that “you cannot control what you cannot measure” [DeMarco, 1982], however it is just as impossible to measure every factor in an IT project as it is to control every factor in such an IT project.

Overly strict adherence to Tom DeMarco’s advice could lead to a blinded view on IT projects. This is because metrics and ratings only provide information on factors that we are actively investigating. E.g. if we do not measure requirements creep, we will never know if and how requirements creep is affecting our productivity. The other side of the coin of not relying on less rigorous data (i.e. semi-and unstructured information) is that you will not notice what is not covered by the structured feedback.

During the problem definition phase organisations attempt to determine which project factors contribute most to the failure to achieve important success factors. The selection of appropriate quantitative goals (the success factors) for an organisation is already difficult [van Solingen and Berghout, 1999, p. 11]. Still, at least at the most global level, the organisation’s goals should be clear to the decision makers.

Data about the success of a project or organisation can only be interpreted when it can be related to the process that lead to this result (e.g., the testing approach used). When one is performing controlled field trials or a controlled experiment, the cause of change can be identified. However when one is studying day-to-day operations instead, data collection should also focus on collecting data about potential causes for the results.

The factors that influence these success factors are even more difficult to identify than the success factors themselves. Relations between project factors and success factors are not obvious and sometimes human intuition about the relation between cause and effect are just plain wrong. The lack of overview on the potential causes for success and failure, make the problem definition a hard one.

Solution evaluation is in principle a less complex problem, as it allows for formal experiments and controlled field trials. Unfortunately few organisations have the resources or the competence to use the laboratory procedures required to evaluate a solution. And when organisations do not employ sufficient experimental control (through isolation, randomisation or matching) the interpretation of these (quasi-)experiments is fraught with the same difficulties as the interpretation of data regarding problem definition.

Summarising, without a clear hypothesis to guide the research and without full experimental control, the collection of metrics and ratings might not alway be the most effective data collection method to gain insight about the development and maintenance
8.3. Analysis Method

In the previous section we have explained that qualitative evidence can be useful to guide the collection of quantitative evidence. What is missing is how evidence, or packaged information, from qualitative and quantitative sources can be integrated into a single view and how qualitative evidence can steer the collection of (more dependable) quantitative
evidence. In this section we propose a process framework to help guide the integration of qualitative and quantitative evidence.

The evidence collection framework consists of two separate, but interlocking cycles: the exploratory and confirmatory cycles of evidence collection. The cycles are interlocking, because the exploratory cycle continuously generates hypotheses about the software engineering process (both development and maintenance) that can be validated or rejected by the confirmatory cycle. The framework is displayed in Fig. 8.1.

In the exploratory cycle of the empirical investigation we first collect less formal evidence. This collection of evidence and subsequent analysis leads to preliminary insights. These preliminary insights provide insight into which potentially interesting relationships between causes and results warrant a more thorough investigation.

Having selected the potentially interesting empirical relationships based on preliminary insights, we start the collection of more formal and rigorous evidence in the confirmatory phase of the research. The definition of and collection of these metrics lead to more rigorous evidence to support the preliminary findings of the exploratory phase.

As the cycles demonstrate, the investigation process should not be seen as two phases in a sequential processes, but instead as two iterative cycles that inform each other but run independently.

The selection of interesting hypotheses and the subsequent selection of metrics to reject or corroborate those interesting hypotheses constitute a flow of information from the exploratory phase into the confirmatory phase. The formal and rigorous data collection procedures (e.g. metrics and ratings instead of semi-and unstructured information) that are typically used in the confirmatory phase have a higher validity and therefore provide more credible evidence on whether to accept or reject a certain hypothesis.

After a confirmatory cycle, it should have become clear whether the initial hypotheses remain plausible or need to be rejected. This acceptance or rejection of hypotheses provides a flow of information to the management and a flow of information to the researchers and practitioners that are involved in the exploratory cycle. Good hypotheses, that have been corroborated by confirmatory evidence, should allow management to take appropriate action to improve the engineering process. But the corroboration or rejection also provides information to researchers in the exploratory cycle about the reliability of their conclusions.

Figure 8.1: Exploratory and confirmatory iterations in empirical software engineering
8.3.1 Exploratory Cycle

When determining which part of the engineering process leaves room for improvement (the so-called problem definition phase), managers and specialists attempt to figure out which areas of the engineering process have problematic performance or leave room for even better performance. One could directly start the data collection process by measuring a few standard indicators (e.g. number of reported defects and number of reported defects) to determine the performance of the organisation as a whole.

Jumping to the data collection process often occurs when assumptions exist with management about the cause of the problem. Frequently these assumptions are not made explicit by these managers. If one can resist the temptation to start measuring without explicit assumptions, one could prevent measurement of irrelevant attributes and perhaps even uncover the mechanisms that have an influence on the success factors.

The collection of measurements can function as blinders in the data collection process, because they focus on only very few specific factors of the engineering process. We can overcome these blinders by taking alternative, less rigorous sources of evidence into account.

During the exploratory cycle, we take these alternative sources of data into account. The alternative pieces of evidence provide a richer, more grounded picture of the actual engineering process. These pieces of evidence sometimes paint a blurred picture in which causes and effect might be intertwined, but they prove to be a valuable source for hypotheses to improve the engineering process.

Examples of these alternative sources of data can be found in documents that are already present in the organisation, such as: project post-mortem evaluation reports, defect reports, records of customer service calls and process improvement requests. But the data could also be elicited by analysts through consultation of experts, panel discussions, questionnaires and idea boxes.

Although these alternative pieces of evidence lack in their rigour and objectivity, they have their semi-structuredness as a valuable advantage. Where specific, closed questions (such as ratings) are able to provide specific information, open questions can provide information that one was not expecting. This unexpected information or feedback may give important clues on what is causing the organisation’s success or failure.

To add more structure to the exploratory cycle, we drew up a process model that consists of the following steps:

unstructured observations By merely observing unstructured and semi-structured data recurring patterns become clear.

intuition By applying common sense and knowledge of the situation, these recurring patterns can be named and ordered into a conceptual framework.

structured observations This conceptual framework serves as a lens with which more structured observations are made.

analysis These observations can be related to the occurrence of desirable or unwanted events, by observing the organisation’s goals.

insight These patterns, once analysed, can be interpreted for their meaning (can this be a causal relationship) and relevance to practice.
Although semi-structured and unstructured sources of data about the development process can provide rich and valuable insights, there is little guidance on how to perform the required packaging and consolidation of this data. And even the guidance and methodologies that exists are less clear and leave more ambiguity than most quantitative methods. The cause why packaging of qualitative information is problematic might lie in the fact that the information is very diverse and non-numerical in nature.

Methods that can help in packaging qualitative data are, amongst others, content analysis (c.f. [Neuendorf, 2001]) and Grounded Theory [Glaser and Strauss, 1967, Strauss and Corbin, 1990]. These methods add structure to the qualitative data.

Chapters 5 and 6 are an example of how the two interlocking cycles of exploration and confirmation work. In Chap. 5 we proposed a five-step method to explore empirical relations between quantitative data in the project database and qualitative data resident in the organisation’s project evaluation database. The goal of the research was to identify areas in the engineering process that have potential for improvement and to improve the post-mortem process itself.

The analysis method for project post-mortem reviews allowed the use of post-mortem project reports (in natural language) as an additional source of data. The five steps of the analysis method were: 1. identify success factors, 2. select project evaluations, 3. identify project factors, 4. interpret project evaluations, 5. analyse correlations.

The concept hierarchy trees produced by the method, allowed us to translate relevant qualitative information in the post-mortem project reports into quantitative information. This quantitative information was subsequently statistically analysed to discover correlations between (project) factors in the post-mortem reports and (success) factors in the project database.

The analysis steps from the example can be mapped to the exploratory cycle in the knowledge discovery framework, as is shown in Fig. 8.2.

It is important to notice that once these insights are obtained one does not stop. Certain insights might lead to suggestions that are relatively risk-free and easy to implement, so that they can directly be converted into action.

The validity of other insights and hypotheses might not be so clear or the solutions that could be based on those hypotheses might be so costly or risky that a more thorough review is called for. These insights are then transferred to the confirmatory cycle, where

---

**Figure 8.2:** Mapping of post-mortem analysis method to exploratory phase
the hypotheses are tested using more rigorous evidence.

8.3.2 Confirmatory Cycle

When the exploratory cycle has provided us with insight, whose validity is not crystal-clear or whose impact is far-reaching, we can select these insights for closer study. The insights we want to verify will call for the collection of data on certain aspects of the engineering process or products. We call this selection process metrics selection.

Having selected the hypotheses to test and the associated variables required to test these hypotheses, we proceed with the following process:

metrics definition Having identified which aspects are of interest, we next need to determine which method to quantify these aspects will be used.

E.g. having defined productivity as a potential problem, we need to study the amount of effort spent and the amount of product produced. This leaves us with the question how to measure the amount of product produced. Usually we will use software size metrics to determine the amount of product produced. Lines of code, function points and COSMIC full-function points are all examples of software size metrics.

Which metric or rating scale depends on the required accuracy, the area in which the metric is used and the effort that can be spent on the data collection. E.g. function points might be a better measure for functional size, however the alloted time allows one only to obtain the amount of lines of code produced.

collection of metrics Once the metrics and rating scales have been defined, data needs to be collected. During this phase one needs to keep an eye on the selection of objects to be measured. E.g. if one decides to measure only the productivity of small projects, one cannot say anything about the productivity of large projects. Other selection criteria, such as self-selection, might lead to other distortions of the portrayed picture.

To make sure the results can be reliably measured, one needs to make the selection process of the objects under study explicit.

data analysis To test the hypothesis under study, we need to analyse the collected data using statistical procedures. More on these statistical procedures can be found in [Kitchenham et al., 1999].

evidence After the data has been statistically analysed, one needs to draw conclusions based on the analysed data.

8.3.3 Validation of Insights

The results of the confirmatory cycle, if conclusive, leads to a validation or rejection of the hypotheses that were raised by the exploratory research cycle. We assume that appropriate hypotheses have been chosen to be confirmed by a confirmatory cycle. By an appropriate hypothesis, we mean that the confirmed (or rejected) hypothesis can be translated into a set of required set actions. Once the hypothesis has been confirmed it should therefore not be hard to decide the appropriate management response.
The use of the information regarding the exploratory hypotheses is however not restricted to the guidance of management action. The outcomes of the confirmatory cycle can also be used to validate the exploratory discovery process itself.

The confirmatory cycle can be seen as a replication of the exploratory research cycle, with more sensitive research instruments. The second study (being of high quality) can provide insights into the validity of the first study. The underlying philosophy is that if no large changes occur in the organisation and no directed improvement actions are executed, stable empirical relationships will persist over time. If the more reliable evidence leads to different conclusions than the less reliable evidence, probably the conclusions based on the less reliable evidence were wrong and therefore the process that produced these conclusions was also wrong.

Therefore we can gain information on the reliability of the exploratory research cycles by examining the correspondence between the results of the exploratory and confirmatory cycles. For this correspondence it is however crucial that no significant changes (organisational or otherwise) occur between the time of exploration and confirmation, otherwise no information regarding the validity of the exploratory cycle can be obtained.

To measure the correspondence between the results of the exploratory and confirmatory cycles, we could examine the correlations between the results of the first and the second cycle, or we could cast the problem into the form of a inter-rater reliability problem (the two cycles being seen as two independent raters of an occurring phenomenon). High correlation coefficients or high inter-rater reliability indices are an indication that both the exploratory and confirmatory cycle are measuring the same underlying construct. In multi-method research, correlation between two independent sources of data that purvey to measure the same phenomenon is seen as evidence of the validity of this evidence, more in specific the convergent validity [Campbell and Fiske, 1959], and is measured by correlation coefficients.

When we want to analyse the validity of our five-step method for post-mortem analysis, we need to compare the correlation matrices that have been produced by the analyses to see how well the correlations between project factors and success factors match between the matrix based on exploratory research and the matrix based on more rigorous confirmatory research. High correspondence between the correlation matrix of the exploratory and confirmatory cycle would build confidence in the method for post-mortem analysis.

Regarding the use of correlation coefficient to judge the convergent validity of the post-mortem analysis method, we would like add the following: If we were in a position to calculate the correlation matrices with the same correlation coefficients, we were able to used stronger validity statistics, such as Cohen’s Kappa [Cohen, 1960] or Krippendorff’s Alpha [Krippendorff, 1980].

8.4 Related Work

Remarkably little related work exists in the field of research methodology for software engineering, the study of the research methods that are used by software engineering researchers. Only recently Shaw observed that the software engineering discipline has no accepted standard for what constitute suitable methods to gather the required evidence [Shaw, 2003]. Although much has been written about individual research methods (e.g.
how to perform an ANOVA test or how to design a randomized experiment) little research has been performed to study how a researcher can choose the best research approach for the research problem at hand.

Adrion started the analysis of software engineering research methods by classifying software engineering research in four distinctive categories [Adrion, 1993]: the scientific method, the engineering method, the empirical method and the analytical method.

Kitchenham provides some insights in the strengths and weaknesses of distinctive research approaches used in software engineering research [Kitchenham, 1996]. She however does not provide normative guidance on choosing a research approach for a given research problem nor does she compare the indicated use of a research method with its actual use by researchers.

Zelkowitz and Wallace [Zelkowitz and Wallace, 1998] and Tichy, Lukowit, Prechelt and Heinz [Tichy et al., 1995] take a different approach; instead of focusing on which research approach would be most appropriate for a certain research problem, they focus on the actual use of a research approach by computer scientists in practice and place less emphasis on what research method is best. By performing content analysis on published papers they are able to show which research approaches are used in successful research.

Shaw takes this approach a little further by comparing the research approaches from papers that have been accepted for a prestigious conference with the research approaches from papers that have been rejected [Shaw, 2003]. Is this manner she sheds some light on what are research approaches that are acceptable in the eyes of other researchers and what research approaches are clearly not acceptable.

Unfortunately no study has been found by the author that takes the full spectrum of questions into account regarding the choice of research approaches for software engineering research.

Others, such as Karahasanovic et al. [Karahasanovic et al., 2005] have also used qualitative data (such as feedback from developers) in addition to quantitative data (e.g. time to solve a problem) to gain knowledge about a problem. An overall framework to integrate this quantitative data with quantitative data in a research program is missing however.

### 8.5 Conclusions

We successfully applied the discovery framework to first analyse and subsequently validate hypotheses regarding the software engineering process at ABN AMRO Bank in Chaps. 3 and 4. On a meta-level we were also able to show that the results of the post-mortem analysis method can provide useful, but not perfect information.

The framework that we proposed in this chapter helps in structuring the data collection process regarding the causes of success or failure of a project. By the subsequent gathering of ever more stronger data, one can build reliable theories regarding the effectiveness of the organisation, processes and tools and at the same time prevent the collection of excessive amounts of data that would bring with it excessive costs.
Chapter 9

Empirical Software Engineering: a Personal Perspective

In the previous chapters I described the various research projects conducted during my work as a Ph.D. student. During this time I did not only learn new things about the effects of software process improvement, yet I also learnt the basics of performing empirical software engineering research. In this chapter I reflect on the lessons I had to learn. During the supervision of a M.Sc. thesis and discussions with fellow Ph.D. students, I noticed that fellow Ph.D. students and master students had to struggle to master similar lessons during the execution of their research.

Some of the insights into the empirical software engineering research methodology are novel enough that they warrant a more thorough description in this thesis (i.e. the statistical methods to compare productivity in Chap. 3 and the framework to integrate exploratory and confirmatory research Chap. 8). Other insights I gained are assumed to be general knowledge in the social sciences, yet these insights are either not widely known or not widely taught and applied in software engineering research.

In Sect. 9.1 I discuss why empirical research skills are relevant for software engineering professionals and in Sect. 9.2, I describe the five simple lessons regarding research methodology I learnt during my Ph.D. studies.

This chapter is based on a paper presented earlier at the 4th International Workshop on Empirical Software Engineering (WSESE-2006) [Schalken, 2006].

9.1 The use of Research Methodology Knowledge

These lessons learnt point to lacunas in the Artificial Intelligence curriculum I followed. These lacunas could very well be typical for most master programs in Computer Science-related topics. Some might falsely believe that empirical software research methodology is useful only for academics.

Evidence-based software engineering (EBSE) is “the means by which current best evidence from research can be integrated with practical experience and human values in the decision making process regarding the development and maintenance of software” [Kitchenham et al., 2004]. Knowledge of evidence-based software engineering
is therefore vital for the maturation of the IT profession.

A basic understanding of software engineering research methodology is required to understand evidence-based software engineering. Evidence from research cannot be interpreted nor assessed without basic knowledge of the process that brings about this evidence (i.e. the scientific method). Instruction in (evidence-based) software engineering is therefore incomplete without instruction in software engineering research methodology.

There is almost no literature available on evidence based software engineering education [Jørgenson et al., 2005] and literature on evidence based software engineering has only started to accumulate since 2004 [Kitchenham et al., 2004]. The lessons I learnt during my work as a Ph.D. student might be useful as an example for instructors to understand the lessons their students need to learn to become knowledgeable in the area of empirical software engineering.

9.2 Lessons Learnt

9.2.1 Research Methodology also for Software Engineers

Experimental procedures in the beta or hard sciences (e.g. physics and chemistry) are conceptually clear. In the beta sciences, researchers are able to directly manipulate the experimental condition which they want to study. At the same time the researchers can isolate the experiment in a laboratory setting, keeping all other external influences constant (e.g. temperature, humidity, polluting chemicals). By keeping the external influences constant the observed effect be caused by the manipulated experimental condition, as alternative explanations can be ruled out. Replicated experiments are likely to have the same outcomes in these settings (i.e. direct control over the experimental condition and isolation of environmental factors).

For the gamma or social sciences on the other hand, experimental procedures give rise to more conceptual problems. Social scientists are unable to manipulate many experimental conditions (e.g. manipulation of gender is infeasible) and social scientists cannot always keep external factors constant (c.f. [Cook and Campbell, 1979, ch. 1]). Even worse, people participating in experiments often behave differently from people outside an experiment, this effect is known as the Hawthorne effect [Franke and Kaul, 1978]. Therefore research methods are needed to decrease the chance that observed phenomenon is not caused by the experimental condition but by an alternative (unanticipated) cause. Research methods determine how the experiment needs to be conducted (experimental design) and how the results need to be analysed (statistical techniques).

Because conceptual problems in experimentation first surfaced in the social sciences, most literature on research methodology is written for social scientists. The orientation towards the social sciences makes literature on research methods not really accessible to most computer scientist. As a freshman, I took the mandatory course “Introduction to Empirical Methods” as part of the schooling in psychology. Yet I remembered little from this course, as I dismissed the subject as interesting, yet only applicable for psychologists.

At a first glance most software engineering problems look like the typical “beta science”-problems, e.g. the expressiveness and the non-ambiguity of a programming language can be proved. But after a more careful look many software engineering problems turn out to be “gamma science”-problems, e.g. we should not maintain that a pro-
gramming language is appropriate when it is so complex that programmers have troubles expressing their ideas in the programming language. This leads to my first lesson: research methodology is also of importance for (empirical) software engineers.

9.2.2 Research Context Matters

Direct manipulation, isolation of external influences and the study of relatively simple objects (e.g. a single molecule) enabled “beta scientists” to state empirical laws that are valid under broad conditions.

“Gamma scientists” on the other hand deal with rather complex objects (e.g. humans or institutions) and typically are unable to keep all external influences constant. The lack of control on external influences makes the validity of empirical theories from “gamma research” outside the context of study uncertain. Many of the empirical theories from “gamma scientists” typically provide partial explanations of observed phenomena, leaving parts of the phenomena unexplained. When these partial explanations do not account for the contextual influence, one does not know if the different contextual factors will have an effect on the phenomenon under scrutiny.

Empirical software engineering insights are usually also dependent on the context of research. One of the main motivations for me to start my Ph.D. studies was the controversy in the trade press between using disciplined methods (c.f. CMM [Paulk et al., 1995]) and agile methods (c.f. Extreme Programming [Beck, 1999]). As a practitioner I had to decide which approach (CMM or agile methods) to follow, based on (at that time) largely unsubstantiated improvement claims, which motivated me to perform research in empirical software engineering in order to provide evidence to decide which method is better.

In hindsight, the difference in opinions about the best software process might be explained by the different context in which the software projects were carried out [Boehm and Turner, 2003]. Large, complex projects typically gain more from the security of a disciplined process whereas smaller, simpler projects typically gain from the increased flexibility of an agile method.

This leads to second lesson I learnt: software engineering insights depend on the context of study. To prevent controversies such as the agile versus disciplined processes debate in the future, one should specify the context of each empirical study (c.f. [Kitchenham et al., 1999]).

9.2.3 Simple Statistics

After an experiment has been conducted and experimental observations have been obtained, the observations need to be interpreted before lessons can be learnt from the experiment. If the experimental observations are collected with a systematic procedure that attaches numbers or symbols to the observed phenomena, the observations are called measurements (c.f. [Fenton and Pfleeger, 1998, p. 7]). The interpretation of measurement data can be performed systematically using statistical techniques.

In the beginning, after my first practical exposure to statistics in my master thesis project, I was seduced to restrict most of my analyses to inferential statistics. Statistical techniques can be grouped in techniques that provide a condensed view of the data (i.e. descriptive statistics) and techniques that allow decisions to be based on the data (i.e. in-
ferential statistics). Inferential statistics can lull researchers into a false sense of security. Having applied the statistical procedure, the researcher bases his judgement solely on a p-value. Although the p-value does indicate that an effect is present in the data, it does not tell anything about the size of the effect nor the direction of the effect. This could lead incautious researchers to believe that their proposal improves the software development or maintenance process, where in practice it really deteriorates that process.

To prevent misinterpretation of the data, it is often advisable to first use descriptive statistics, such as scatter plots, contingency tables, summary statistics and confidence intervals. When potential patterns in the data have been found using descriptive statistics, inferential statistics can determine if the supposed patterns are really present or just random noise.

A second mistake I have made is the application of advanced statistical techniques (i.e. principal component analysis, multi-dimensional scaling and MANOVA [Stevens, 2002]) when simpler techniques would have sufficed. Advanced statistical techniques are powerful in finding effects in complicated experiments that measure results in multiple dimensions/variables. Unfortunately most researchers find it hard to create a mental picture of what these advanced techniques do and are sometimes not able to understand the underlying calculations.

The law of Pareto states that 80% of the benefits come from 20% of the effort. This law is also applicable to statistical analyses, with a correct experimental or quasi-experimental design [Cook and Campbell, 1979], most questions can be analysed using simple parametric statistical techniques (e.g. ANOVA and linear regression analysis [Peter et al., 1996]).

Measurement is not an absolute requirement for the interpretation of experiments. When measurement is infeasible or undesirable, qualitative techniques like grounded theory [Glaser and Strauss, 1967] are available to interpret the observations. The additional flexibility of qualitative techniques does usually come at a price of reduced objectivity and repeatability, which makes the conclusions based on qualitative research less reliable. In quite a number of cases these serious draw-backs from qualitative research are however outweighed by the improved relevance of these research efforts (c.f. [Lee, 1999]).

With a slight detour on qualitative research, this exposé leads to the third lesson I learnt: whenever possible visualise data before making inferences and in most cases simple statistical techniques are superior to advanced statistical techniques.

### 9.2.4 Real-life Data is Messy

I still remember the first time I analysed the data from my first experiment. The textbooks on statistical techniques quietly assumed that all measurement data is complete, correctly measured and on the right scale. Moreover most statistical textbooks assume that data conforms to the assumptions that the statistical tests place on them. As I quickly discovered during my first attempts to analyse the data, this is rarely the case.

In many experiments and most field experiments data is not so clean: data is missing, is incorrectly measured or recorded, some observations should not have been included in the sample and a vast range of other issues will be associated with the experimental data.

This lead to my fourth lesson: always inspect and clean your data before you run any analysis. During their education, students should not only receive basic training in
the application of statistical techniques when the data is clean, but they should also be exposed to dirty data and receive instructions on how to deal with outliers and incomplete data (c.f. [Allison, 2002]).

9.2.5 Why was the Data Collected

In most rational organisations, measurements are collected for a reason (this is the basic tenet of the Goal-Question-Metric approach [Basili et al., 1994b]). When performing field research it is imperative that you keep in mind the reason for which the data was collected.

I learnt this lesson the hard way. When trying to analyse the effect of process improvement on planning accuracy of a company, I noticed that the delta between planned effort and realised effort did not decrease with improved maturity, rather the delta remained at a constant low. When analysing the data with more scrutiny, I discovered that planned effort almost perfectly correlated with realised effort. It turned out that the company steered on staying within the planned (or budgeted) effort and the bonuses of employees depended on this conformance to planned effort. Employees used creative administrative tricks to ensure that the realised effort stayed within the planned effort (mostly by replanting the project at the end of the project). Similar effects have also been observed by Verhoef [Verhoef, 2007].

Software engineering researchers need to realise that software products and software development processes are so complex that it is infeasible to measure all aspects of them. If sufficient motives are provided to manipulate the data, one can be sure that there will be employees who will take advantage of this fact.

If on the other hand data is not collected for any apparent reason, the collected data is also in doubt. If the usefulness of the collected data is unclear, there will be little motivation to provide accurate information. Perhaps the best measurement data can be found in sources that are used by the practitioners themselves, but that are not used by management to steer the organisation.

The last lesson I learnt is that it is important to understand for which reasons the measurement data has been collected and to be alert when these motives can have a negative impact on the quality of the data.

9.3 Conclusions

In this chapter I provided an overview of five simple lessons I that learnt during my Ph.D. research in empirical software engineering. The first lesson I learnt was that although research methodology received most attention in the social sciences, it is also relevant for software engineering research. I also learnt that one should always explain the context of the findings of your research. In addition it is important to learn to deal with messy data and to use as simple statistical techniques as possible. Last but not least: never forget that the data that is used in a field research project might have been collected for totally different purposes and that this can have a serious impact on the validity of the results.

The lessons I learnt might seem like open doors, yet I have seen fellow Ph.D. students and master students struggle with the same problems. It might be interesting to improve instruction of empirical software engineering based on the problems experienced by graduate students and junior researchers in their research endeavours.
The lessons I learnt are not new, they are described in intermediate textbooks on statistics and software metrics. The difficult lies making students sensitive to these problems, so that they can recognise data validity issues when they arise and that they can apply appropriate statistical techniques when dealing with software metrics. The students should learn active data collection and analysis skills instead of passive knowledge about empirical software engineering.

What aggravates the problem is that authoritative textbooks in statistics and research design are usually aimed at social scientists or business researchers. Students need to translate the research examples drawn from the social sciences to software engineering context in order to understand and recognise the basic principles and lurking pitfalls.

Personally I do not believe that lectures in empirical software engineering theory alone will suffice to make students sensitive to the intricacies of empirical software engineering research. Instead instructors should combine instruction (in software measurement procedures, experimental design and basic statistics) with exposure to real-life problems. An approach to expose students to real-life problems is to let graduate students and junior researchers explain the challenges of their current empirical research projects. An alternative approach would be to let students perform practical research assignments in the area of empirical software engineering.
Chapter 10

Conclusions

This chapter describes the conclusion of my thesis research. The research started out as a project to measure the effects and return on investment of software process improvement, but evolved into a research project into the empirical techniques needed to identify these effects as well as techniques to identify areas that leave room for improvement.

During the course of my research it became clear that standard empirical research approaches could not be used to assess the effects of large software process improvement programs. Traditional empirical research tries to corroborate the existence of causal relationships between causes and their hypothesised effects by conducting and subsequently analysing experiments. Most experiments are setup according to well known research designs that help a researcher collect empirical that is relatively easy to analyse, provides unbiased, dependable results. Research can corroborate the existence of a causal link between cause and effect with experiments, because experiments minimize the chance that unknown factors (instead of the hypothesised cause) are responsible for the observed effect.

The strength of experiments lies in the fact that two strategies are combined to rule out these unknown factors (which are also called alternative explanations). Experiments are conducted in a more or less controlled environment isolates the study from disturbances that could also cause the desired effect. In addition, randomisation procedures in the experiment assign subjects randomly to the experimental treatment (the hypothesised cause) or to the comparison group that does not receive the experimental treatment (also called the control group). This randomisation helps to rule out (unknown) alternative explanations, because through randomisation both the experimental group and the control group will be roughly equally affected by these unknown factors. How scientifically sound the results of experiments might be, in terms of having high probability of arriving at the correct conclusions, it is unfortunately an idyllic view of the place of a researcher in society because it requires a researcher to have the authority to exercise experimental control.

A fitting example in the context of software process improvement of this idyll would be the following: in order to investigate the effects of software process improvement on productivity, one would either execute each project twice –once with the old development process and once with the new development process– to isolate the differences between the two projects, or one would need to be in charge of randomly selecting which project
will use which development process. In the first option the researcher would need to be in
the position to demand that all projects are carried out twice (and will therefore be twice
as expensive) or in the second option the researcher would need to have the authority to
force the use of a specific method by a project (and which organisation would knowingly
opt for a development process that is supposed to be less effective). A researcher might
be able to recreate an experiment with a small development project/assignment to test
certain ideas, but it would be prohibitively expensive to experiment with projects of a
realistic size.

Perhaps empirical software engineering researchers will need to accept that the im-
pact of their work is not significant enough to let society bear the costs of such extensive
research. Only when the stakes are high enough, such as in the medical profession, will
society allow us to perform the costly research to gain higher levels of confidence in our
results. Until society demands us to give assurances similar to those that for example
the Federal Drug Administration demands from pharmaceutical companies, if that time
will ever come, we will need to be content with research results that either miss the high-
est form of scientific rigidity (and therefore the highest chance that the conclusions are
indeed true) or that miss face value with practitioners (because the experiments were per-
formed under circumstances that are not similar to industrial settings). Without sufficient
resources (and therefore funds) we will never be able to escape the trade-off between the
relevance and the rigor of our scientific results [Lee, 1999].

In this research project, that started out as an investigation into the effects of software
process improvement, we have chosen the relevance of our results over the rigor from the
experiment. Still we tried to partially sidestep the relevance/rigor trade off by applying
statistical and qualitative techniques to the study of software process improvement. Al-
though one will never be free from the relevance/rigor trade-off, at least the techniques
presented in this thesis will provide some way to game the system.

10.1 Research Questions revisited

In Chap. 1 we stated with the following research questions:

- How can empirical data be used to strengthen a software process improvement
  initiative?

We subsequently divided this research question into the following three subquestions:

1. How can empirical data be used to get feedback about the effects of software pro-
   cess improvement?

2. How can empirical data be used to get feedback about the software development
   processes that still have room for improvement?

3. What are the quantitative effects of software process improvement on software
   development productivity?

This section provides an overview of the lessons we learnt during our research project.
10.1.1 How can empirical data be used to get feedback about the effects of software process improvement?

In attempting to calculate the savings gained by introducing software process improvement, we started out in Chap. 3 with the productivity indices that are a simple quotient of the size of the product delivered divided by the effort required to construct the product. When examining the results of this exercise, we found out that the resulting quotient is not normally distributed and so we are unable to use the test of the field of classical statistical inference (a result which will probably amaze some computer scientists, but which will be as expected for mathematicians who know that the quotient of two normally distributed stochasts will be Cauchy distributed).

Instead of abandoning the classical metric of productivity (the productivity index) and using non-parametric statistical inference, we opted to take a deeper look at the data. In order to compare similar projects with each other, one needs to take variable returns to scale into account. In other words one needs to accept the fact that smaller projects do not have the same productivity as large projects.

However if one takes a better look at the data, one sees that projects that are executed in one domain of the organisation differ from projects that are executed in another domain of the organisation. With the use of hierarchical linear models, one can also compensate for this.

When attempting to judge whether facilitated workshops have a positive effect on productivity when compared to traditional requirements engineering processes, one runs into the problem that projects that use a different methodology might be comparable from the outside (if one views the project as a black box) but that the projects are not comparable if one wants to compare phases of projects that do not have the same life cycle. Manually mapping the phases of a project to a reference model will help one to find which numbers can be meaningfully compared and which numbers cannot be compared.

The two studies show that obstacles can sometimes be overcome using advanced statistics and sometimes using background knowledge of the problem. To solve the problem of not being able to measure the effects of software process improvement when many COTS components are used to develop a system, we designed a metric to size the installation and configuration aspects of an IT project in Chap. 7. The metric we designed is called the Infrastructure Effort Point metric. The metric is an objective metric that can be used to measure the size of IT infrastructure projects.

Infrastructure Effort Points can help to assess the effectiveness of processes and techniques, making it a valuable tool for process improvement for organizations that deliver IT infrastructure. The Infrastructure Effort Point metric is unsuitable for the upfront estimation of a project’s schedule or costs, because the required information needed to size the project is available only at a late stage of the project.

10.1.2 How can empirical data be used to get feedback about the processes that still have room for improvement?

Chapter 5 provided a novel way, based on grounded theory, to analyse the content of written documents, such as project post-mortem documents. Using this analysis, we were able to find novel insights into the success and failure factors of an IT project. We
Conclusions

performed the method in two different settings, once in a large organisation that already
had a reasonably mature metrics program in place and once in a small organisation, which
had previous experience with empirical software engineering research, yet that lacked a
mature metrics program.

In the subsequent chapter, Chap. 6, we took a look at the the validity of the proposed
method, by triangulating the results of the study with new data. It became apparent
through this triangulation that it is important to look at the quality of the data. When
data quality is mediocre, results cannot be repeated (and are therefore useless), otherwise
even with exploratory methods one can find some level of repetition in the results.

10.1.3 Integrating the pieces of the puzzle

During the process of arriving at the above findings, often the feeling crept up upon us
that doing research was not the planned, sequential, deductive process that social science
textbooks on empirical research portray us. Instead one often feels one should (or was to)
come back on assumptions and chosen research paths, which makes performing research
an iterative (or at least incremental) process that should be described as such, especially in
a textbook. Although the classical approaches are very good in communicating the results
of research (c.f. [Parnas and Clements, 1986] for an analogy in software engineering), but
not so good in guiding empirical field research.

Chapter 8 describes an iterative research cycle that is useful for organisations that
wish to improve their performance through empirical research. The iterative research
framework assumes that the organisations continuously collects information using broad
indicators that give an indication of the performance of their core processes. This con-
tinuous information gathering and analysis will lead to indications of problems and will
identify opportunities for improvement. These indicators will typically lack statistical
significance and therefore a second research cycle is started to confirm the validity of the
indicators or reject them as false positives. When a problem is confirmed, subsequent
improvement actions can be started.

This research cycle can also be seen as an integration of the techniques that are first
used to identify improvement areas which are subsequently verified. Understanding that
each phase in your research has different goals, makes it possible to accept the limitations
with respect to rigor that are inherent in many studies.

10.1.4 The benefits of SPI

Besides examining the strategies to determine the effects of software process improve-
ment, we also have some outcomes to report regarding the actual benefits of software
process improvement.

Facilitated Workshops

Facilitated workshops are a technique for structured meetings, led by a workshop facil-
itator, that are ideally suited to arrive at a consensus between participants of the work-
shop. For a software project to have change to become successful, the stakeholders in
the project should agree to what functionality the software product should and should not
provide. Dynamic Systems Development Method therefore uses the technique to gather
requirements.
When looking at the impact of facilitated workshops on the productivity of the requirements engineering, we can see no overall improvements. For smaller projects, requirements engineering using the traditional interviewing technique (inherited from Method/1) is more productive whereas for larger projects facilitated workshops are more productive. Using regression analysis, the turning point where Method/1 projects become less productive than DSDM projects with facilitated workshops has been determined at 171 function points.

Having analysed the effect of the requirements engineering technique (facilitated workshops versus Method/1 interviews) we are unable to discern a significant difference with respect to the duration of requirements engineering.

When we look at satisfaction with facilitated workshops, it appears that stakeholders are more satisfied about the results of Method/1 projects than they are about the results of DSDM projects. This was contrary to what we expected. A closer inspection of the individual aspects of satisfaction provides a potential explanation. Customer ratings of questions that are involved with expectancies on the project are higher for Method/1 than DSDM projects. The potential explanation is that customers have high expectancies of new technologies and tools, whereas project managers have seen numerous technologies fail before and are therefore less likely to be disappointed. When we look at the satisfaction with the clarity of the requirements (the end product of the requirement engineering phase) we see no discernible difference.

Overall maturity

We also compared the productivity of projects executed at ABN AMRO Bank under a CMM level 2 or level 3 regime and compare those with projects that have been executed under a immature CMM level 1 regime. We used the metric hours per function point as a measure for the productivity in software development.

From the study we have found clear evidence that CMM does increase the productivity of an organisation. We found a productivity increase of 20%. More planning and more attention to management and work processes do seem to have a positive effect on the productivity of the organisation. The improvements made in this study are smaller than found in certain similar studies, but we believe that this might be explained because in some studies small convenience samples are analysed instead of the productivity data on all projects in that organisation.

In Chap. 1 we noted that CMM-based software process improvement ideas have evolved in a time when linear software development methods dominated the methodological landscape. It was therefore uncertain if in the new context of agile development practices CMM would still lead to improvements. In our study we found a positive effect of the concurrent introduction of CMM-based software process improvement and an agile development method. This means that benchmark-based improvement models (such as the CMM model) and agile development methods are not necessary exclusive options for management, the two approaches can be combined.

Another important observation we made during our study is that one should be careful to avoid writing thick manuals to induce heavy processes. Instead a pragmatical approach to process improvement is required, where the benchmarking model is followed in spirit rather than to the last letter. Only then can (and are) software process improvement programs be beneficial.
10.2 Further Work

Although a lot of work has been done during the last four years, many open ends still lay waiting to be solved.

When looking at the effects of facilitated workshops on efficiency and reduced time-to-market, we did not have the data to take into account the number of stakeholders that needed to be involved, the amount of requirements creep (requirements that are changed during requirements engineering phase) and the amount of conflicting requirements that exist between the stakeholders. Kulk and Verhoef have studied requirements creep at the ABN AMRO Bank N.V. [Kulk and Verhoef, 2007] to assess the risk of the IT projects involved.

With this data one can paint a more accurate picture of the actual difficulty of the requirements elicitation and negotiation task. Using functional size as a measure for requirements engineering paints a part of the picture, but definitely not the whole picture.

In our study we used satisfaction as a measure for the quality of the requirements engineering process. Satisfaction is a possible metric for the quality of the requirements, but not the only one and definitely not the best. However it is a decent measure to determine requirements quality as can be determined in a short span of time. The real quality of requirements can only become apparent after, and sometimes long after, the system has been made operational. It is only then that we see a plethora of change requests for ill-conceived systems. A study that is able to follow the requirements of a system over a long period of time and that correlates unnecessary (foreseeable) requirements changes with the requirements engineering technique used should be able to give the definitive answer on the applicability of facilitated workshops.

Although the results for an Infrastructure Effort Point metric look very promising, still lots of work needs to be done. Currently only the feasibility of collecting the metric has been tested. First, data about IT infrastructural project containing at least the IEP size drivers and the effort consumed by the project needs to be collected. This collected database will be used to calibrate the parameters in the Infrastructure Effort Point model and to analyse how good Infrastructure Effort Points correlate with the real expended effort.

After the calibration of the IEP model, one should try one or more case studies to make sure the results of the measurements concur with the opinion of the practitioners, in other words the face validity will need to be tested.

Overall we see that there is still a lot of room for improvement when it comes to empirical software engineering. For a discipline that has its roots in Mathematics, it is not so difficult to understand why the constructive approach or engineering approach has been the prevailing paradigm in software engineering research. Current problems in the development and maintenance of software systems are however largely unrelated to the underlying technology, instead current problems have a more organisational and managerial nature.

The solution to these organisational and managerial problems will most likely not be found in more sophisticated technology. Instead we will need to rethink our techniques, work practices and methods. The evaluation of a new technique can only be performed in the real world, as proofs and prototypes are not suitable to demonstrate the effectiveness of new work practices and methods. Software engineering practitioners and researchers will need to learn this new empirical paradigm, which requires new skills.
Practitioners of tomorrow will need to be able to understand and assess empirical studies of software engineering before they can make fact-based decisions with respect to software development methods and tools. Only with these skills will IT professionals be able to choose IT solutions that fulfill not only the technological requirements of an organisation but also the other requirements that the organisation has.

Software engineering researchers of the future will need to learn not only how to invent new technological solutions, but they will also need to learn how to evaluate those solutions within a realistic setting. By focusing on the usefulness and applicability of software engineering research results, we will be able to bridge the chasm between state of the art software engineering and the current state of practice.

Before this ideal of evidence based software engineering is achieved and before it is common practice that software engineering professionals choose solutions based on best available evidence much still needs to be done. Software researchers will need to gain familiarity with the concepts and methods of empirical software engineering. Empirical methods known in other disciplines will need to be transferred to our field of study and new measurement procedures will need to be developed. As with most paradigm shifts in science, it can easily take a generation of practitioners and researchers before we arrive at this vision.
Samenvatting

Onze samenleving in het algemeen, en grote organisaties in het bijzonder, zijn meer en meer afhankelijk geworden van ICT systemen om efficiënt te kunnen functioneren. De toenemende afhankelijkheid aan deze systemen leidt tot grotere en complexere ICT systemen, die steeds duurder worden om te ontwikkelen en te onderhouden. Het merendeel van de kosten van deze ICT systemen bestaan niet uit hardware kosten (de computers en het netwerk), maar uit de benodigde arbeidsinspanning om de software van deze ICT系统 en te ontwikkelen, te implementeren en te verbeteren.

Met de toenemende complexiteit van software systemen, neemt ook de onvoorspelbaarheid van het ontwikkelen van die software toe. Projecten, die als doel hebben om de software systemen te ontwikkelen of te onderhouden, lopen een groot risico om te lang te duren, te veel te kosten en bovendien niet precies op te leveren wat er gevraagd werd.

Om bovengenoemde problemen aan te pakken kan een bedrijf de software ontwikkelers betere software gereedschappen geven om de software mee te maken, men kan de te ontwikkelen software vereenvoudigen (minder is soms meer), men kan beter software ontwikkelaars in dienst nemen, maar men kan ook de manier van de werken van de software ontwikkelaars en hun managers verbeteren.

De vakgebieden software procesverbetering (software proces improvement) en software methodologie houden zich bezig met de bestudering van de wijze waarop software ontwikkelaars (en hun managers) hun werk aanpakken. Terwijl onderzoek naar software methodes (het zogenaamde software methodologie onderzoek) kant-en-klare raamwerken, handleidingen en stappenplannen ontwikkelt om beter software te bouwen of te onderhouden, richt software procesverbetering onderzoek zich op de vraag hoe een IT organisatie meer volwassen wordt en wat karakteristieken zijn van volwassen IT organisaties.

Bij het meeste informatica onderzoek kan men na afloop van het onderzoek zien of de doelen bereikt zijn. Door het nieuw ontwikkelde software systeem te testen kan men de sterke en zwakke kanten van de oplossing relatief eenvoudig beoordelen. Immers: “een systeem doet het, of hij doet het niet”. Bij onderzoek naar software procesverbetering en software methodologie is de geldigheid van een methode of verbeterplan niet aan het papier af te lezen. De werkwijzen en instructies zullen zich in de praktijk moeten bewijzen, nadat de IT professionals zich de nieuwe vaardigheden hebben meester gemaakt. Bovendien is het niet eenvoudig vast te stellen of een methode of verbeterinitiatief echt gewerkt heeft. Men kan vast stellen of de medewerkers zich aan de voorschriften houden, maar men weet dan nog niet of de voorschriften ook echt een positief resultaat bieden. Ieder IT project is immers anders en dat maakt het vergelijken van projecten en werkmethodes lastig.
Een gevolg hiervan is, dat er binnen het vakgebied van de informatica geen consensus bestaat over de te volgen werkmethodes. Met hetzelfde ritme waarin technologische vernieuwingen elkaar opvolgen, volgen ook software methodes elkaar op. Al naargelang de heersende mode en de gebruiken binnen het bedrijf wordt er nu gekozen voor een flexibele, lichtgewicht methode, waarna later de meer gedisciplineerde aanpak volgt met meer nadruk op documentatie en officiële, standaardwerkprocessen. Tussen IT-ers kunnen er soms gepassioneerde discussies ontstaan over welke ontwikkelmethode het beste is en over het nut (of de onzin) van software procesverbetering.

Een bijkomend nadeel, voortvloeiend uit de onzekerheden met betrekking tot de doelmatigheid van software procesverbetering, is dat de bedrijven soms halverwege een verbeterprogramma stoppen, omdat er onduidelijkheid bestaat over de opbrengsten terwijl de kosten wel zichtbaar drukken op het ondernemingsresultaat. Het stoppen met software procesverbetering gebeurt dan terwijl er mogelijk wel een goed resultaat in het verschiet lag.

Gezien de grote maatschappelijke behoefte aan goede, betrouwbare en betaalbare software, is het niet wenselijk dat er onduidelijkheid bestaat over de doelmatigheid van ontwikkelmethodes en raamwerken voor procesverbetering.

Zoals gezegd lijken gaat het toetsen van de doelmatigheid van een software ontwikkelproces of van een raamwerk voor software procesverbetering totaal anders. In plaats van een beta-blik op de ICT, dient men een meer gedragswetenschappelijke bril op te zetten. Met behulp van het in kaart brengen van positieve en negatieve factoren, tijdens experimenten en pilots, krijgt men langzaam aan een beter beeld van de werking van een methode. In plaats van terug te vallen op wiskunde en logica (zoals dat een informaticus betaamt), heeft men slechts veronderstellingen en theorieën, die door middel van waarnemingen en statistiek onderbouwd of weerlegd kunnen worden.

In mijn onderzoek heb ik gekeken naar de effecten het uitvoering van een software procesverbetering stap en naar de effecten van de invoering van een software ontwikkelmethode. De ABN AMRO Bank heeft mij in staat gesteld om onderzoek te doen naar de effecten van de invoering van het Capability Maturity Model (CMM) en Dynamic Systems Development Method (DSDM). Het CMM is een bekend raamwerk voor software procesverbetering en DSDM is een moderne, lichtgewicht ontwikkelmethode. In het kader van het Inspiration programma heeft de ABN AMRO Bank deze methodes in de eigen organisatie geïntroduceerd.

Als resultaat van een eerder uitgevoerd verbeterprogramma, beschikte de ABN AMRO Bank over een gedetailleerde database met gegevens over de voltooid en actieve projecten. Op basis van deze projectinformatie is het merendeel van het onderzoek uitgevoerd.

In mijn onderzoek ben ik uitgegaan van drie onderzoeksvragen. Twee liggen vragen op het methodologische vlak en één ligt op het inhoudelijke vlak. De onderstaande onderzoeksvragen, op het gebied van het meten aan software processen, behandel ik in mijn proefschrift:

1. Hoe kan men empirische data gebruiken om feedback te krijgen over de effecten van software procesverbetering?

2. Hoe kan men empirische data gebruiken om te onderzoeken bij welke processen nog verbetering te halen valt en om te identificeren in welke oplossingsrichtingen men moet denken?
Onderzoeksvraag 1 heb ik verder geconcretiseerd door hem op te splitsen in de volgende twee deelvragen:

1.a Hoe kan men empirische data van verschillende soorten projecten vergelijkbaar maken?

1.b Hoe kan men bepaalde aspecten van het software ontwikkeltraject, waarvoor nog geen meetinstrumentarium bestaat, kwantificeren?

Ten slotte heb ik ook een concrete vraag op het gebied van software procesverbetering in mijn proefschrift behandeld, namelijk:

3. Wat zijn de kwantitatieve effecten van software procesverbetering op de productiviteit van de IT afdeling?

Een van de belangrijkste bevindingen van mijn onderzoek liggen op het gebied van de statistiek, die nodig is om de projectgegevens op een zinvolle wijze te analyseren. Door gebruik te maken van de techniek van Hierarchical Linear Models kan men verschillende ICT projecten, uitgevoerd in verschillende bedrijfsonderdelen, toch met elkaar vergelijken zonder dat men appels met peren vergelijkt. Door gebruik te maken van Box-Cox transformaties, ook een wiskundige techniek, kan men de storende invloed van verschillen in omvang van de projecten neutraliseren. Meer over deze technieken leest u in hoofdstuk 3.

Ook is er gekeken naar de invloed van de wijze van communiceren tussen IT-ers en opdrachtgevers. Men heeft, als onderdeel van de software ontwikkelmethode DSDM, een nieuw soort overlegstructuur geïntroduceerd: de facilitated workshop. Na onderzoek blijkt dat men facilitated workshops het beste in kan zetten op grote projecten, omdat voor kleinere projecten de oude methode tot een goedkoper resultaat leidt. De resultaten van dit onderzoek vindt u in hoofdstuk 4.

Behalve het onderzoeken van de bestaande cijferbronnen, zijn tijdens het onderzoek ook nieuwe bronnen aangeboord; er is een methode ontwikkeld om de effecten van procesverbetering in kaart te brengen wanneer men oplossingen op basis van bestaande software systemen implementeert. Het meetinstrument voor deze systemen, Infrastructure Effort Points wordt beschreven in hoofdstuk 7.

Daarnaast hebben we gekeken naar hoe we het beste informatie kunnen halen uit project evaluaties. Door ieder project na afloop te evalueren, ontstaat er een schat aan gegevens, die de organisatie zou kunnen helpen om in de toekomst beter te presteren. Het probleem is dat men de inzichten uit honderden geschreven evaluaties niet eenvoudig boven tafel haalt. Door gebruik te maken van Grounded Theory is er een methode ontwikkeld om deze inzichten, die diep in project evaluaties verborgen liggen, boven tafel te halen. Meer over deze methode leest u in hoofdstukken 5 en 6.

In hoofdstuk 8 staat te vinden hoe men om kan gaan met verschillende bronnen van informatie over een software ontwikkelproces. De ene keer krijgt men feedback vanuit de projectadministratie, dan uit de projectevaluaties, dan weer uit een onverwachtse nieuwe bron. Deze manier van onderzoeken en analyseren wijkt af van de methode waarop sociale wetenschappers hun experimenten inrichten. Om als professional effectief om te kunnen gaan met deze steeds wijzigende gegevens, is een onderzoekssraamwerk ontwikkeld dat iteratief gebruikt kan worden om steeds betere informatie te krijgen over
het software ontwikkelproces. Hierdoor kan het management steeds tijdig de processen bijsturen of aanpassen wanneer dat nodig is.

Uiteindelijk blijft de hamvraag: *Heeft procesverbetering geholpen bij de ABN AMRO Bank*? Het antwoord hierop is bevestigend: door naar het tweede volwassenheidsniveau van het CMM model te stijgen en door het invoeren van DSDM is de ABN AMRO Bank in staat om 20% effectiever te werken. Dat wil zeggen dat de ABN AMRO Bank in de nieuwe omgeving dezelfde software kan maken voor slechts 80% van de kosten.
Bibliography


SIKS Dissertation Series

1998

1998-01 Johan van den Akker (CWI)  
DEGAS - An Active, Temporal Database of Autonomous Objects

1998-02 Floris Wiesman (UM)  
Information Retrieval by Graphically Browsing Meta-Information

1998-03 Ans Steuten (TUD)  
A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective

1998-04 Dennis Breuker (UM)  
Memory versus Search in Games

1998-05 E.W.Oskamp (RUL)  
Computerondersteuning bij Straftoemeting

1999

1999-01 Mark Sloof (VU)  
Physiology of Quality Change Modelling: Automated modelling of Quality Change of Agricultural Products

1999-02 Rob Potharst (EUR)  
Classification using decision trees and neural nets

1999-03 Don Beal (UM)  
The Nature of Minimax Search

1999-04 Jacques Penders (UM)  
The practical Art of Moving Physical Objects

1999-05 Aldo de Moor (KUB) Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems

1999-06 Niek J.E. Wijngaards (VU)  
Re-design of compositional systems

1999-07 David Spelt (UT)  
Verification support for object database design

1999-08 Jacques H.J. Lenting (UM)  
Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation

2000

2000-01 Frank Niessink (VU)  
Perspectives on Improving Software Maintenance

2000-02 Koen Holtman (TUE) Prototyping of CMS Storage Management

2000-03 Carolien M.T. Metselaar (UvA)  
Sociaal-organisatorische gevolgen van kennishechtechnologie; een procesbenadering en actorgespreksgezicht

2000-04 Geert de Haan (VU)  
ETAG, A Formal Model of Competence Knowledge for User Interface Design

2000-05 Ruud van der Pol (UM)  
Knowledge-based Query Formulation in Information Retrieval

2000-06 Rogier van Eijk (UU)  
Programming Languages for Agent Communication

2000-07 Niels Peek (UU)  
Decision-theoretic Planning of Clinical Patient Management

2000-08 Veerle Coupé (EUR)  
Sensitivity Analysis of Decision-Theoretic Networks
<table>
<thead>
<tr>
<th>Year</th>
<th>Code</th>
<th>Author(s)</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>09</td>
<td>Florian Waas (CWI)</td>
<td>Principles of Probabilistic Query Optimization</td>
</tr>
<tr>
<td>2000</td>
<td>11</td>
<td>Jonas Karlsson (CWI)</td>
<td>Scalable Distributed Data Structures for Database Management</td>
</tr>
<tr>
<td>2001</td>
<td>01</td>
<td>Silja Renooij (UU)</td>
<td>Qualitative Approaches to Quantifying Probabilistic Networks</td>
</tr>
<tr>
<td>2001</td>
<td>02</td>
<td>Koen Hindriks (UU)</td>
<td>Agent Programming Languages: Programming with Mental Models</td>
</tr>
<tr>
<td>2001</td>
<td>03</td>
<td>Maarten van Someren (UvA)</td>
<td>Learning as problem solving</td>
</tr>
<tr>
<td>2001</td>
<td>04</td>
<td>Evgueni Smirnov (UM)</td>
<td>Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets</td>
</tr>
<tr>
<td>2001</td>
<td>05</td>
<td>Jacco van Ossenbruggen (VU)</td>
<td>Processing Structured Hypermedia: A Matter of Style</td>
</tr>
<tr>
<td>2001</td>
<td>06</td>
<td>Martijn van Welie (VU)</td>
<td>Task-based User Interface Design</td>
</tr>
<tr>
<td>2001</td>
<td>07</td>
<td>Bastiaan Schonhage (VU)</td>
<td>Diva: Architectural Perspectives on Information Visualization</td>
</tr>
<tr>
<td>2001</td>
<td>08</td>
<td>Pascal van Eck (VU)</td>
<td>A Compositional Semantic Structure for Multi-Agent Systems Dynamics</td>
</tr>
<tr>
<td>2001</td>
<td>09</td>
<td>Pieter Jan 't Hoen (RUL)</td>
<td>Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes</td>
</tr>
<tr>
<td>2001</td>
<td>10</td>
<td>Maarten Sierhuis (UvA)</td>
<td>Modeling and Simulating Work Practice BRAHMS: a multiagent modeling and simulation language for work practice analysis and design</td>
</tr>
<tr>
<td>2001</td>
<td>11</td>
<td>Tom M. van Engers (VU)</td>
<td>Knowledge Management: The Role of Mental Models in Business Systems Design</td>
</tr>
<tr>
<td>2002</td>
<td>01</td>
<td>Nico Lassing (VU)</td>
<td>Architecture-Level Modifiability Analysis</td>
</tr>
<tr>
<td>2002</td>
<td>02</td>
<td>Roelof van Zwol (UT)</td>
<td>Modelling and searching web-based document collections</td>
</tr>
<tr>
<td>2002</td>
<td>03</td>
<td>Henk Ernst Blok (UT)</td>
<td>Database Optimization Aspects for Information Retrieval</td>
</tr>
<tr>
<td>2002</td>
<td>04</td>
<td>Juan Roberto Castelo Valdueza (UU)</td>
<td>The Discrete Acyclic Digraph Markov Model in Data Mining</td>
</tr>
<tr>
<td>2002</td>
<td>05</td>
<td>Radu Serban (VU)</td>
<td>The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents</td>
</tr>
<tr>
<td>2002</td>
<td>06</td>
<td>Laurens Mommers (UL)</td>
<td>Applied legal epistemology: Building a knowledge-based ontology of the legal domain</td>
</tr>
<tr>
<td>2002</td>
<td>07</td>
<td>Peter Boncz (CWI)</td>
<td>Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications</td>
</tr>
<tr>
<td>2002</td>
<td>08</td>
<td>Jaap Gordijn (VU)</td>
<td>Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas</td>
</tr>
<tr>
<td>2002</td>
<td>09</td>
<td>Willem-Jan van den Heuvel (KUB)</td>
<td>Integrating Modern Business Applications with Objectified Legacy Systems</td>
</tr>
<tr>
<td>2002</td>
<td>10</td>
<td>Brian Sheppard (UM)</td>
<td>Towards Perfect Play of Scrabble</td>
</tr>
<tr>
<td>2002</td>
<td>11</td>
<td>Wouter C.A. Wijngaards (VU)</td>
<td>Agent Based Modelling of Dynamics: Biological and Organisational Applications</td>
</tr>
<tr>
<td>2002</td>
<td>12</td>
<td>Albrecht Schmidt (UvA)</td>
<td>Processing XML in Database Systems</td>
</tr>
<tr>
<td>2002</td>
<td>13</td>
<td>Hongjing Wu (TUE)</td>
<td>A Reference Architecture for Adaptive Hypermedia Applications</td>
</tr>
</tbody>
</table>
2002-14 Wieke de Vries (UU)
Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems

2002-15 Rik Eshuis (UT)
Semantics and Verification of UML Activity Diagrams for Workflow Modelling

2002-16 Pieter van Langen (VU)
The Anatomy of Design: Foundations, Models and Applications

2002-17 Stefan Manegold (UvA)
Understanding, Modeling, and Improving Main-Memory Database Performance

2003

2003-01 Heiner Stuckenschmidt (VU)
Ontology-Based Information Sharing in Weakly Structured Environments

2003-02 Jan Broersen (VU)
Modal Action Logics for Reasoning About Reactive Systems

2003-03 Martijn Schuemie (TUD)
Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy

2003-04 Milan Petkovic (UT)
Content-Based Video Retrieval Supported by Database Technology

2003-05 Jos Lehmann (UvA)
Causation in Artificial Intelligence and Law - A modelling approach

2003-06 Boris van Schooten (UT)
Development and specification of virtual environments

2003-07 Machiel Jansen (UvA)
Formal Explorations of Knowledge Intensive Tasks

2003-08 Yongping Ran (UM)
Repair Based Scheduling

2003-09 Rens Kortmann (UM)
The resolution of visually guided behaviour

2003-10 Andreas Lincke (UvT)
Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture

2003-11 Simon Keizer (UT)
Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks

2003-12 Roeland Ordelman (UT)
Dutch speech recognition in multimedia information retrieval

2003-13 Jeroen Donkers (UM)
Nosce Hostem - Searching with Opponent Models

2003-14 Stijn Hoppenbrouwers (KUN)
Freezing Language: Conceptualisation Processes across ICT-Supported Organisations

2003-15 Mathijs de Weerdt (TUD)
Plan Merging in Multi-Agent Systems

2003-16 Menzo Windhouwer (CWI)
Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses

2003-17 David Jansen (UT)
Extensions of Statecharts with Probability, Time, and Stochastic Timing

2003-18 Levente Kocsis (UM)
Learning Search Decisions

2004

2004-01 Virginia Dignum (UU)
A Model for Organizational Interaction: Based on Agents, Founded in Logic

2004-02 Lai Xu (UvT)
Monitoring Multi-party Contracts for E-business

2004-03 Perry Groot (VU)
A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving

2004-04 Chris van Aart (UvA)
Organizational Principles for Multi-Agent Architectures

2004-05 Viara Popova (EUR)
Knowledge discovery and monotonicity

2004-06 Bart-Jan Hommes (TUD)
The Evaluation of Business Process Modeling Techniques
2004-07 Elise Boltjes (UM)  
Voorbeeldig onderwijs; voorbeeldgestuurde onderwijs, een opstap naar abstract denken, vooral voor meisjes

2004-08 Joop Verbeek (UM)  
Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politiegegevensuitwisseling en digitale expertise

2004-09 Martin Caminada (VU)  
For the Sake of the Argument; explorations into argument-based reasoning

2004-10 Suzanne Kabel (UvA)  
Knowledge-rich indexing of learning-objects

2004-11 Michel Klein (VU)  
Change Management for Distributed Ontologies

2004-12 The Duy Bui (UT)  
Creating emotions and facial expressions for embodied agents

2004-13 Wojciech Jamroga (UT)  
Using Multiple Models of Reality: On Agents who Know how to Play

2004-14 Paul Harrenstein (UU)  
Logic in Conflict. Logical Explorations in Strategic Equilibrium

2004-15 Arno Knobbe (UU)  
Multi-Relational Data Mining

2004-16 Federico Divina (VU)  
Hybrid Genetic Relational Search for Inductive Learning

2004-17 Mark Winands (UM)  
Informed Search in Complex Games

2004-18 Vania Bessa Machado (UvA)  
Supporting the Construction of Qualitative Knowledge Models

2004-19 Thijs Westerveld (UT)  
Using generative probabilistic models for multimedia retrieval

2004-20 Madelon Evers (Nyenrode)  
Learning from Design: facilitating multidisciplinary design teams

2005

2005-01 Floor Verdenius (UVA)  
Methodological Aspects of Designing Induction-Based Applications

2005-02 Erik van der Werf (UM)  
AI techniques for the game of Go

2005-03 Franc Grootjen (RUN)  
A Pragmatic Approach to the Conceptualisation of Language

2005-04 Nirvana Meratnia (UT)  
Towards Database Support for Moving Object data

2005-05 Gabriel Infante-Lopez (UVA)  
Two-Level Probabilistic Grammars for Natural Language Parsing

2005-06 Pieter Spronck (UM)  
Adaptive Game AI

2005-07 Flavius Frasincar (TUE)  
Hypermedia Presentation Generation for Semantic Web Information Systems

2005-08 Richard Vdovjak (TUE)  
A Model-driven Approach for Building Distributed Ontology-based Web Applications

2005-09 Jeen Broekstra (VU)  
Storage, Querying and Inferencing for Semantic Web Languages

2005-10 Anders Bouwer (UVA)  
Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments

2005-11 Elth Ogston (VU)  
Agent Based Matchmaking and Clustering - A Decentralized Approach to Search

2005-12 Csaba Boer (EUR)  
Distributed Simulation in Industry

2005-13 Fred Hamburg (UL)  
Een Computersmodel voor het Ondersteunen van Euthanasiebeslissingen

2005-14 Borys Omelayenko (VU)  
Web-Service configuration on the Semantic Web: Exploring how semantics meets pragmatics

2005-15 Tibor Bosse (VU)  
Analysis of the Dynamics of Cognitive Processes
<table>
<thead>
<tr>
<th>Year</th>
<th>Author(s)</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005-16</td>
<td>Joris Graaumans (UU)</td>
<td>Usability of XML Query Languages</td>
</tr>
<tr>
<td>2005-17</td>
<td>Boris Shishkov (TUD)</td>
<td>Software Specification Based on Re-usable Business Components</td>
</tr>
<tr>
<td>2005-18</td>
<td>Danielle Sent (UU)</td>
<td>Test-selection strategies for probabilistic networks</td>
</tr>
<tr>
<td>2005-19</td>
<td>Michel van Dartel (UM)</td>
<td>Situated Representation</td>
</tr>
<tr>
<td>2005-20</td>
<td>Cristina Coteanu (UL)</td>
<td>Cyber Consumer Law, State of the Art and Perspectives</td>
</tr>
<tr>
<td>2005-21</td>
<td>Wijnand Derks (UT)</td>
<td>Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics</td>
</tr>
<tr>
<td>2006-01</td>
<td>Samuil Angelov (TUE)</td>
<td>Foundations of B2B Electronic Contracting</td>
</tr>
<tr>
<td>2006-02</td>
<td>Cristina Chisalita (VU)</td>
<td>Contextual issues in the design and use of information technology in organizations</td>
</tr>
<tr>
<td>2006-03</td>
<td>Noor Christoph (UVA)</td>
<td>The role of metacognitive skills in learning to solve problems</td>
</tr>
<tr>
<td>2006-04</td>
<td>Marta Sabou (VU)</td>
<td>Building Web Service Ontologies</td>
</tr>
<tr>
<td>2006-05</td>
<td>Cees Pierik (UU)</td>
<td>Validation Techniques for Object-Oriented Proof Outlines</td>
</tr>
<tr>
<td>2006-07</td>
<td>Marko Smiljanic (UT)</td>
<td>XML schema matching – balancing efficiency and effectiveness by means of clustering</td>
</tr>
<tr>
<td>2006-08</td>
<td>Eelco Herder (UT)</td>
<td>Forward, Back and Home Again - Analyzing User Behavior on the Web</td>
</tr>
<tr>
<td>2006-09</td>
<td>Mohamed Wahdan (UM)</td>
<td>Automatic Formulation of the Auditor’s Opinion</td>
</tr>
<tr>
<td>2006-10</td>
<td>Ronny Siebes (VU)</td>
<td>Semantic Routing in Peer-to-Peer Systems</td>
</tr>
<tr>
<td>2006-11</td>
<td>Joeri van Ruth (UT)</td>
<td>Flattening Queries over Nested Data Types</td>
</tr>
<tr>
<td>2006-12</td>
<td>Bert Bongers (VU)</td>
<td>Interactivation - Towards an e-ology of people, our technological environment, and the arts</td>
</tr>
<tr>
<td>2006-13</td>
<td>Henk-Jan Lebbink (UU)</td>
<td>Dialogue and Decision Games for Information Exchanging Agents</td>
</tr>
<tr>
<td>2006-14</td>
<td>Johan Hoorn (VU)</td>
<td>Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change</td>
</tr>
<tr>
<td>2006-15</td>
<td>Rainer Malik (UU)</td>
<td>CONAN: Text Mining in the Biomedical Domain</td>
</tr>
<tr>
<td>2006-16</td>
<td>Carsten Riggelsen (UU)</td>
<td>Approximation Methods for Efficient Learning of Bayesian Networks</td>
</tr>
<tr>
<td>2006-17</td>
<td>Stacey Nagata (UU)</td>
<td>User Assistance for Multitasking with Interruptions on a Mobile Device</td>
</tr>
<tr>
<td>2006-18</td>
<td>Valentin Zhizhkun (UVA)</td>
<td>Graph transformation for Natural Language Processing</td>
</tr>
<tr>
<td>2006-20</td>
<td>Marina Velikova (UvT)</td>
<td>Monotone models for prediction in data mining</td>
</tr>
<tr>
<td>2006-21</td>
<td>Bas van Gils (RUN)</td>
<td>Adaptness on the Web</td>
</tr>
<tr>
<td>2006-22</td>
<td>Paul de Vrieze (RUN)</td>
<td>Fundaments of Adaptive Personalisation</td>
</tr>
<tr>
<td>2006-23</td>
<td>Ion Juvina (UU)</td>
<td>Development of Cognitive Model for Navigating on the Web</td>
</tr>
<tr>
<td>2006-24</td>
<td>Laura Hollink (VU)</td>
<td>Semantic Annotation for Retrieval of Visual Resources</td>
</tr>
<tr>
<td>Year</td>
<td>Code</td>
<td>Author(s)</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>-----------</td>
</tr>
<tr>
<td>2006-25</td>
<td>Madalina Drugan (UU)</td>
<td>Conditional log-likelihood MDL and Evolutionary MCMC</td>
</tr>
<tr>
<td>2006-26</td>
<td>Vojkan Mihajlovic (UT)</td>
<td>Score Region Algebra: A Flexible Framework for Structured Information Retrieval</td>
</tr>
<tr>
<td>2006-27</td>
<td>Stefano Bocconi (CWI)</td>
<td>Vox Populi: generating video documentaries from semantically annotated media repositories</td>
</tr>
<tr>
<td>2006-28</td>
<td>Borkur Sigurbjornsson (UVA)</td>
<td>Focused Information Access using XML Element Retrieval</td>
</tr>
<tr>
<td>2007-01</td>
<td>Kees Leune (UvT)</td>
<td>Access Control and Service-Oriented Architectures</td>
</tr>
<tr>
<td>2007-02</td>
<td>Wouter Teepe (RUG)</td>
<td>Reconciling Information Exchange and Confidentiality: A Formal Approach</td>
</tr>
<tr>
<td>2007-03</td>
<td>Peter Mika (VU)</td>
<td>Social Networks and the Semantic Web</td>
</tr>
<tr>
<td>2007-06</td>
<td>Gilad Mishne (UVA)</td>
<td>Applied Text Analytics for Blogs</td>
</tr>
<tr>
<td>2007-07</td>
<td>Natasa Jovanovic’ (UT)</td>
<td>To Whom It May Concern - Addressee Identification in Face-to-Face Meetings</td>
</tr>
<tr>
<td>2007-08</td>
<td>Mark Hoogendoorn (VU)</td>
<td>Modeling of Change in Multi-Agent Organizations</td>
</tr>
<tr>
<td>2007-09</td>
<td>David Mobach (VU)</td>
<td>Agent-Based Mediated Service Negotiation</td>
</tr>
<tr>
<td>2007-11</td>
<td>Natalia Stash (TUE)</td>
<td>Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System</td>
</tr>
<tr>
<td>2007-12</td>
<td>Marcel van Gerven (RUN)</td>
<td>Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty</td>
</tr>
<tr>
<td>2007-13</td>
<td>Rutger Rienks (UT)</td>
<td>Meetings in Smart Environments: Implications of Progressing Technology</td>
</tr>
<tr>
<td>2007-14</td>
<td>Nick Bergboer (UM)</td>
<td>Context-Based Image Analysis</td>
</tr>
<tr>
<td>2007-15</td>
<td>Joyca Lacroix (UM)</td>
<td>NIM: a Situated Computational Memory Model</td>
</tr>
<tr>
<td>2007-16</td>
<td>Davide Grossi (UU)</td>
<td>Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems</td>
</tr>
<tr>
<td>2007-17</td>
<td>Theodore Charitos (UU)</td>
<td>Reasoning with Dynamic Networks in Practice</td>
</tr>
<tr>
<td>2007-18</td>
<td>Bart Orriens (UvT)</td>
<td>On the development an management of adaptive business collaborations</td>
</tr>
<tr>
<td>2007-19</td>
<td>David Levy (UM)</td>
<td>Intimate relationships with artificial partners</td>
</tr>
<tr>
<td>2007-20</td>
<td>Slinger Jansen (UU)</td>
<td>Customer Configuration Updating in a Software Supply Network</td>
</tr>
<tr>
<td>2007-21</td>
<td>Karianne Vermaas (UU)</td>
<td>Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005</td>
</tr>
<tr>
<td>2007-22</td>
<td>Zlatko Zlatev (UT)</td>
<td>Goal-oriented design of value and process models from patterns</td>
</tr>
<tr>
<td>2007-23</td>
<td>Peter Barna (TUE)</td>
<td>Specification of Application Logic in Web Information Systems</td>
</tr>
<tr>
<td>2007-24</td>
<td>Georgina Ramrez Camps (CWI)</td>
<td>Structural Features in XML Retrieval</td>
</tr>
</tbody>
</table>