Theory in Practice: A Case Study of Requirements Engineering Process Improvement

by

James Chisan
B.Sc., The University of Calgary, 2001

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of

MASTER OF SCIENCE

in

the Department of Computer Science

THE UNIVERSITY OF VICTORIA
April 29, 2005
© James Chisan, 2005
All rights reserved.
This dissertation may not be reproduced in whole or in part, by photocopying or other means, without the permission of the author.
Abstract

The notion that requirements is one of the most important steps in developing software, is perpetuated by claims that requirements engineering can improve a software project in many ways: by reducing project risk, assuring quality and improving productivity. Yet despite its apparent importance and wide-ranging benefits, poor requirements engineering is still one of the leading factors contributing to project failure.

However, many of the potential benefits of requirements engineering are largely unproven. A close examination of the literature reveals that there is little systematic, detailed evidence which supports the claims of the benefits of requirements engineering. Such evidence could serve to motivate industrial adoption of requirements engineering techniques, validate claims and contribute to our understanding of how the benefits of requirements engineering are realized in practice.

This thesis presents an analysis of the causal relationship between requirements engineering practice and the beneficial effects in risk management, quality and productivity. Conducting a 30-month long case study of one software organization that had implemented requirements process improvement, this work seeks to examine how requirements engineering can affect software development. The case study followed the entire soft-
ware development project from inception to after deployment and was guided by specific claims in literature: payoffs of requirements engineering practice include increases in productivity, quality and risk management. It sought to unveil the details of how the requirements process would affect the organization's ability to make resource estimations, negotiate with its customers, improve software quality, maintain customer satisfaction and assure effective product testing. It also examines how requirements engineering process interacts and is interdependent on other development processes such as planning, tracking and testing.

The research brings forward valuable evidence showing how the organization was able to use requirements engineering to improve risk management, product quality and developer productivity throughout the project's life cycle. In particular, requirements practices were beneficial to estimations, improved communication and increased developer understanding. The research also contributes to theory by proposing a map of interaction, showing how requirements and other development processes interact and are interdependent. These findings and the experience of this research raises important questions in which to drive future research in new directions by considering the role of communication, the importance of the requirements specification and lack of established research instruments in conducting this type of research.
Contents

Abstract ................................................................. ii

Contents ................................................................. iv

List of Tables ........................................................... viii

List of Figures ........................................................... ix

List of Charts ............................................................ x

Acknowledgements ....................................................... xi

1 Introduction ............................................................ 1

2 Background .............................................................. 11
   2.1 Software Engineering ............................................ 12
   2.2 Software Process Improvement .................................. 15
   2.3 Requirements Engineering ...................................... 19
      2.3.1 Benefits of Requirements Engineering .................. 20
      2.3.2 Why is Requirements Engineering Important? ........ 23
      2.3.3 The Need for Evidence .................................... 24
      2.3.4 Related Empirical Work ................................. 26
Contents

2.4 Summary .......................................................... 30

3 Research Design .................................................. 31
3.1 The Research Question ........................................ 31
3.2 Case Study Propositions ....................................... 35
   3.2.1 Problem Understanding .................................. 36
   3.2.2 Accurate Estimation Gathering ......................... 36
   3.2.3 Schedule Commitments .................................. 37
3.3 Unit of Analysis ................................................ 37
   3.3.1 The Company ............................................. 38
   3.3.2 The Product ............................................. 39
   3.3.3 The Project ............................................. 39
3.4 ACUS' Requirements Engineering Process .................. 40
   3.4.1 The Former Process ..................................... 40
   3.4.2 Process Revisions ..................................... 42
3.5 Data Collection ................................................ 46
   3.5.1 Ethics .................................................. 49
3.6 Aspects Investigated ......................................... 50
   3.6.1 Initial Phase ........................................... 50
   3.6.2 Intermediate Phase .................................... 51
   3.6.3 Final Phase ........................................... 55
3.7 Summary ....................................................... 62

4 Case Study Findings ........................................... 63
4.1 General Perceptions of the Requirements Process After Development .................................... 64
<table>
<thead>
<tr>
<th>Contents</th>
<th>vi</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2 Problem Understanding</td>
<td>65</td>
</tr>
<tr>
<td>4.2.1 Details, Dependencies and Complexities of Features</td>
<td>65</td>
</tr>
<tr>
<td>4.2.2 Wasted Effort</td>
<td>65</td>
</tr>
<tr>
<td>4.2.3 Improved Communication</td>
<td>66</td>
</tr>
<tr>
<td>4.2.4 Informed Decisions</td>
<td>68</td>
</tr>
<tr>
<td>4.3 Estimations</td>
<td>69</td>
</tr>
<tr>
<td>4.4 System-Test Defects</td>
<td>71</td>
</tr>
<tr>
<td>4.5 Requirements Creep</td>
<td>73</td>
</tr>
<tr>
<td>4.6 Post-deployment Indications</td>
<td>75</td>
</tr>
<tr>
<td>4.7 The Role of REP Components in Contributing to Observed Effects</td>
<td>76</td>
</tr>
<tr>
<td>4.8 Requirements Engineering Process Effects on Major Process Areas</td>
<td>77</td>
</tr>
<tr>
<td>4.9 REP Component Impact</td>
<td>79</td>
</tr>
<tr>
<td>4.9.1 Five highly affected processes</td>
<td>80</td>
</tr>
<tr>
<td>4.9.2 Component Responses Across Process Areas</td>
<td>82</td>
</tr>
<tr>
<td>4.10 Relative Impact of REP Components</td>
<td>84</td>
</tr>
<tr>
<td>4.11 Summary</td>
<td>86</td>
</tr>
<tr>
<td>5 Discussion</td>
<td>87</td>
</tr>
<tr>
<td>5.1 Relationships Among Development Processes</td>
<td>88</td>
</tr>
<tr>
<td>5.1.1 Risk Management</td>
<td>93</td>
</tr>
<tr>
<td>5.1.2 Quality</td>
<td>101</td>
</tr>
<tr>
<td>5.1.3 Productivity</td>
<td>104</td>
</tr>
<tr>
<td>5.2 Early Engineer Response</td>
<td>109</td>
</tr>
<tr>
<td>5.3 Applicability</td>
<td>110</td>
</tr>
</tbody>
</table>
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4 Limitations</td>
<td>113</td>
</tr>
<tr>
<td>5.4.1 Alternative Theories</td>
<td>113</td>
</tr>
<tr>
<td>5.4.2 Shortcomings in Methodology</td>
<td>117</td>
</tr>
<tr>
<td>5.5 Summary</td>
<td>118</td>
</tr>
<tr>
<td>6 Conclusions</td>
<td>120</td>
</tr>
<tr>
<td>6.1 Insights</td>
<td>120</td>
</tr>
<tr>
<td>6.1.1 Communication and Collaboration</td>
<td>121</td>
</tr>
<tr>
<td>6.1.2 Room for Agility</td>
<td>122</td>
</tr>
<tr>
<td>6.2 Implications for Practitioners</td>
<td>124</td>
</tr>
<tr>
<td>6.3 Reviewing the Research</td>
<td>127</td>
</tr>
<tr>
<td>6.4 Recommendations for Practitioners</td>
<td>129</td>
</tr>
<tr>
<td>6.5 Directions for Future Research</td>
<td>132</td>
</tr>
<tr>
<td>6.6 Contributions</td>
<td>133</td>
</tr>
<tr>
<td>6.7 Closing Remarks</td>
<td>134</td>
</tr>
</tbody>
</table>

Bibliography | 136 |

A Questionnaire 2 | 143 |

B Questionnaire 3 | 148 |
List of Tables

4.1 Artefacts affected by change requests ........................................ 74
4.2 Change requests involvement according to role .......................... 74
4.3 Requirements process impact responses by REP component for highest scoring sub-processes ........................................ 81
5.1 Summary of observed requirements process payoffs during development ........................................ 91
List of Figures

3.1 Timeline showing development and research phases  . . . .  47

5.1 Bold arrows indicate the requirements process positive ef-
fect on other processes which contributed to project effects
summarized in Table 5.1.  . . . . . . . . . . . . . . . . . . . . . .  90

5.2 Detailed requirements engineering process interactions lead-
ing to accurate estimations, effective project negotiation and
reduced requirements creep  . . . . . . . . . . . . . . . . . . . . .  96

5.3 Detailed requirements process interactions leading to fewer
defects and better feature coverage  . . . . . . . . . . . . . . . 101

5.4 Detailed requirements process interactions leading to reduced
rework and improved communication  . . . . . . . . . . . . . . . 107
List of Charts

4.1 Importance of understanding requirements during design, coding, testing and documentation ................. 66
4.2 Agreement that requirement artefacts enabled post-requirement activities .............................................. 67
4.3 Agreement that requirement had intangible benefits ....... 68
4.4 Importance of requirements in estimation for design, implementation, testing and documentation .................. 69
4.5 Estimation error made before and after requirements analysis 70
4.6 Absolute (and cumulative) estimation error before and after requirements analysis ................................. 71
4.7 Proportion of pre-deployment defect types, according to respondents .................................................... 72
4.8 Relative importance of individual REP components ...... 77
4.9 Requirements Process Impact on Process Areas .......... 78
4.10 Requirements process impact on processes organized by process area (average of responses) .................... 79
4.11 Cumulative REP component scores by process area ...... 83
4.12 REP impact on processes organized by process area (average of responses) ....................................... 84
Acknowledgements

I would like to thank Dr. Daniela Damian, my supervisor and my friend, without whom this research would not have been possible. Thank you for the foresight and trust in allowing me to continue the work you started. Thank you also for your council, patience, endurance and support.

Thank you to my committee for their insightful comments and gratifying interest.

Thank you to Elizabeth Wolfe, my lab colleague and friend, who provided me with encouragement and insight.

Funding for my work was provided by an NSERC scholarship, and awards provided by the University of Victoria, and the Department of Computer Science. I want to thank these funding institutions for recognizing the importance of academic work... and for paying my rent.

Thank you to my parents who has been very supportive of my work and my studies, particularly when I needed it the most.

Finally thank you to Carolyn Campbell for her indispensable editorial assistance and, more importantly, her relentless encouragement and unwavering support.
Chapter 1

Introduction

The most important aspect of every development project, whether it is to build a space-vehicle, to build a software application or to build tool shed, is to determine what purpose the project aims to achieve. After all, designing a telecommunications satellite is far different than designing a manned mission to mars, even if the final outcome is a space-vehicle.

The essence of requirements engineering as a discipline is about determining what the project should endeavour to achieve. However, even the notion of requirements engineering is complex: How should project goals be determined? How does one sufficiently describe a thing that they envision? How should that description be disseminated among those involved in the project? These questions all depend on a more fundamental issue: what is the purpose of the description? In other words, what is the purpose of requirements engineering and what is its role with respect to the rest of the project? For example, to what extent is requirements engineering responsible for providing the following functions:

- gauge feasibility at inception
- promote project success
- solidify project goals
Chapter 1. Introduction

- provide common understanding to those involved in the project
- promote communication among stakeholders
- judge the intermediate or final outcomes of the project
- structure, prioritize and resolve conflicting goals
- aid subsequent project planning and tracking
- determine and deliver minimum standards for quality
- construct preliminary designs
- maintain the currency of project goals over the course of the project

Just as the first step in developing software should be to determine what it should do, so too should the first step in developing an effective requirements engineering process is to determine what that process should do. Encouraging the adoption of requirements practices in industry and enabling effective academic research first requires an understanding of how requirements activities can affect software development projects. This thesis contributes toward answering that question.

Although it is possible to apply the lessons of requirements engineering to projects in many domains, it is typically understood to concern software development projects. For the purposes of this thesis, requirements engineering will be considered completely within the domain of software engineering. By doing so, it is possible to leverage well defined terminology and concepts.
Chapter 1. Introduction

Software Engineering is the disciplined and systematic approach to software development, it concerns both the project's logistic and technical aspects: from project inception through to software implementation, deployment and subsequent maintenance. Requirements engineering is a topic of software engineering which is concerned with the early stages of software development. Although it is recognized that requirements engineering is an important part of a software development project, oft-referenced authors suggest that requirements engineering is also the most difficult aspect of a software development project (Brooks, 1987). However, even among those who agree with this opinion, there is a lack of understanding as to how successful requirements engineering imparts success on software development and why.

Industrial practitioners who are optimizing their software production are confronted with a myriad of attractive claims about the benefits of requirements engineering. There are claims advocating the virtues of adopting particular methods, techniques and tools that are all meant to establish or improve an organization's requirements engineering practices. Unfortunately, authors tend to base their claims on scientifically weak or indirect evidence. Many, for example, focus on comparing a particular technique in a particular situation. Such comparisons, while useful, fail to capture the full context of their success or failure, and thus do little to advance broader theory. In some cases claims are supported by anecdotal evidence from the author or experts, but clearly suffer from a lack of systematic scientific rigour. So, while there are many claims that requirements engineering leads to more successful software projects, there is little evidence
Academics in the requirements field have recognized this problem. They have been frustrated by what they perceive as low rates of adoption of requirements engineering techniques and methodologies. They believe that for more widespread adoption to occur, the effects of requirement engineering on software development must be thoroughly analyzed and understood. Moreover, such work would enable more informed and broader comparisons of emerging techniques and thus promote more effective research.

In response to these concerns, this thesis scientifically and empirically contributes to our understanding of requirements engineering. It endeavours to unveil the relationship between requirements engineering and software development. To achieve this, established research methods are brought to bear in order to systematically collect much needed empirical evidence. Finally, through analysis of the evidence and the context of its collection enables the construction of a theoretical map of interaction for how requirements activities may affect software development.

Research Question

This research is guided by the overall question asking: how does requirements engineering affect software development throughout a project's development life cycle? The answer should shed light on how requirements engineering leads to more successful projects and how in particular it affects software engineering practices. In particular, this investigation considers the long-term role requirements engineering plays in project planning,
software implementation and the efficacy of other development processes.

By using claims made in the relevant literature about the benefits of requirements engineering the study begins by seeking to detect desirable payoffs related to risk management, quality and productivity. These findings alone provide valuable corroborative evidence; and add much-needed empirical support to these claims. To investigate the issue more deeply this research also examined the nature and extent of process interaction by considering how the requirements process had affected other planning and development processes (such as resource estimations or system testing, for example).

Approach

Carrying out research of this kind requires the systematic analysis of a software development project over its full development life-cycle. This research follows the development of a software project at an Australian software development organization. This organization had revised their requirements processes just prior to the commencement of the subject project. This provided a unique opportunity to study how adopting a more formal requirements engineering process would affect software production in an industrial environment over the life-cycle of the project.

Given the nature of the research question and the subject of the study the research was structured as a case study. This detail oriented research method is uniquely appropriate for unveiling the nuances of phenomena over which the researcher has no control. Current theory from the field was used to inform the particular design of the case study, focusing on
investigating aspects of risk management, quality and productivity.

A variety of evidence was collected over the course of the investigation. Conducting interviews, administering questionnaires and inspecting technical documents and project artefacts produced a multitude of qualitative and quantitative evidence. As the software development project proceeded, a series of questionnaires and artefact inspections carried out at various points throughout the development project provided the majority of evidence. Using multiple data sources enables the use of triangulation, which can increase confidence in the interpretation of evidence.

Results

The evidence collected by this study shows how requirements engineering positively affected the software development throughout its life cycle. For example, the organization was able to conduct more effective negotiations with its stakeholders and derive more accurate estimations after requirements analysis, compared to estimates made before analysis. Late during development, communication within the organization had improved and, compared to previous projects, feature creep had been stifled. Testing was by all accounts more effective and far fewer system-test defects were recorded. Post-deployment defects were equally low and customer satisfaction high.

In addition to measuring attributes of the project, this research also explains how the requirements engineering process directly contributed to the efficacy of other processes in the organization. For example, it shows how the analysis and [more accurate] resource estimation of requirements
empowered negotiators, which resulted in more concrete project commitments. It also explains how the conception of test scenarios during requirements analysis benefited system testing enabling more effective test for feature coverage and fewer defects. These relationships are based on empirical evidence collected throughout the study, contributing to a detailed understanding of how requirements engineering affected many aspects of the software development project.

Contributions

This research advances the understanding of requirements engineering among both academic researchers and industrial practitioners. It provides important evidence that not only implicates requirements engineering in contributing to software success, but also shows how these improvements can be realized through improved project planning, tighter project control and enhanced communication.

Furthermore, this work contributes by proposing a theoretical map of how requirements engineering both affects and depends on other development activities. Evidence collected throughout the study provides an empirical basis for understanding how requirements and development process interact with each other and the extent to which they are mutually dependant, beneficial or detrimental.

This understanding is of significant importance to practitioners who are considering how requirements engineering can supplement their existing software development practices. More importantly, the investigation raises critical questions of the nature and value of requirements engineer-
Chapter 1. Introduction

ing itself, particularly by illustrating that requirements engineering can play a powerful role in the efficacy of other development activities.

Provocative underlying trends in the collected evidence inspire new directions to pursue for future research. For example, the importance of communication throughout the project, both among developers and between stakeholders, and the role of requirements engineering in this endeavour is considered. Likewise, the evidence casts doubt on the need to maintain or even develop a rigorous requirements document.

In relating the research experience, fundamental insufficiencies of existing research methods and instruments are exposed. Identifying the effects of requirements practices is problematic due to the period of time between development of requirements and the end of the project. This passage of time permits potential long-run impact far beyond the immediate product of requirements activities. The lack of established, well-known peer-reviewed empirical instruments undeniably impairs this research and the potential for it to be compared to similar future research. In lieu of these shortcomings this thesis carefully documents, in detail, the methods and instruments used to gather evidence.

Thesis Organization

Chapter two includes extensive background material on software engineering, requirements engineering and software process. It also provides academic justification for the methods and approaches used to carry out the study and compares this work to other recent empirical work.

Chapter three describes the design of the study, starting with the re-
search questions that were used to guide the investigation. The study's propositions, inspired by prevalent academic claims and which guide the collection of data, are described. The characteristics of the case are explored, including the software development organization, its development processes and the subject project. Chapter three concludes with an overview of the particular data collected over the course of the study and the means by which it was analyzed.

In chapter four, salient findings from the investigation are presented. The first half of the chapter reveals apparent payoffs of adopting requirements engineering process evident throughout the project's life cycle. In particular improved resource estimations, less wasted effort, improved communication and enhanced problem understanding among developers. The extent of the organization's change management practices and the apparent lack of requirements creep are noted. Project statistics are explained, showing marked improvements in pre and post-deployment defect rates, customer support rates and indications of good customer satisfaction. In the second half of the chapter focuses on results concerning the requirements process itself and its effects on other processes. The relative effectiveness of particular elements of the requirements process are shown. Finally, an analysis of the requirements process's effect on other planning and development processes is presented. This analysis shows how particular elements of the requirements processing such as the conception of test scenarios during requirements analysis improved the efficacy of other processes such as system testing.

Chapter five constitutes a discussion of the findings presented in chap-
Combining evidence of payoffs and process interactions produces a holistic picture of how requirements engineering affected the software development project. Doing so illustrates how requirements practices increased the efficacy of other processes thereby contributing both directly and indirectly to the resulting observed payoffs. A map of process interaction illustrates some of the potential interactions and dependencies between processes during software development. The chapter concludes with a discussion of the limitations of the study which stem from competing alternative interpretations of the evidence, and from fundamental short-comings in the implementation of the research.

Chapter six begins by raising questions about the role of requirements engineering and ultimately proposes possible new directions for future research to pursue. For example, the role of requirements engineering in fostering communication and collaboration is considered. Based on the outcomes of the research, the chapter also provides suggestions about which simple requirements strategies industrial practitioners may use to improve their practice. In conclusion of the thesis, the research is reviewed with respect to the original research questions, and the fundamental contributions of this work is reiterated, including the contribution of much-needed systematic, empirical evidence to the field and the map of process interaction in contributing to requirements theory.
Chapter 2

Background

This chapter provides the background necessary to appreciate and understand the problem that this thesis addresses. To do that, a brief review is provided of the basic tenants of software engineering and the unique characteristics of software development that make software projects particularly challenging. Many of these challenges stem from the use of requirements that guide the project and shape the final product. In Section 2.2 the subject of process improvement is introduced, common approaches to process improvement are reviewed as they pertain directly to the systematic improvement software development. These approaches often start by addressing requirements engineering. The field of requirements engineering refers broadly to the capture, analysis and management of software requirements; this field and how it relates to software engineering is outlined in Section 2.3. Common claims from literature about the benefits of requirements engineering to risk management, quality and productivity are introduced and discussed. In Section 2.3.2, motivation for studying requirements engineering in particular is provided. Section 2.3.3 shows emphasizes the need for empirical evidence to support claims made in literature. Finally Section 2.3.4, reviews commonly used empirical research methods and reviews a sample of empirical work which serves to establish
how this research presented in this thesis is unique.

2.1 Software Engineering

Software increasingly permeates the fabric of everyday life for millions of people around the world. Computers, and the software they run, have proven to be an invaluable tool improving countless domains from entertainment to medicine to transportation and everything in between. As humans become more and more dependant on software so too does our demand for software grow. There is little reason to doubt the continued growth of software development activities worldwide.

Software development is unique among most industrial endeavours due to its intrinsic malleability, its potential for reuse and its fundamentally creative nature. Unlike traditional manufacturing of physical material where engineering results in a design for a product that may be mass produced, software’s engineering effort results in the final product – the software itself. This unique characteristic of software, wherein the specification is the product, naturally leads to the notion that software is a highly adaptable and malleable form that can be easily reshaped and altered. This notion leads many software consumers and users to have high expectations of software providers. The promise of reuse in software exacerbates this belief.

It is argued that software development is inherently a highly creative endeavour. Some claim that whenever a developer encounters an ‘old’ software problem, he need merely refer to its existing solution. This is
more difficult to achieve in practice than in theory. Nevertheless, this argument is true to the extent that it justifies software development as primarily a creative exercise. Even when reuse falls short in practice, creative energies are used to conceive new components even if similar ones already exist.

The intellectual and creative challenges that are the hallmarks of software development multiply the usual complexities faced during any project such as planning, shared vision among stakeholders and the incorporation of new or emerging technologies. Shortly after the inception of the term 'software engineering' Royce introduced his now (in)famous waterfall model (1970) of software development which describes a process-based approach to govern software development projects. This model, despite its scorn among academics for being dangerously simplistic, still enjoys widespread use in software projects today (Larman & Basili, 2003). Royce’s model features distinct phases of software development, most significantly: requirements analysis, design, implementation and testing.

Examination of most software engineering textbooks reveals that each software project phase is critical to producing software that is free of defects, fulfills the expectations of its users and does so in an economically feasible way (Sommerville, 2000; Pfleeger, 2001; Pressman, 2004). During requirements analysis, the context in which the system is meant to be deployed is analyzed and the purpose of the software is determined. In the words of Brooks, it is in this phase when one establishes “what needs to be built” (Brooks, 1987). During design, designers use requirements while considering the technological aspects of the system to produce a design
that guides programmers or software implementors during the implementation phase of development. It is during implementation that developers codify the software system, usually in the form of source code. It is this codified artefact that ultimately results in the final software product. The testing phase of development denotes the practice of testing the software product, either in whole or in part, to assure the software is correct while satisfying the requirements that had been established at the beginning of the project. While Royce’s waterfall model describes phases of development, this delineation rarely bares out in practice. Typically, development phases overlap or are iteratively repeated as described by the spiral model (Boehm, 1988). Nevertheless, the waterfall model’s basic phases serve to illustrate the elements common to most software development projects.

Although in many projects a majority of time and effort is spent during implementation and test phases of development, requirements engineering activities have long been recognized as one of, if not the most difficult aspect of software development (Brooks, 1987). It has been shown that errors which occur early, during this stage of development are costly to fix if they are not detected and corrected (Boehm, 1981). Therefore, the field of requirements engineering is primarily concerned with the elicitation, analysis and management of requirements during software development projects.
2.2 Software Process Improvement

Software process improvement (SPI) refers to the improvement of software development practices through prescribed manipulation of the software processes. It is founded on the premise that shortcomings in software development performance are partially to blame on the project's process. This is naturally a very compelling argument given that the whole purpose of the process is to describe the steps necessary to produce the desired outcome. If the outcome is wrong, the steps to achieve that outcome must have been wrong too. The popularity of formal software process improvement models appears to reflect the hopes that these models can improve development practices.

When revising software process for the purposes of improving the software, there are two main schools of thought: heavyweight and agile processes. Heavyweight methodologies are typified by the Capability Maturity Model (CMM) (Software Engineering Institute, 1995), its successor CMMI (Software Engineering Institute, 2001) and process evaluation frameworks like it, such as the ISO 15504. Although they do not necessarily dictate heavyweight processes, they do assume a document-oriented development process. The virtue of these heavyweight methodologies is thought to be systematic structure, control and repeatability.

The Agile movement, on the other hand, promotes the adoption of lightweight methodologies that rely heavily on social interactions among programmers, and among programmers and customers. In contrast to heavy processes, Agile processes rely on maintaining information in the developer's heads rather than codifying and maintaining that informa-
tion in physical artefacts. Agile process methodologies praise flexibility, laud organic self-organization and view code as 'the primary measure of progress'. As a result requirements engineering is considered an unnecessary effort that needlessly separates the user from the eventual delivery of the product. Instead, to understand the needs of the user, Agile processes emphasize direct customer contact and frequent product iterations.

Both the CMM and ISO 15504 deserve special interest here because they appear to suggest a means by which an organization can achieve incremental process improvement. Although both standards are careful to disclaim their use as a methodology they do have standards that can be used to inform software process improvement. The models both define various levels of 'maturity' or 'capability', and for each level describe a series of dependant process characteristics. Both claim that organizations who exhibit more and more of these characteristics are more likely to deliver software that is both correct and on-time – with the obvious implications to quality and productivity.

These assessment models most likely owe their popularity to the benefits to productivity, quality and risk management that they each promise. (As well as the necessity for organizations to be formally assessed as a stringent condition for bidding on many military and government funded projects.) For example, Paulk, Curtis, Chrissis & Weber (1993), describing the CMM, cite hypothesized benefits in: (1) productivity: cost decrease, shorter development time, and increased quality; and (2) project performance management: more accurate, less variable project performance forecasts. Not surprisingly, the CMM addresses requirements en-
engineering and requirements management as an initial indication of maturity, and on which subsequent improvements can be founded. The first level of improvement, 'level 2', specified by the CMM, defines one of its five key process areas as 'requirements management'. This key process area establishes a baseline for software engineering and management used to guide software plans, products and activities that are consistent with system requirements. ISO 15504 describes the 'Develop software requirements' process. This process, given successful implementation, promises that requirements will be congruent with customers' stated and implied needs as well as correct and testable.

As one might expect, these models are not without their critics. Although ISO 15504 suggests that assessment results indicate an organization's ability to achieve productivity, it is ambiguous whether this can be achieved via individual processes alone, or whether a combination of processes is required (Emam, Melo & Drouin, 1997). Furthermore, ISO standards certification is said to rely heavily on the expertise and training of assessors rather than its prescribed standards (Paulk, 1994). Even so, assessment models that address the entire software life cycle often treat each development process equally, arguably treating fundamentally important topics like requirements engineering insufficiently.

Experts within the requirements field have suggested that process improvement and maturity models do not adequately attend to requirements issues. Sawyer, Sommerville & Viller (1997) criticize the CMM and ISO models for being vague and for "offering little direct help to an organization committed to serious improvements in their requirements pro-
This sentiment appears to disagree with El Emam and Birk's findings (2000) that among large organizations, good requirements practice correlates highly with ISO 15504 maturity, although the study does not establish causation. Instead of broadly-applicable models, Sawyer et al. have proposed their own requirements process maturity model that details a series of implementation-ready industrial practices to improve the requirements process. They argue that higher maturity in this model yields improved consistency in project risk, enhanced software quality and a 'capability to solve unforeseen requirements problems'. However, even Sawyer and Sommerville admit in their book of good practices (1997) that their practices are heavily dependant on an organization, its development process, its tools and particular circumstances.

Unfortunately all of these frameworks are largely meta-processes. They provide high-level, abstract guidelines that emphasize implementing a software process without actually specifying any details of that process.

If every organization needs a different process, or indeed "every project needs a different process" as at one requirements guide suggests (Robertson & Robertson, 1999), it seems acceptable that these models would be bereft of details. Perhaps academics and experts avoid concrete, detailed advice because of the tension between process rigour and the flexibility required to develop software. As Armour (2003) suggests: software development is a unique endeavour emphasizing creative solutions to new problems, and that the necessity to remain dynamically adaptable is at odds with rigid process prescription. Armour believes that "if a process can tell us exactly what to do, then it should do it for us". This sentiment
implies that a well defined process cannot anticipate every occasion and that there will be some disparity between process and practice.

2.3 Requirements Engineering

One of the first tasks of software development is to decide the nature of the outcome: in other words, to determine what software to build. How to determine what to build, what it will do and who will use it, is addressed by requirements engineering (Sommerville & Kontonya, 1998; Robertson & Robertson, 2004). Requirements engineering refers to the process by which the necessary requirements of a system and its constraints are established, analyzed, documented and maintained. Although this sounds easy enough, the inherent complexity of the system, the potential variation or even conflict among users, customers and project stakeholders, and the uncertainty of business demands all make for an extremely challenging problem. It is no wonder why requirements engineering is sometimes considered the most difficult part of software development.

Requirements engineering, however, is not limited merely to determining appropriate requirements to guide the development of software systems. It is far more wide ranging and long-term than that. In addition to coherently articulating the needs and goals of the project stakeholders in the early stages of the project, it must also concern assure that requirements have been realized in the final product. Making such assurances usually necessitates considering how to determine whether the requirement has been satisfied by the final product. Furthermore, requirements
management endeavours to maintain the integrity of established requirements while the development proceeds. Customer's needs adjust, business markets change and the vision of the project evolves. Ideally these changes must be captured and managed to ensure that the product reflects the environment it is to be deployed within. So while major requirements engineering activities occur at the beginning of a project, requirements engineering also plays an important role throughout a project's development life cycle.

2.3.1 Benefits of Requirements Engineering

A review of the current literature reveals that requirements engineering offers many potential benefits to a software development project. The works referred to below demonstrate the implied benefits of requirements engineering on productivity, quality and risk management. They also show how authors have chosen to operationalize these concepts into measurable objective project attributes. For example, these works reveal how productivity can be measured by decreases in developer effort, how quality can be measured by customer satisfaction, or risk management by the accuracy of resource estimations. Many of these benefits are attributed to successful requirements process improvement, or used as dependent variables with which to measure the impact of new methods or techniques.

Productivity

Productivity can take many forms, but essentially falls into two camps: increased development effectiveness as might be envisioned from the use of
a new tool or process, or increases in efficiency, such as preventing rework
(i.e. unwanted features) or lowering the cost of development. Lauesen &
Vinter (2001) considered requirements engineering performance strictly in
terms of hours saved. They report that there are particular requirements
techniques, such as user scenarios, that are clearly superior to others ac-
cording to their metrics. For others a successful requirements engineering
experience is one where software delivery is on-time (Wohlwend & Rosen-
baum, 1993).

Quality

Further, there are many claims about improved quality realized through
requirements process improvement. Herbsleb & Goldenson (1996), found
that mature organizations, according to CMM, exhibited significant im-
provements in self-assessed product quality and customer satisfaction. Af-
ter software improvement initiatives at Schlumberger, engineering teams
that had formally been plagued with delivering incomplete functionality
began to ship software that was ‘correct’ (Wohlwend & Rosenbaum, 1993).

Risk Management

While productivity and quality are critical factors in the development of
software, Brodman and Johnson (Brodman & Johnson, 1995) found that, in
fact, many companies look to implement software process improvement
primarily as a means of reducing their exposure to risk. The companies
they surveyed expressed a keen interest in the accurate assessment of costs
and scheduling while decreasing variability in project success and/or per-
formance. Although costs and scheduling can be considered productivity concerns, their forecasting in the initial stages of a project is clearly a matter of risk management (Humphrey, Snyder & Willis, 1991) and tightly related to activities of requirements management.

Similarly, eight of the ten risks identified by Boehm (1991) refer to establishing realistic schedules, specifying accurate requirements, or controlling requirements change. Establishing accurate project estimates is often identified as the responsibility of requirements engineering (Finkelstein, 1994). These and many other authors clearly imply that good requirements engineering can benefit project estimations, aid specification of requirements through stakeholder agreement, and help control emergent changes.

Other Benefits

There are other beneficial collateral effects that have been documented from successful software process improvement initiatives. For example, Wohlwend & Rosenbaum (1993) note that after successful improvement, developer morale improved markedly. Others (Brodman & Johnson, 1995), suggest that companies have also found that less overtime leads to improved confidence, less turnover and increased intra-organizational cooperation. Beecham, Hall & Rainer (2003) specifically link improved requirements process to better staff retention rates, while Humphrey et al. (1991) found that 'Pride [from continuous improvement] feeds on itself' and leads to success.
2.3.2 Why is Requirements Engineering Important?

Given the apparent potential benefits of requirements engineering on a project, it's not surprising that it has often been identified as responsible for the endemic project failures that seem to plague the software industry. The Standish Group, which claims to be “the largest body of primary research in the IT community ... spanning 40,000 projects, thousands of surveys, [and] hundreds of focus groups” (2004) published a report in 1994 on software project failure. The report describes the results of surveys of over 8000 software projects, of which 31% ended in complete failure (by being canceled sometime during their development cycle). Of the projects that had succeeded, four of the top five success factors included requirements related or requirements dependant activities, including: user involvement, clear statement of requirements, proper planning and realistic expectations. The report illustrates that it was the absence of these factors that led to project failures including: lack of user input, incomplete requirements, unrealistic expectations and changing requirements (1994). In 2001, The Standish Group released another report indicating that although only 23% of projects had failed, suggesting improved success in the industry, the factors of success and failure had largely remained the same as previously reported (2001). The apparently persistent challenge of carrying out good requirements engineering in industry has some wondering why there has not been more emphasis placed on improving industrial requirements practices.
2.3.3 The Need for Evidence

Researchers in the field are equally frustrated by industry's reluctance to adopt a requirements engineering philosophy: its concepts, tools and techniques. The annual International Requirements Engineering Conference has hosted panels considering the problem of promoting requirements practice. One of the first panels, held in 1997, concisely summarizes the issues:

The path from conceptualization of a good idea to its widespread use in industry is usually long, complicated and fraught with peril. Too often, research justified as satisfying the needs of industry begins with wrong or simplified understanding of real problems. (Miller, 1997)

Of course, this sentiment can be repeated for almost any academic field that claims to have a good idea. However, the push for requirements engineering adoption suffers from some unique challenges:

- It is hard to rationalize high up-front costs to do a thorough job of requirements analysis, given that the potential benefits of requirements engineering are largely unproven
- It is difficult to establish that requirements engineering methods have preserved or improved the quality of business products, services or processes
- Requirements engineering is inherently difficult

-(Morris, Masera & Wilikens, 1998)
Yet, despite the potential benefits of requirements engineering Kaindl, Brinkkemper, Jr., Farbey, Greenspan, Heitmeyer, do Prado Leite, Mead, Mylopoulos & Siddiqi (2002) suggests that many industrial organization remain unconvinced. Kaindl et al. conclude that a persistently outstanding obstacle to industrial adoption is the lack of concrete knowledge about what organizations can gain from applying requirements approaches.

The absence of supporting empirical evidence is not limited to requirements engineering. In their editorial introduction to a special issue of IEE Proceedings - Software Engineering, Kitchenham & Budgen (2002) begin by noting that “although [software engineering] employs concepts and practices that are drawn from experience and observation, we rarely possess any empirical validation of these ideas”. Kitchenham & Budgen go on to discuss the inherent challenge of collecting such evidence and the role of the case study research method as important methodological tool for detailed investigation of such phenomena.

Likewise, Tichy (1998) has expressed his dismay at “how much the computer industry and sometimes even university teaching relies on so-called ‘experts’ of all kinds, who fail to back up their assertions with evidence”. He also notes the role of the case study as a substitute for experimentation “when control is impossible.” Although he admits that some believe this method treads on ground that some consider soft science.

However, using qualitative evidence to support theory is not necessarily any weaker than quantitative evidence, according to Carolyn Seaman. In her analysis of qualitative research methods in software engineering (1999), she reminds readers that “a hypothesis cannot be proven, it can
only be supported or refuted.” She contends that “software engineers are apt to attribute more significance to a single statistically significant finding simply because empirical findings are so scarce in our field.” Qualitative methods provide powerful explanatory information and “help in refining a proposition to better fit the data.” To this end, to ensure the validity of research and its methods, Seaman emphasizes the important function of triangulation, the practice of gathering different types of evidence which support a particular proposition.

2.3.4 Related Empirical Work

This review presents a sample of existing literature showing how researchers have empirically investigated requirements engineering phenomena. Examining similar work justifies this research’s use of the case study to investigate the effects of requirements engineering. It also establishes how this research is unique and why it emphasizes using existing theories in literature to inform a systematic life-cycle-long, explanatory case study.

Empirical Research Methods

The most prevalent research methods in empirical software engineering and requirements engineering research are the experiment, the case study and the survey research method. In theory these methods are appropriate for investigating particular kinds of questions. For example, according to Yin (1994), the experiment is appropriate when the investigator manipulates some aspect of the event, the independent variable, to examine or explain some phenomenon. In contrast, surveys and case studies are both
appropriate for investigating issues where the researcher has no control; although the survey is uniquely suited to achieving a broad overview of the phenomenon. The case study is uniquely suited for investigating in detail the nature of complex phenomena over which the researcher has no control. The quintessential usage of the case study research method is to build holistic understanding of interrelated activities (Joe R. Feagin & Sjoberg, 1991). This pattern is largely evident in existing empirical work.

For example, well known empirical work in requirements engineering is often carried out via the survey wherein authors study prevalent trends in the industry. Herbsleb & Goldenson (1996) used surveys to poll industrial practitioners to determine whether CMM maturity correlates to product quality and customer satisfaction. In the same way Emam & Birk (2000) used the survey to investigate CMM and requirements practices. Although these studies are important by implying that there is a causal link between CMM and project success, they are unable to capture how or why that may be the case.

Experiment driven research such as Porter's work on comparing methods for inspecting software requirements is very effective when experimental control is possible (Porter, Lawrence G. Votta & Basili, 1995). More often than not, experimental research focuses on comparing particular tools and techniques to justify their effectiveness. Coincidentally, this type of work is often conducted by the author of the tool or technique in question. There are many examples of work that fits this theme, such as Boehm's experiments exploring the virtues of his Win-Win requirements negotia-
Related Empirical Studies

Our research approach distinctively follows the effects of requirements engineering throughout the development life cycle so as to explain how improvements in risk management, quality and productivity are realized. Furthermore, this research, in contrast to others, is an *in situ* analysis of real industrial software development organization who have implemented their own software improvements. This characteristic uniquely provides a more impartial perspective from which to observe the impact of requirements practice as it occurs.

Other empirical work in requirements engineering process improvement differs widely in both method and objective. For example, Lauesen & Vinter (2001) compare specific requirements techniques in terms of hours saved. Lutz & Mikulski's look at safety-critical anomalies provides insight into how to learn from system anomalies by examining software systems from a strictly post-deployment point of view (2004). In contrast, Herbsleb & Kuwana (1993) contribute an in-depth analysis of collaborative development practices, but do so to inform methodological support. Solid evidence on requirements engineering effects have typically been difficult to obtain because organizations rarely measure the costs and benefits of requirements engineering activities. Data released by NASA (source: W. Gruhl in (Fosberg & Mooz, 1997), p. 45) stands as

\footnote{According to Yin and the widely accepted definition of a case study, this experiment is mis-labelled as a case study because the investigator has manipulated the context by mandating the use of the Win-Win approach and related 'Win-Win tools'.}
an exception. It suggests that time spent on requirements engineering activities negatively correlates with project cost overruns. While these two characteristics provide provocative statistics, they provide only a bird's-eye view of 25 completed projects—failing to capture any details which could explain how requirements activities prevent such overruns.

Much existing empirical literature is specifically about comparing particular tools or techniques. Such work tends to focus only on the techniques being examined and, unfortunately end when the techniques initial outcomes can be observed. For example, the work of Porter (1995) and In et al. (2001) examine only the requirements phase of development. Although these works were experimental in nature, case studies often suffer similarly. In proposing a means to extract requirements from user interface prototypes, Ravid & Berry (2000) use a case study\(^2\) to examine only the extraction and subsequent documentation of requirements. Due to cost and complexity, most empirical research naturally attempts to collect empirical data as early as possible. Research that compares tools, techniques or methodologies tends to examine the immediate outcome of that tool, technique or methodology, rather than its long term impact on a project. Such emphasis is useful for comparison but does not address the need to understand long-term impact of requirements engineering on software development as a whole.

\(^2\)Actually a retrospective 'after-the-fact case study', wherein Ravid uses his technique on existing UI artefacts from an 'almost-complete' development project.
2.4 Summary

This chapter has established the relative importance of requirements engineering in the successful production of software. Experts, both academic and industrial, suggest that good requirements engineering can lead to benefits in risk management, productivity and quality. Yet, despite these widely repeated claims evidence to support them is scant, preventing the industrial adoption of requirements practices. Finally, a variety of possible research methods, which could be used to bring produce such evidence were considered. We contrast the kind of long-term detail-oriented evidence needed with that which has been more typically collected.
Chapter 3

Research Design

This chapter describes the research and the methodology used to carry out the work that this thesis describes. The research question, which considers the details of how requirements engineering affects software development, demands a structured framework in which to analyze the complex intricacies of a software project. The case study provides this structure (Yin, 1994). The majority of this chapter explains how the case study was designed: its propositions, phases of research, its unit of analysis and the full context in which the research occurred.

The next two sections describe the research question and propositions that guided the study. The case study's unit of analysis is described in Section 3.2.1. A description of the software project that was investigated, particularly with respect to their requirements processes is provided in Section 3.4. Details of how data was collected and its relation to the research question is provided in Sections 3.5 and 3.6.

3.1 The Research Question

The previous chapter shows that although there are strong claims that requirements engineering leads to more successful software projects, there
is little systematic, detailed evidence to support this position.

The goal of the research was to explore, explain and validate this presumed causal link between requirements practice and its benefits in risk management, quality and productivity. Unfortunately, requirements practice is tightly integrated with other development activities and complicated by organizational practices that occur over the course of a project.

Software development is usually a complicated endeavour wherein many different processes contribute to a final product. The elapse of time between requirements engineering activities that normally occur at the beginning of the project and the release of the final product further confound matters. It is fallacious to definitively conclude that requirements engineering leads to success through mere examination of the final outcome.

Tracing a causal link between requirements engineering and its benefits through a myriad of potential process interactions is too complex to study in isolation. It would be impossibly expensive and extremely difficult to carry out an in vitro experiment which could accurately replicate the scale and complexity of such a project (Basili, 1996). Instead such an endeavour requires in situ, in-depth, systematic analysis. Therefore, the case study, rather than survey or experimental techniques were chosen to structure the empirical research to answer the question:

*How does requirements engineering affect software development throughout the development life cycle?*

Unfortunately, although this question guided the study as a whole, it was too broad and abstract to steer this empirical work. As stated, the
question does not assume the necessary passage of time over which both the project proceeds and the research is conducted. In light of this, four additional questions were formulated over the course of the research.

The four additional research questions offered more focus by expressly considering the effects of requirements engineering at different stages of the project. Each question was constructed according to the research's current progress, representing an evolution of inquiry. Each progressive question was designed to clarify and understand phenomena that had been left unexplained by previous questions. Naturally, each question guided separate data collections and analysis, as described in Section 3.5.

To explore issues related to early adoption and process change early into the project the research considered:

1. **How does requirements practice impact the pre-design stages of development?**

This preliminary question sets the stage for subsequent work by considering the nature of the requirements activities being conducted and their subsequent effects on planning. In particular the study considered effects on software estimations, on project negotiation and on the ability of engineers to comprehend the major issues that the software project was meant to address.

By the time the design had been completed, the implementation was fully underway and testing had begun, it was possible to consider how requirements activities had affected development, namely design, implementation and testing. Data from the initial question provided some in-
sight into how developers in the organization felt about the new requirements engineering process and their project. To understand how the requirements process had affected their work, the study considered:

2. How does requirements engineering practice affect downstream development?

3. In what way did each component of the requirements process contribute to the evident effects and to development?

The focus on downstream development essentially includes all development activities that had already occurred, namely: design, implementation as well as those activities that might be described as project logistics such as project tracking and control. Since the project was coming to a conclusion it was a priority to determine how the requirements process had affected productivity and quality. At this time it also became possible to examine how the requirements process had affected the organization's ability to manage and control risk.

Endeavouring to answer the previous questions provided a wealth of information. A clear picture began to emerge detailing the effects that the requirements process had on the project. To shed further light on how exactly these effects had been realized, the study considered:

4. How could the interaction between requirements processes and other processes have contributed to the effects that had already been observed earlier in the study?

The challenge of explaining how requirements process affects software development is to understand such effects in context with the other activ-
Chapter 3. Research Design

ities that occur during development, for example: project tracking, testing or development practices. Given the effects that had observed by this point in the research, it became relevant to consider how the requirements process had worked in conjunction with other activities to produce the effects that had been observed. This research question motivated the research's last collection of data and the analysis that led to finally addressing the overall research question.

3.2 Case Study Propositions

According to Yin (1994), although the research question "provides an important clue regarding the most relevant research strategy to be used", the study propositions "direct attention to something that should be examined within the scope of the study". Propositions provide a basis for examining particular aspects of the phenomenon. By testing these propositions during the study, one begins to move toward answering the larger research questions.

The propositions used in this study all originate from existing assertions made in the relevant literature (see Section 2.3). The authoritative origin of these claims affords justification for their inclusion in this research. Since these claims largely pertain to risk management, quality and productivity – effects that can only be observed well into a project – these propositions apply primarily to the first and second research questions.
3.2.1 Problem Understanding

The most general and sweeping proposition about requirements engineering practice is that it provides a better basis on which engineers developing software can make decisions. In other words, they better understand the problem the project is supposed to address. This is sometimes called 'problem understanding'. Although problem understanding may not be easily detectable there are a variety of readily detectable by-products.

Better problem understanding leads to improved productivity, fewer development errors, fewer defects, and software that is 'more correct'. Correct software is software that addresses the problem it was meant to address and does so according to how it was specified. Better problem understanding also affects testing: this understanding results in tests that more accurately tests the functionality that the software is designed to provide, suggesting that requirements engineering leads to more complete feature coverage. Finally, requirements engineering can simplify communication patterns by allowing engineers to have common understanding enhanced by finalized, centralized and well-defined requirements artefacts.

3.2.2 Accurate Estimation Gathering

By considering the details of feature commitments during project planning (consideration that might otherwise be left until design) it is thought that more accurate estimations can be made by engineers. Since project planning estimations are almost always a product of the estimations provided
by engineers, it follows that more accurate estimation produces more accurate project plans.

### 3.2.3 Schedule Commitments

By solidifying project commitments during the planning stages of the development process, planning estimations are more accurate by virtue of the more limited, controlled change during project development. Projects without such commitment suffer from continuous change arising from both emergent customer demands and enhancements that originate internally from developers. Given continuous change it becomes impossible to predict the final product or its delivery.

### 3.3 Unit of Analysis

This section describes the unit of analysis used to define this case study. Every case study has a unit of analysis, referring to the “case” that is being examined (Yin, 1994).

This study’s unit of analysis is a single software project over the course of its life cycle from early planning, through development and implementation, to deployment. However, it is difficult to delineate a precise ‘unit’ because examination of the project requires an analysis of the environment in which it is being developed. So, to address the research question this case study must also consider variables that transcend the boundaries of any single project. In addition to describing the project itself, the organization and its software development process are all separately discussed.
3.3.1 The Company

The software development project was undertaken by the Australian Center for Unisys Software (ACUS), located in Sydney, Australia. ACUS is a software development organization that is part of a much larger multinational corporation, Unisys Systems whose headquarters in the United States of America. ACUS' mandate is to develop software of strategic interest to its parent company and it therefore receives direction from marketing business units in the US.

The American marketing unit acts as proxy-customers to ACUS, providing a single entity from which requirements originate. When a new release is being planned, features originate exclusively through this entity. This arrangement is partially to coalesce the varying demands from a wide range of customers that are distributed around the world, and partially to capture the strategic corporate concerns of the company. Although ACUS designs, develops, tests and delivers software, it has no direct access to its users during development beyond its marketing unit. After development, however, ACUS does provide support and receives feedback from its customers in order to address customer concerns and manage maintenance of its software releases.
3.3.2 The Product

ACUS develops a successful software product line that has a 20 year history. It is a large multi-platform product, approximately 4 million lines of code and it is written in a variety of languages including: Java, C++, Cobol, Algol and Smalltalk.

The software product itself is an application development platform supporting the development of enterprise-scale, transaction intensive applications. The product is typically customized by customers or third-party developers before usable by end users.

3.3.3 The Project

This research's subject of investigation, the case study's unit of analysis is a development project at ACUS that lasted from September 2001 until approximately May 2004 (including a 13 month post-deployment period). The purpose of the project was to design the next iteration of its flagship product, and it essentially involved the entire organization. This project, like past ACUS projects, consisted of a variety of feature additions and enhancements. In the estimation of managers from ACUS, this iteration was of similar size and complexity as recent past projects.

The terms of the research stipulated repeated interactions with managers and engineers (e.g. interviews, questionnaires, observation, etc) and access to project planning documentation (e.g. process documents, planning schedules, requirements specifications, change request artefacts, etc.). The terms did not provide access to source code, defect descriptions, de-
signs or test artefacts. Therefore, this research did not analyze those particular production artefacts in detail, although in some cases engineers were asked to comment on their production.

3.4 ACUS' Requirements Engineering Process

The opportunity to conduct research at ACUS was extremely fortunate, since the organization had recently revised its process. This provided the chance to examine how this change in process would affect its software development practices. The following subsections briefly document how projects had normally been conducted in the past, ACUS' concerns about those practices, and the process changes that were made to address those concerns. The following are documented as conditions of the case study because these changes were made wholly by ACUS, independent of this study or its researchers.

3.4.1 The Former Process

In August 2001 ACUS engaged in a software process improvement initiative with the goal of gaining CMM level 2 certification. A CMM mini-assessment which had been conducted at the company in July 2002 indicated requirements management as a Key Process Area that required significant improvement.

Prior to this initiative ACUS effectively had no well-defined requirements engineering process. Despite having major stakeholders spread across several continents (North America, Australia and Europe), the prod-
uct development group had limited experience with formal requirements management processes. Management at the company expressed their concern that recent projects at ACUS had consistently suffered from significant cost and schedule overruns.

ACUS had difficulty understanding the requested features and providing reasonably accurate development estimates. Managers at ACUS cited ineffective negotiations between ACUS and their marketing unit. Ineffective negotiations made it difficult to align development capacities and marketing needs.

In particular their requirements practice suffered in the following ways:

- requirements were being provided to ACUS in the form of one or two line 'system features'
- ambiguous requirements were almost impossible to scope
- expectations of project commitments were poorly managed because of ineffective negotiations
- inadequate requirements management and control compounded these problems by enabling the US marketing unit to demand new features late into the development cycle
- poor requirements management and commitments to inadequately understood requirements
- any collective understanding of features among developers largely happened accidentally via word of mouth, rather than by design
Although this list captures the major challenges identified by the CMM mini-assessment, it does not capture their compounded effects. For example, requirements creep during the project frequently caused ACUS to drop functionality that it had already committed so that the project would remain on schedule. Subsequently these actions resulted in tension between ACUS and its marketing unit, who perceived these failures as a sign of incompetence.

3.4.2 Process Revisions

In response, ACUS revised its requirements engineering process to specifically address these challenges. Requirements engineering was elevated as a central facet of their development regime. The revised process defined a distinct, discrete phase of the project during which requirements would be elicited, analyzed and negotiated. When this requirements phase had ended, requirements were to be have been agreed on, committed to and then baselined during project planning. Subsequent changes would only be allowed after being approved in a formal requirements management process.

The details of their major process changes are summarized below.

Cross functional Teams

Teams responsible for implementing features were re-organized. Whereas previously individual would implement a particular feature on a particular hardware platform, under the new process teams were responsible for implementing a complete requirement across all platforms. Teams were
encouraged to openly cooperate and communicate with teams in other functional departments (i.e. testing and documentation); this occurred primarily through group-analysis sessions attended by representatives from all functional departments.

Group Analysis Sessions

Group analysis sessions were used to analyze feature requests made of the marketing unit. These sessions, attended by a cross-section of functional departments including management, developers, testers and documenters, were designed to decompose feature requests into individual software requirements. During these sessions, technical issues or questions would be raised, discussed and documented so that they could be clarified in collaboration with the marketing unit during negotiation sessions. However, the majority of effort was spent conducting very high-level design to understand the impact that features would have on the architecture of the existing product.

Defined Specifications of Requirements

Also during these analysis sessions, requirements were described in structured requirements documents. The description of each requirement was articulated using a language template to limit the ambiguity of free-form natural language. (Please refer to the following sample requirement and sentence template example.)

1At ACUS, most requirements were mutually exclusive of feature, likely owing to the maturity of their software product.
### Sample Requirement:

**Initial Feature:** Scriptable Interface

**Derived Technical Requirement:** EA Developer shall provide a version of the D2L utility that allows the exchange of affected screen definitions from Graphical Interface Workbench to EA Developer.

**Rationale:** Customers may already have painted versions of the affected screens in GIW. They will likely wish to keep those existing painted screens, and be able to enhance them in the future.

**Test Scenario:** Enhance screens in a GIW environment. Load the provided model into EA Developer. Use D2L to transfer the enhanced screens from the GIW environment to EA Developer, into the installed model. Verify that the enhanced screens are available in EA Developer and can be further enhanced in EA Developer Painter.

---

<table>
<thead>
<tr>
<th>An unrelated example of parsing a full text requirement into the sentence template (from Halligan, R. (2000))</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Full text requirement:</strong> When in full operation the computer system shall provide thirty per cent reserve in channel capacity.</td>
</tr>
<tr>
<td><strong>Initiator of Action:</strong> The computer system</td>
</tr>
<tr>
<td><strong>Conditions for Action:</strong> in full operation</td>
</tr>
<tr>
<td><strong>Action:</strong> shall provide</td>
</tr>
<tr>
<td><strong>Object of Action:</strong></td>
</tr>
<tr>
<td><strong>Refinement/Source of Object:</strong></td>
</tr>
<tr>
<td><strong>Refinement/Destination of Action:</strong> reserve in channel capacity of 30%</td>
</tr>
</tbody>
</table>
Traceability Links

To preserve the relationship among dependent requirements, traceability links were established and maintained within their requirements tool, Requisite Pro. These links connected the requirements document to both requirements rationale and to test scenarios that were specified for each requirement.

Test Scenarios

During analysis sessions, test scenarios were conceived to test the functionality of the feature and its individual requirements. These scenarios were linked to requirements with traceability links as described above. During development, these test scenarios would provide the fundamental basis of the actual tests used during product testing to both ensure quality and to confirm that the requirement had been realized in the final product.

Change Management

The final major component of the revised process included a completely new change management process to carefully control the revision of requirements once development began. Requirements changes originating from either development or management required formal application and approval through the change management process. This process required that the change request be articulated in a request document containing a description of the change, rationale for the change, the extent of the proposed change and an estimation of effort to implement the change. These
requests were then reviewed and subject to the approval of a software change control board. This board included engineers from all functional departments, upper-level management and even representatives from the marketing unit. Only after approval by the board and the Project Manager, could the change proceed to implementation.

### 3.5 Data Collection

Data collection occurred primarily during three separate phases of investigation (see Figure 3.1). During each investigation a variety of material was collected, but the most significant source of data originated from three surveys conducted during each of the phases. These questionnaires invited members of the project to comment on the project and the revised requirement engineering process.

During each investigation a variety of project members from each functional department were invited to participate, including management, design, implementation, test and documentation. ACUS enjoys low employee turnover and many participants were personally familiar with the 15 year history of the product and with the software development practices used by the team to develop that product. To leverage this experience, in lieu of quantitative historical data, questionnaires often asked respondents to compare aspects of the current development to previous projects.

The particulars of the data collected during each phase of investigation are described below:
Figure 3.1: Timeline showing development and research phases

**Initial Phase**

The initial phase of research was conducted between May of 2001 and July 2002 by Dr. Daniela Damian. During this period Dr. Damian collected data as an on-site participant observer at ACUS. Her initial work (Damian, Zowghi, Vaidyanathasamy & Pal, 2002) consisted primarily of an observation of the organization’s requirements analysis and negotiation sessions. She also administered questionnaires, conducted numerous interviews of managers and engineers, and collected process documentation. Thirty-four software engineers, documenters and managers took part in interviews and questionnaires during this period of time.

Due to her other professional commitments, in 2002 Dr. Damian became unable to devote the time necessary to conduct the ACUS research
by herself. At that time I continued her work by conducting subsequent research after September 2002 with ACUS under the supervision of Dr. Damian. My work constitutes the intermediate and final phase of research and is the primary topic of this thesis.

**Intermediate Phase**

The intermediate phase constituted the largest collection of data. A questionnaire that was administered to thirty-one project members constitutes this study’s most significant single data collection (reproduced in Appendix A). This questionnaire was primarily administered electronically (via Word document), although some managers were given paper copies at their request. People from all functional departments participated in this questionnaire, including engineering, testing, product documentation and management. In addition to the questionnaire, a series of interviews were also conducted. All interviews were open-ended, semi-structured and recorded. Subsequently, interview data was analyzed, integrated with qualitative responses from the questionnaire and informally examined for patterns of responses.

A selection of project artefacts was also collected at this point. This included requirements process documentation, requirements specifications, change requests, and entries from software tool used to manage requirement artefacts. In addition, some project statistics were also available, including development defect rates and project planning documents that included scheduling and effort forecasts as well as cumulative real effort spent during each functional phase of development (i.e. on design, imple-
mentation, test, etc).

Final Phase

By the time the final phase of investigation took place the project had completed and thus access to project engineers was very limited. To minimize disruption and simplify distribution, the final questionnaire was administered via the internet, and made available to 20 subjects of whom 15 participated (see Appendix B). This questionnaire exclusively focused on the extent to which the requirements process had affected other development processes. Therefore, managers, technical managers and team leads where solicited to participate since, given their perspective, they would be more likely to be able to judge the subtle process interactions.

Additional project materials were also collected at this time including statistics on released defects and customer service requests.

3.5.1 Ethics

The majority of evidence was collected anonymously to alleviate the perception by respondents that their answers might be scrutinized by the organization. To address the potential for apprehension among participants who wished to answer honestly and perhaps critically, participants were informed that their names would be kept confidential. All communication involved in data collection occurred between the participants and the researchers conducting the study, even in the case where physical documents were exchanged. Finally, all respondents were instructed that the purpose of the research was to improve development practices and to ad-
3.6 Aspects Investigated

The aspects of investigation describe in detail how the case was investigated relative to the study propositions (see Section 3.2) and the specific context of the ACUS project. Rationale is also provided to explain why these aspects were investigated.

Aspects are organized by the phase in which they were studied in order to show the progressive development of the study. Each phase of research conferred further evidence toward an understanding of what was happening at ACUS, informing the next stage of research. This iterative approach provided the opportunity for the research to evolve. Each phase of research was designed not only to corroborate emerging patterns, but to build toward a deeper, more detailed level of understanding of how their requirements engineering process affected their project.

3.6.1 Initial Phase

The primary phase of research provided an important first-look into the inner workings of ACUS. This stage was primarily exploratory, during which a picture of ACUS and its processes were developed.

During this research phase ACUS was actively conducting its requirements engineering activities through requirements analysis sessions and
negotiation sessions with its marketing unit. Observing these interactions provided insight into ACUS' requirements practices.

Early in the project while the new process was still being introduced and integrated in the organization, engineers and managers were assessed to determine how they perceived the new, as yet, untested process. The initial questionnaire administered during this period was designed primarily to determine in what ways the engineers anticipated the new process would help them. A second questionnaire, administered toward the end of the phase, provided engineers with an opportunity to indicate how the revised process had helped them to conduct early requirements and negotiation activities.

It became possible to determine how engineers' perceptions had evolved as they used their new process by comparing the responses made by engineers in each of the two questionnaires (Damian et al., 2002).

3.6.2 Intermediate Phase

General Feedback

This questionnaire represented the first opportunity to survey engineers who would have, at that point, have fully experienced the revised requirements process. The questionnaire begins by determining, first at a very general level, how engineers felt about the revised requirements process and its long-term effects.

Questions also asked engineers to rate the importance they placed on the revised processes, how it affected the work of other project members
Problem Understanding

To determine whether engineers were better able to understand the problem that they were in the process of building a solution for, the questionnaire asked engineers to comment whether the process had revealed feature details and how important that information was to each development activity (design, implementation, testing and documentation). Furthermore, engineers were asked to reveal how they used that information: whether it facilitated informed decisions, their sense of ownership, their sense of accountability or their creativity. In addition, to determine how engineers used requirements they were asked whether requirements were used as a reference to re-familiarize themselves with particular features, to gain deeper understanding of the feature, to understand the motivation or to ensure requirement compliance.

Existing theory suggests that there are a variety of tangible beneficial effects that stem from engineers who understand their problem. To test for these effects the questionnaire asked whether their improved understanding (if that had been the case) had perceptibly altered the amount of effort spent revising or reworking artefacts.

In addition to the production of development artefacts, engineers spend a great deal of time communicating with one another. Therefore it is reasonable to speculate that if improvements lead to reduction of artefact revisions then engineers may also spend less time communicating with one
another. Or, perhaps, if they did not spend less time spent communicating, then they would at least enjoy more efficient communication. During the initial phase, some engineers had commented that a lot of time had been spent 'clarifying' requirements amongst themselves. The questionnaire followed up on this insight by asking engineers to comment on how communication patterns had changed compared to previous projects, with respect to their necessity to seek clarifications. During the initial phase, engineers had generally responded positively to the cross-functional communication that had been fostered by the group analysis sessions, so this questionnaire asked engineers to consider whether that communication had continued on into the project. Finally, engineers were also asked to speculate how improvements in communication had affected quality and productivity.

Estimation

If requirements engineering activities are conducted at the beginning of a project, and are meant to provide understanding about what the project will deliver, it follows that this information could profoundly affect planning activities that also occur early in the project life cycle. Estimations of effort to implement product features are a critical input for project planners to utilize.

To investigate the extent by which the requirements engineering process improved estimations, team leads and managers were asked how important the revised requirements process was to them in providing estimations. To explore to what extent estimates may have been affected by
aspects other than requirements engineering one must also consider what other practical matters may have affected estimations. The questionnaire also asked these respondents to speculate as to why estimations may have been inaccurate and what steps they would take to improve the accuracy of such estimations in future.

During this phase of investigation effort estimations documents were made available by the organization. These documents contained the project estimates that had been made before requirements analysis had occurred, and estimations that had been made after requirements analysis had occurred. The document also contained 'cumulative effort', a record of effort that had been spent by engineers on the project. This document was analyzed to compare how accurate requirements estimations were and how they compared to the estimations made before requirements analysis had occurred.

Requirements Change Management

No project can operate in a vacuum, immune to the developments and events in the world around it. Requirements change management was a new initiative at ACUS, a part of the requirements process improvement. Like all change management programmes its purpose is to manage change, preventing unnecessary project changes that could potentially derail project schedules, while providing the flexibility to respond to emerging issues. It has been shown that the longer one waits to address requirements errors, the more expensive they are to fix (Pressman, 2004), suggesting that it is important to address necessary changes as soon as possible
to minimize the costliness of making requirements corrections. Finally, failure to capture and adapt to the ever changing context of the software problem have often been blamed as significant challenges to project success.

During this investigation, the organization provided extensive requirements change management artefacts for analysis. These documents consisted almost exclusively of change requests. The investigation utilized this resource to calculate change request and change approval statistics. Furthermore, since the request documents included comments and approvals of managers, team-leads and engineers who presided over the change request, it was possible to determine how rigorous the change approval process was in practice.

The questionnaire administered during this phase also included a small section on change management. Managers and team-leads were asked whether the revised requirements process had helped them to assess requirement change impact.

3.6.3 Final Phase

During the intermediate phase of investigation, one of the more interesting findings related to the relative effectiveness of the various components of the requirements process. This finding helped to shape the focus of this phase of investigation toward determining how the components of the revised requirements process affected particular processes and how the interaction between the requirements process and other development process contributed to the holistic effects that had been captured during
the previous phase.

This final phase of investigation was constrained by the requirement that the questionnaire be relatively short (compared to the previous one), and that it be administrable via the internet (in particular the world wide web). Keeping the questionnaire concise yet comprehensive and comprehensible yet meaningful proved to be a major challenge.

The questionnaire designed for the final phase of investigation had a far more structured, consistent form that asked team-leads and managers to rate the effects the requirements process had on other development process, according to their perceptions. In addition to rating 'effect', respondents could also indicate that a particular component had contributed to the 'effect'.

Put simply, the questionnaire consisted of a list of all development processes in each of the organization's five process areas. For each of these processes respondents could choose a value, between -3 and +3 indicating the degree of effect the requirements process had had on that process. Subjects were instructed that negative values represented hinderance by the requirements process, while positive values represented beneficial impact of requirements process on the process in question. For each process, respondents could optionally choose one of five possible REP components to indicate which REP component had predominately contributed to the rated effect.

For example, in response to the process 'tracking' a respondent may choose +2 for 'REP Impact'. Such a response would suggest that requirements process had a moderately positive effect on project planning. Fur-
thermore, had the respondent chosen 'traceability' as the contributing REP component, such a response would indicate that of all the REP components, traceability had been primarily responsible for the moderately positive effect on project planning.

To complete the documentation of the questionnaire, the remainder of the subsection explains which REP process components were included in the questionnaire. Each process is documented according to their meaning within the context of the development organization.

**REP Components**

Among the REP components that were considered during this phase, and available for respondents to choose from, a simplified set of REP components from the previous questionnaire was used. Recall that in that former questionnaire, respondents were asked to rate the relative importance of each of eight REP components. In this investigation, some REP components that had been identified as less important were combined so as to minimize respondent choice and providing only distinctive, significant options.

Elements of the requirements document, which scored consistently low during the intermediate investigation, *Language Template, Structured Requirement Specification* and the *Context Diagram*, were coalesced and referred to simply as *Structured Requirements*. Similarly, the distinction between *Rationale Traceability* and *Traceability to Test* seemed too slight to be separately included, instead these components were succinctly combined into *Requirements Traceability*. 
Thus the following REP components were selectable as contributing REP sub-processes:

- Feature Decomposition
- Requirements Traceability
- Group Analysis Sessions
- Cross Functional Teams
- Structured Requirements
- Testing According to Requirements

Development Processes

The following provides a description of each process area and descriptions for each of the area’s constituent sub-processes. Some sub-processes are followed by an abbreviated short name in brackets, this short name is used in the charts and tables provided in Section 4.

**Project Planning and Tracking:** This process area is largely a managerial responsibility wherein the software project is initially planned, schedules formulated, resources allocated and milestones determined. Once the project is under-way, subsequent tracking and monitoring of the project progress also falls within this process area. Seven sub-processes of project planning and tracking were identified:

- feature sizing (sizing): estimations of required effort to design, test, document and implement features or requirements
- risk assessment (risk): determination of the technical risk of implementing particular components of the software system

- scheduling: planning the length of time for specific project phases to complete

- resource planning (resources): allocating developers to specific features and project roles

- change management: review and control of change requests from developers

- responsibility allocation: assigning lead roles for the implementation of particular features

- requirements tracing: ensuring that traceability links between requirements and other design artefacts are maintained

**Software Quality Assurance:** This process area's role is to track software design and development, specifically with respect to ensuring a high standard of quality is maintained. Four sub-processes of software quality assurance were identified:

- tracking: monitoring the progress of implementation and feature testing

- SQA team (teams): the formation of a team responsible for SQA

- meetings and reports (meetings): regular meetings and reports are conducted and produced by the SQA team regarding software quality during development
Chapter 3. Research Design

- deviations: review and control of process and major project deviations during development

**Software Configuration Management:** This process area’s role is to maintain consistency of project artefacts and in particular to assure effective software configuration management (SCM). Artefacts subject to management not only includes source code artefacts, but all formal project documents including project plans, reports, designs, test cases, etc.

- baselining: determine base software versions and milestone feature sets
- change management: review and control of change requests from developers
- change metrics (metrics): provide rudimentary measure and extent of change in code artefacts
- role adherence (roles): monitor and ensure that development roles are being fulfilled, particularly with respect to those responsible for the SCCB (see below) and managerial issues
- software change control board (SCCB): this group of people were created to review, discuss and approve developer change requests
- SCM tools: prescribe the use of a tool of set of tools to manage version control on intermediate development artefacts such as documents and source code.
Development: This process area constitutes all aspects concerning the design and implementation of the software product, but does not include documentation or testing which are both addressed by separately.

- Team reorganization (teams): as a part of the revised requirements process development staff were reorganized into teams responsible for implementation of requirements

- Split leads: organizationally, lead development responsibility is shared by two separate individuals, one responsible for managerial issues, and one responsible for technical issues

- Planning: emphasize and follow plans used to direct software development

- Conformance: software developers and designers are expected to follow and be guided by feature proposals, requirements specifications and design specification.

- Inspection: development artefacts, such as source code, are inspected for defects

Testing: Processes related to testing are responsible for the development test scenarios, test plans, and test execution to ensure quality and validate software products.

- Automation: some testing was conducted in an automated fashion

- Requirements Validation (validation): test scenarios and detailed test cases were written against requirements to validate promised functionality
Peer-Review: test artefacts such as test scenarios and tests themselves were peer reviewed.

Repeated Testing (repeated): many tests were repeatedly conducted to validate the maintenance of functionality (i.e.: Regression testing).

3.7 Summary

This chapter has explained the research methodology and context. Introducing the research questions has established the study's guiding central theme: how requirements engineering affects software development. By using the case study research method this work endeavours to collect solid empirical evidence from the ACUS project. Using existing requirements theory as propositions to guide the research serves to justify the collection and analysis of evidence to determine how ACUS's requirements practices affected their ability to manage risk, produce quality software and maintain productivity.

In the following chapter, the evidence collected over the course of this research is presented.
Chapter 4

Case Study Findings

So far we have considered the theoretical claims that advocate the practice of requirements engineering. So too, we have considered literature which explains and promotes the best ways to conduct and adopt requirements engineering practices. In the preceding chapter the research's design is described to test how well these tenants fair in the harsh, messy reality of software development.

This chapter presents the findings of the case study. These findings are organized according to the phases in which the findings were collected. For the sake of completeness, as much information about the data are included (i.e. the number of responses, the originating artefact, etc). The questions that comprise each of the questionnaires all stem from the particular aspects the research was designed to address. To understand the context and rationale for findings stemming from questionnaire questions, cross-references to the appropriate 'aspect of investigation' described in Section 3.6 are provided. Please note that this chapter includes a presentation of evidence collected over the course of the research, a discussion of this evidence and how it contributes to answering the research question is provided in Chapter 5.

The findings from the intermediate phase of investigation (Section 4.1
Chapter 4. Case Study Findings

to Section 4.6) provided extensive insight into how the revised requirements process had affected software development practices. Evidence found in planning artefacts as well as the qualitative and quantitative responses to each questionnaire show how the revised requirements process affected the project at ACUS. The most significant of these effects related to more accurate estimations, fewer defects, more efficient communication and improved understanding among developers.

Data and comments originating from specific questions found in the intermediate questionnaire are indicated with Q# where the number refers to the particular question (please see Appendix A).

4.1 General Perceptions of the Requirements Process After Development

The revised requirements process amounted to a significant change for engineers who had been used to working independently but who were now required to work together as a team of other developers, responsible for the complete fulfillment of particular requirements. After spending a significant amount of time participating in group analysis sessions while engaged in an extended period of requirements engineering, general feedback from engineers sets the tone of their perceptions of the process as a whole.

91% of 24 participants felt that the revised requirements process was important (Q1) and 71% of 22 engineers would spend even more time on the requirements phase in the future (Q3). In participants’ words, the
contribution of the new process (Q2) was as simple as helping to "discover goals more easily", "make the boundaries of features clearer" or "understand what needed to be done". For some the important effect is to broaden developer thought to a more comprehensive perspective: "[the requirements process] made me more aware of the impact in other areas, it made me think of the total package", "[requirements] formed a useful framework which underscored the whole design".

4.2 Problem Understanding

4.2.1 Details, Dependencies and Complexities of Features

Engineers unanimously agreed (100% of 23 respondents) that the requirements process revealed further details, dependencies and complexities of features (Q5). Chart 4.1 (23 respondents, Q13) shows the percentage of responses on the importance of understanding requirements in later phases of development, indicating that many engineers felt that the revised requirements process was a particularly productive and worthwhile use of resources. The requirements analysis sessions "helped to identify dependencies between requirements very early", and "identify features that would have otherwise been unaccounted for until much later".

4.2.2 Wasted Effort

In terms of tangible benefits evident in reduced rework, results were also very positive. While a majority of respondents (65%, 23 respondents, Q7)
felt that there had been less rework under the revised process, 10% indicated there was more. Some respondents qualified their choice: “there is always some level of rework due to inevitable technical issues”.

4.2.3 Improved Communication

In the earlier assessment, intra-team and inter-team communication was perceived to be improved due to revised requirements process. When asked whether this effect continued beyond requirements and into later stages of development, most developers agreed (24 respondents, Q9). Of those who responded neutrally and negatively to this question, half were from test and documentation departments within the organization. This significant representation is explained by comments made by managers who indicate that testing and documentation were sometimes left out of communications during analysis sessions.

When asked about communication, respondents felt that ‘clarification’ communication had been reduced, but that confirmations were still sought
Chapter 4. Case Study Findings

(Q10). The requirements provided a broader perspective and answered questions relating to rationale. As Chart 4.2 (23 responses, Q6) shows, requirements were used extensively by developers, most significantly to re-familiarize themselves with the characteristics of the feature (85%). This suggests that developers sought the re-iteration of feature information and that they were able to utilize the requirements artefacts to aid them in that task. Engineers also used requirements to validate coverage of features (90%). This confirms the requirements' role as a means to checklist system functionality during testing and review. Engineers responded favourably in other areas too, reporting that requirements had an impact on deepening understanding, facilitating rational designs and providing feature rationale.

Chart 4.2: Agreement that requirement artefacts enabled post-requirement activities
4.2.4 Informed Decisions

Further data shown in Chart 4.3 (21 respondents, Q8) indicates that improved feature understanding contributed significantly to developers' ability to make informed decisions. A somewhat less convincing though still positive result was with respect to the process helping to foster a sense of accountability (64% of responses). Creativity and ownership enhancements were less sure, but as one manager suggested this may have been due to the perception that the more rigorously defined requirements provided less room for creativity among designers and implementers. Improved understanding of the features also enabled ACUS to negotiate with its US-based product managers from a position of strength, given that the negotiations were based on firm data. These negotiation sessions were shorter given that the discussions were centred on tangible information. The scope creep was minimized as the requirements analysis sessions helped ACUS to understand and define very clearly what was being captured.

Chart 4.3: Agreement that requirement had intangible benefits
4.3 Estimations

The responses from 15 team leaders and managers provide evidence of strong agreement that the requirements were an important element of estimation. In particular, Chart 4.4 (13 respondents, Q14) illustrates that over 85% of respondents found that the thorough analysis of features was important in estimating effort required during design and implementation. Respondents were appreciative of the more thorough feature definitions and the added focus that the requirements process provided, ultimately preventing the need for extensive guessing and unwarranted commitment to feature development. Managers attributed fewer estimation errors to systematic estimation of smaller units (i.e. technical requirements), a benefit of the improved requirements process.

![Chart 4.4: Importance of requirements in estimation for design, implementation, testing and documentation](image)

In addition to qualitative responses, planning artefacts were analyzed to collect quantitative data available on estimations made during the project. Over the course of the project ACUS made estimates of effort required
to complete design, coding test and documentation. These estimations were made (1) during an initial feasibility study, before starting the new requirements process and before requirements analysis, and (2) after requirements analysis had completed using the revised requirements process. Finally, actual values were calculated at the end of the project. These estimates and actual values were collected for each of the project's 26 features. Chart 4.5 shows the difference between actual expended efforts and the effort estimations made before and after requirements analysis. Values above the x-axis correspond to optimistic estimations that are lower than actual required effort (ie. actual – estimate > 0).

Chart 4.5: Estimation error made before and after requirements analysis

Chart 4.5 shows that the estimations made after requirements analysis were closer to final effort, while the estimations before requirements analysis were predominantly pessimistic. Further, Chart 4.6 below ignores 'the direction' of error, instead showing absolute estimation error on the left-hand y-axis (referred to as 'delta pre- and post-analysis') and cumulative [absolute] estimation error (shown by horizontal dotted-lines) on the
right-hand y-axis. In particular, the cumulative error clearly indicates the superior accuracy of the post-analysis estimations as compared to the pre-analysis estimations. While it is true that one would expect estimations to improve as the project progresses, no development effort occurred between these estimates except for the requirements analysis phase itself. This leads us to conclude that the improvement in estimation can be primarily attributed to an enhanced ability to use the requirements in the estimation process.

![Chart 4.6: Absolute (and cumulative) estimation error before and after requirements analysis](chart)

4.4 System-Test Defects

Managers had informally indicated that there had been noticeably fewer pre-deployment, system-test defects during this project than in past projects. Unfortunately, this defect data was not made available for analysis, so questionnaire respondents were asked to recall the number of defects they
Chapter 4. Case Study Findings

had dealt with. In addition, respondents were encouraged to consider which aspect of development these defects originated from: requirements, designs, implementation or somewhere else.

Chart 4.7: Proportion of pre-deployment defect types, according to respondents

Responses from Q18 (15 respondents) provided results indicating most defects were code related. According to respondents less than 7% of defects could, in their opinion, be traced to problems in the requirement (see Chart 4.7). Many respondents commented that they believed this improvement to be a result of “improved product quality”, “more effective peer review”, or “better, more understandable requirements”. Although some testers expressed disbelief due to the increases in testing over the previous project, they also indicated that “schedules were more realistic” and that “engineers [were] more aware of how their piece fit in with the rest”. 
4.5 Requirements Creep

To gain insight into the organization's experience with the change management portion of their revised process, the organization's change management artefacts were collected and analyzed. Recall that the new process required change requests to explicitly document the nature of the change, rationale and description of the artefacts that would be affected by the change. In addition, each request document also included comments and approvals provided by engineers who were assigned to review the change request. Analyzing change request artefacts provides a means to scope the extent of their use of the change management processes; how changes affected other development artefacts; and the extent of involvement in the review process.

The rigour of ACUS' change management process and its role in preventing requirements creep is evident from their change management artefacts. Over the course of the development life cycle, 77 change requests were created: 2 during requirements definition, 37 during design, 31 during coding and unit testing and 7 during integration testing. As shown in Table 4.1, 158 documents were cited: 50 of them involved modifications to the requirements specification, suggesting that the structured requirements and traceability links were effective.

It is apparent that in practice each change request was extensively reviewed by many engineers from a variety of functional roles. On average, approximately seven engineers and managers were involved in every change request. As shown in Table 4.2, almost all change requests involved participation by the engineering managers, program manager,
Chapter 4. Case Study Findings

<table>
<thead>
<tr>
<th>Artefacts affected by change</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements Specifications</td>
<td>50</td>
</tr>
<tr>
<td>Functional Specifications</td>
<td>50</td>
</tr>
<tr>
<td>Design Specifications</td>
<td>16</td>
</tr>
<tr>
<td>Documentation</td>
<td>13</td>
</tr>
<tr>
<td>Test Plans</td>
<td>6</td>
</tr>
<tr>
<td>Other (e.g. project plans)</td>
<td>23</td>
</tr>
</tbody>
</table>

Table 4.1: Artefacts affected by change requests

Project managers and perhaps most significantly, testing (40%) and documentation groups (49%).

<table>
<thead>
<tr>
<th>Functional Role of Participants</th>
<th>Proportion of requests in which role participated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engineering Manager</td>
<td>100%</td>
</tr>
<tr>
<td>Project Manager</td>
<td>91%</td>
</tr>
<tr>
<td>Product Manager</td>
<td>65%</td>
</tr>
<tr>
<td>Testing</td>
<td>40%</td>
</tr>
<tr>
<td>Documentation</td>
<td>49%</td>
</tr>
</tbody>
</table>

Table 4.2: Change requests involvement according to role

The thoroughness of each change request submission most likely contributed to the reviewers' ability to assess requests: 7 out of 8 team-leads indicated that the requirements process provided them with enhanced ability to assess the impact of changing requirements.

Although only 12% of requests were rejected or withdrawn, comments provided by engineers suggest that not only did the process make change management more documented and rigorous, it effectively discouraged all but the most necessary changes. As one engineer commented, "change
control process made changes more documented ... fewer changes were submitted, it made people analyze about [sic] the changes before they were approved”

4.6 Post-deployment Indications

Post-deployments statistics gathered by ACUS as a part of their project management activities give an indication of how the project’s product had been received by customers. ACUS formally collected post-deployment defects and customer support requests, and informally rated customer satisfaction during the 13 weeks following product release. This has been a well-established practice at ACUS so defect and support rates were made available for both this project and the two previous.

Although many factors affect customer satisfaction and its measurement is hotly debated, the management at ACUS indicated that users appeared to be generally more satisfied with this release than with the last. This sentiment agrees with the marked decrease in support requests and reported defects.

At the end of 13 weeks following product deployment, ACUS’ data shows a 45% decrease in user support requests compared to their previous two projects. Reported defects showed similar improvement, down 48% over the last project and down 59% compared to two projects previous. These are encouraging results if, as engineers and managers reported, the current project was of comparable scope and complexity as previous projects.
4.7 The Role of REP Components in Contributing to Observed Effects

Finally, Q4 seeks to determine, according to ACUS engineers, which particular component of the requirements process was more important. To accomplish this, respondents were asked to select what parts of the revised requirements process were most important or useful in their work. For each main development activity (design, implementation, testing and documentation) respondents could select one or more REP component.

Although responses to other questions show that there was general agreement that the revised process was important, it is clear from the results (aggregated across development activity) shown in Chart 9, that some aspects of the process appeared to be far more useful to developers than others. In particular, the importance of cross functional team involvement, analysis sessions and conception of test scenarios during requirements clearly stands out.

These findings suggest that different REP components were clearly more important than others and that their importance depended heavily on the particular development activity being carried out. For example, the responses showed that test scenarios were particularly important during implementation. This insight guided the third and final phase of research which endeavoured to add clarity to the relative importance of REP components in contributing to other activities that occurred during planning and development.
4.8 Requirements Engineering Process Effects on Major Process Areas

The questionnaire administered in the final phase of investigation (Appendix B), concerned only the effects of the revised requirements process on other development process areas, and each area's constituent subprocesses. Findings from this questionnaire are organized first according to requirements process' effect on other processes, the most effective REP components, how REP components affected different process areas, and finally the relative importance of each REP component.

Before considering specific processes, respondents were asked to consider requirements process impact on other process areas as a whole. Chart 4.9 summarizes these responses, showing the average response of participants in each of the five main process areas. As the questionnaire allowed impact responses in the range between -3 to 3 inclusive, 3.0 represents the
maximum possible value (as in both Chart 4.9 and Chart 4.10).

![Bar Chart]

**Chart 4.9: Requirements Process Impact on Process Areas**

Similarly, Chart 4.10 shows average responses to individual processes grouped according to process area. For example, within the testing process area, testing against requirements received on average the most positive responses (1.85 out of a possible 3.0), indicating that this process area was positively impacted by the revised requirements process. In particular trends indicate high impact on project planning, testing, and to some degree, development. With the exception of the tracking process, SQA and SCM were not as highly impacted by the requirements process.

Furthermore, five processes are emphasized in Chart 4.10 to indicate that they had been highly impacted by the revised requirements process: sizing in project planning, tracking in SQA, conformance to specifications in development, and peer review and testing against requirements in testing. These processes and the REP components that affected them are dis-
Chapter 4. Case Study Findings

4.9 REP Component Impact

Findings presented in this section concern participants’ responses to the optional selection of a ‘predominant REP component’ for each of the development processes affected by the requirements process. By allowing respondents to make this selection it is possible to gain further insight into the nature of the requirements process’ effect on other processes.

Understanding the findings from the questionnaire responses takes some time due to the inherent layered relationship of responses. A respondent’s rating of requirements process effect (described and summarized in the previous section) seems inherently linked to their selection of a REP com-
ponent indicating its predominant role contributing to that effect. To illustrate how these two related sets of responses were synthesized, data showing REP component responses for the five most highly affected subprocesses is presented first. Subsequently, REP component responses over each process area is described.

4.9.1 Five highly affected processes

Component responses provide further evidence that shows which components of the requirements process contributed to particular development processes. To illustrate, this section considers in detail, the five most highly affected processes from Chart 4.10: sizing, tracking, conformance, peer review and testing against requirements.

Note that the traceability process is ignored in this examination because although it scores very highly, it is too closely related with the traceability component of the requirements process. In fact, traceability as a project planning process is almost completely satisfied by the traceability component of the revised requirements process. While this apparent correlation was considered during questionnaire design, traceability was included as a means to baseline the instrument. The relatively high score for traceability in Chart 4.10 (1.9) suggests that respondents recognized the close relationship between these two very similar processes.

Recall that respondents were instructed that a selection of an REP component would indicate that component as predominantly responsible for impact on the given process. Table 4.3, given below, shows detailed component responses for the five highly affected processes. Below each pro-
cess heading, the REP impact ‘score’ is indicated together with the number of responses contributing to this score ('resps'). The score column is a calculated value that encapsulates both the number of responses and the rating of requirements process impact each respondent chose for that component selection.

<table>
<thead>
<tr>
<th>Component</th>
<th>Sizing</th>
<th>Tracking</th>
<th>Conformance</th>
<th>Peer-Review</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decompos'n</td>
<td>17</td>
<td>0</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Traceability</td>
<td>0</td>
<td>9</td>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>Analysis Sess.</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X-Func. Teams</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Structur'd Rqmts.</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Test Scenarios</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.3: Requirements process impact responses by REP component for highest scoring sub-processes

The score value was calculated to capture the strong relationship between component identification and perceived impact. Therefore, component scores are essentially component responses weighted according to impact. For example, a respondent that selects decomposition and who perceives "3" as REP impact qualifies as a score of 3 in favour of decomposition for that particular process. In the same way, another respondent who makes the same selection but perceives "1" as REP impact records "1" in favour of decomposition for that particular process. All component responses, weighted in this manner, were added to produce a single score for each component in each process.

Examination of Table 4.3 clearly shows that participants believed fea-
ture decomposition had the greatest impact on improving 'sizing'. Likewise, traceability is shown to positively contribute to both tracking and peer-review processes. Under the development process area, conformance is perceived to be positively impacted evenly by both de-composition and traceability.

Table 4.3 serves to provide evidence that confirms some of the claimed benefits of the requirements process. Specifically, showing feature decomposition as contributing heavily to estimations during project planning, presumably due to the more thorough understanding necessary to accomplish decomposition. Likewise, project tracking during SQA activities and peer-reviewed testing, both benefited from artefact traceability established as a part of the revised requirements process. Finally, developing code that was conformant with specifications is perceived to be improved both by decomposition and traceability because both of these activities enable developers to access more detailed information about the requirements, making developing conformant code more straightforward.

4.9.2 Component Responses Across Process Areas

Although component responses are important among those processes which were most highly affected by the revised requirements process, it is informative to consider how REP components contributed to each major process area. Those results are summarized in Chart 4.11, and show how significantly each REP component contributed to each of the five major process areas at ACUS.

Chart 4.11 shows the cumulative REP component score (calculated as
Chart 4.11: Cumulative REP component scores by process area

described above) among each of the five major process areas. For example, in project planning, *decomposition* had an cumulative score of 34 across each of the seven individual processes that constitute project planning. The chart emphasizes important trends and corroborate evidence collected earlier in the study. In particular, that the requirements process is of significant importance to planning activities, that test scenarios are critical during testing activities and that traceability is an important component of the requirements process which finds application in most activities.
4.10 Relative Impact of REP Components

There is some similarity between the relative impact of REP components and a question from the intermediate questionnaire, Q4, which asks engineers to identify important components of the requirements process with respect to their work activities (see Chart 4.8). The similarity of these two questions naturally leads to their comparison.

Comparing both Chart 4.12 and earlier results suggest that respondents opinions appear to have shifted, emphasizing the technical outcomes of the requirements process: rating traceability, test requirements and decomposition very highly. This contrasts with the earlier results suggesting the overwhelming importance of cross-functional team involvement and group analysis sessions.

Chart 4.12: REP impact on processes organized by process area (average of responses)
While traceability and testing related activities still rate highly, it is surprising to see group analysis sessions and cross functional team involvement score so poorly in this most recent questionnaire. Perhaps this can be attributed to passage of time between questionnaires. It is possible that participants in the earlier questionnaire were aware of the important 'soft' products of the requirements process, while the tangible end results were more memorable in the minds of our phase 3 subjects.

Unfortunately the comparison between questions is not perfect. Whereas Chart 4.8 compares importance of particular components of the requirements practice, Chart 4.12 above, compares relative significance of requirements processes. The distinction is subtle, but essentially the former concerns impact on development activities (design, implementation, test, etc.) whereas the latter concerns impact on processes (estimation, inspection, etc.).

In addition, the individual requirements components differed between questions. While some categories have a clear unambiguous mapping there are some notable exceptions, for example test scenarios were clearly an outcome of the requirements practice (earlier question), however testing according to requirements is a directive demanded by the requirements process (latest question). Similarly, decomposition was included in this most recent questionnaire as it is clearly a goal of the requirements process, but we did not include it as a mean to achieve that goal when we asked about requirements practice during earlier (Chart 4.8).
4.11 Summary

This chapter described the evidence collected over the course of the research study at ACUS. It showed that compared to previous projects, ACUS has experienced many positive effects that are attributable to their revised requirements processes. They enjoyed far more accurate project resource estimations, had more effective communication among developers, reduced their rework, controlled feature creep and experienced few pre and post-deployment defects, all while maintaining customer satisfaction. Despite the outlay of time and effort spent conducting requirements activities, engineers' willingness to commit equal or more resource to requirements engineering in the future attest to its value. The findings also reveal that particular components of the requirements process affected different development activities in varying degrees. Furthermore, the evidence indicates that other development processes also benefited from the revised process.

In the following chapter a comprehensive analysis of these findings is presented which combines the observed project effects with the findings related to inter-process effects.
Chapter 5

Discussion

This discussion describes both the results and limitations of the research. It begins by synthesizing the results of the findings to holistically describe the effects of the requirements engineering on software development. To do this, a map of process interactions is introduced and described in Section 5.1. This map shows how the requirements engineering process not only directly contributed to the detected payoffs, but also indirectly contributed by producing beneficial effects on other planning and development processes. This discussion emphasizes the apparent interaction and interdependence of processes within the organization.

This chapter also includes a discussion of the applicability and limitations of this work. In particular, the limitations of the study are considered along two dimensions: potential alternative theories and inherent limitations of the way the study was implemented. Alternative theories are an important component of the case study research method, serving to raise potentially competing theories which might otherwise call into question the validity of the central finding (Yin, 1994). Such theories and their short-comings are explored below in Section 5.4.
5.1 Relationships Among Development Processes

The evidence presented and discussed so far provides a holistic picture of how requirements engineering affects software development projects. The payoffs of the new requirements process are described in Chapter 4, showing wide-ranging improvements from feature coverage, to communication, to estimations and requirements creep. In addition, evidence describing the effect the requirements process had on other development process has been presented in Sections 4.8-4.10. Combining all of these findings helps to finally answer the question of how requirements engineering relates to improved risk management, quality and productivity.

This following discussion explores this relationship by examining the interaction of processes, illustrated in Figure 5.1, and explains how requirements engineering ultimately produced the end-effects observed during the study. While the requirements process appears to have impacted the project directly, it also contributed significantly to the efficacy of other process which in turn contributed to overall project success. This interaction suggests that not only can requirements engineering process affect other development processes in different ways, but that the requirements process, in part, owes its success to those other development processes. In ascribing the successes of this project to the requirements process and acknowledging that the same process has improved the efficacy of other processes, then surely the former depends on the latter.

Figure 5.1 illustrates how the requirements engineering process relates
to other processes to produce effects in productivity: project communication and rework; quality: defects and feature coverage; and risk management: estimations, project negotiations and requirements creep.

In depicting this complex relationship, Figure 5.1 brings together the findings of the entire study. In particular, the payoffs of the requirements process on the project are summarized in Table 5.1 (described in detail in Sections 4.1 – 4.6) and the effects of the requirements process on other processes (Sections 4.7 – 4.10).
Figure 5.1: Bold arrows indicate the requirements process positive effect on other processes which contributed to project effects summarized in Table 5.1.

In the centre of Figure 5.1, the requirements process is shown connected to intermediate processes which in turn are connected to the observed payoffs shown as terminal endpoints around the periphery of the diagram. These connections indicate positive contributing effect. For ex-
ample, the requirements process improved testing against requirements which in turn led to improved feature coverage, a measurable attribute of the project.

The arrows originating from the centre node indicate a positive contribution by the requirements engineering process on other planning and development processes. The results that these arrows illustrate are derived from the responses on requirements process impact on other development processes (see Chart 4.10).

<table>
<thead>
<tr>
<th>Payoff</th>
<th>Detail</th>
<th>Trend</th>
<th>Evidence of impact</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Improved Risk</strong></td>
<td>Estimation</td>
<td>Up</td>
<td>50% improvement</td>
</tr>
<tr>
<td>Management</td>
<td>Feature Coverage</td>
<td>Up</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Requirements Creep</td>
<td>Dn</td>
<td>90% positive response</td>
</tr>
<tr>
<td></td>
<td>Project Negotiations</td>
<td>Up</td>
<td>100% pos. manager response</td>
</tr>
<tr>
<td><strong>Increased Quality</strong></td>
<td>Support Requests</td>
<td>Dn</td>
<td>45% fewer</td>
</tr>
<tr>
<td></td>
<td>Post-Deployment Defects</td>
<td>Dn</td>
<td>55% fewer</td>
</tr>
<tr>
<td><strong>Increased Productivity</strong></td>
<td>Superfluous Communication</td>
<td>Dn</td>
<td>85% positive response</td>
</tr>
<tr>
<td></td>
<td>Effective Communication</td>
<td>Up</td>
<td>57% positive response</td>
</tr>
<tr>
<td></td>
<td>Informed decisions by developers</td>
<td>Up</td>
<td>91% positive response</td>
</tr>
<tr>
<td></td>
<td>Rework</td>
<td>Dn</td>
<td>65% positive response</td>
</tr>
</tbody>
</table>

Table 5.1: Summary of observed requirements process payoffs during development

The sections that follow examine parts of Figure 5.1 in detail and show the exact impact, as rated by respondents, of the requirements process. They also show which particular requirements process component primarily contributed to that impact, directly derived from responses given during the final questionnaire, described in Section 3.6.3 (also as shown in
Table 4.3 for sizing, tracking, conformance and peer-review).

Arrows connecting intermediate processes to the observed effects on the periphery of the diagram complete the path from requirements process through intermediate processes and then to the final payoff. These arrows are based primarily on qualitative evidence collected during the intermediate questionnaire (see Section 3.6.2).

Qualitative evidence strongly implies that processes also contributed to the overall measurable project effects. This qualitative evidence is provided in each section below. The discussion is organized according to the effects observed in the three categories of (1) risk management: accurate estimations, effective project negotiations and reduced requirements creep, (2) quality: fewer defects and improved feature coverage, and (3) productivity: improved project communication and reduced rework. So while the quantitative evidence collected in the last questionnaire of the study justifies the connection of the requirements process to intermediate processes, qualitative evidence from the intermediate questionnaire justifies connecting intermediate processes and the end effects.

Below, each portion of the diagram relating to risk management, quality and productivity are explained separately. The chain of reasoning justifying each labelled arrow in the diagrams is provided.
5.1.1 Risk Management

Effect: More Accurate Estimations

This study presents a clear indication of the role of requirements practice in improving estimation in software development and project management. At ACUS, the estimation process required team leaders, in consultation with their engineers, to make effort estimates for particular features. The overwhelming response from those involved in the estimations suggests improvement in estimation due to improved requirements practice. The fact that the post-analysis estimates were much closer to the actual efforts strongly indicate that the requirements analysis sessions that generated technical requirements were influential in enabling more accurate estimations.

Although pre-analysis estimates are expectedly inaccurate, due to the ambiguity of 'one-statement' feature requests, one might expect those estimates to be optimistic, due to easily overlooked technical details, rather than the decidedly pessimistic estimations made at ACUS. This is likely because ACUS had, in past projects, over-committed to features causing severe schedule overruns. In addition, pre-analysis estimates included a component of rework, which to some extent did not eventuate, given the improvements resulting from earlier definition of the work scope. This added to the volatility of the pre-analysis estimates, given that pre-analysis estimates were "best guesses" without the benefit of a clear understanding of the requirements.

In summary, ACUS' estimation ability was enhanced largely by their
ability to use the more detailed information on features analyzed as part of the revised requirements process. Improved estimates enabled early identification of resource constraints, which motivated the early revision of project commitments to accommodate these constraints. As the project manager noted, “the revised requirements practice enabled us to conduct better estimates and project planning and as a result, cull functionality for which we did not have adequate resources. In prior releases, we would have attempted to deliver more functionality as we would have had a poorer understanding of the requirements and their scope.”

Effect: Reduced Requirements Creep

ACUS’ change management process was instrumental in preventing unconstrained requirements creep. Compared to past projects which had exhibited explosive feature creep that crippled project progress, the evidence strongly suggests that the rigorous approval process reigned in change. Much of this success can clearly be attributed to the software change control board, whose mandate it was to approve only the most necessary changes. Their stewardship of the change process assured its success as a credible defence against change.

Through the definition and enforcement of a strict change management protocol, engineers were dissuaded from making discretionary changes. The new process put the onus of initiating change on those seeking it. For developers, this additional cost likely convinced many to disregard ad hoc changes altogether. This is evident from one engineer’s assessment of the process: “[it] had a big impact: it made people analyze and think twice
about the changes they were considering.”

The important changes, the ones formalized in change requests, were rigorously scrutinized for their potential impact on the project by the software change control board. The board was comprised of both cross-functional representatives (designers, developers, testers and documenters) and also of a cross-section of organizational responsibility (engineers, team-leads, technical managers and executive management). Essentially each change request provided a precisely focused forum in which developers and decision makers could openly communicate. This provided a unique opportunity for the individuals to exchange and disseminate their specific concerns among the team. In the face of requests for change, ACUS senior management no longer had to make decisions without thorough information on the impact of change. The involvement of such a variety of functional roles provided for more informed decisions and an increased ability to reject changes that could not be accommodated.

It is very likely that cross-functional communication and involvement, evident in change artefacts, contributed significantly to their more holistic assessment. The extent to which development artefacts were referred to in change requests also suggests that traceability is an important aspect of the change management process. The fact that 50 modifications to the requirements specification were made as a result of 77 change requests (the majority during design and coding) suggests that engineers were by and large successfully able to leverage traceability links back to requirements, thus facilitating more accurate impact analysis. Only 4 modifications to the requirements spec were initiated during integration testing (as a result
of 7 change requests), suggesting that most of the requirements-related changes were addressed before integration testing. Assuming that integration testing exhibited similar rigour as testing in earlier phases, this is a very positive result.

Improved requirements change management at ACUS provided project members and senior project stakeholders enhanced ability to negotiate any requests for changes and thus to prevent feature creep throughout the development life cycle.

**Process interactions leading to improved risk management**

Figure 5.2: Detailed requirements engineering process interactions leading to accurate estimations, effective project negotiation and reduced requirements creep
Effective Negotiations

Effective negotiation was made possible by the requirements engineering's decomposition of features critical to making sizing estimations, which subsequently enabled ACUS to effectively assess resource requirements. Structured JAD-like meetings held at regular, pre-specified intervals throughout the requirements phase lent credibility and significance to negotiations. This confirms the role of JAD-like meetings as an effective strategy in requirements engineering process improvement (Jones, 1996). Structured requirements documents aided the change management process which by its very existence threatened to restrict late-changes, which in turn motivated stakeholders to align their expectations.

Arrow XII

The systematic analysis of requirements led to developers' improved comprehension of features during sizing (REP impact of 1.8 on sizing as shown in Figure 4.10, 'decomposition' from Figure 4.11). That subsequently enabled the project team to make more accurate effort estimations. Those estimations provided critical input to negotiations, and aligned ACUS' development capacity with the market priorities. One team-lead commented that "having requirements done early, it became obvious we could not deliver all of the expected functionality, so we agreed to cut them. Previously we would not have known until it was too late and then everyone would have to go into a mad rush."

Arrow X

It was understood by both stakeholders at the outset of the project that the revised requirements process would prevent major features changes
after the negotiated requirement set had been committed to. The threat of limited change motivated ACUS to seek detailed information before commitment, and motivated marketing to prioritize their needs in light of ACUS’ finite capabilities. This new understanding led to “more effective expectation management” or as one manager put it, “made it much easier to get everyone singing from the same sheet”. The change control processes empowered both stakeholders by permitting the marketing unit to comment on change proposals.

More Accurate Estimations

Accurate project estimations were primarily the result of accurate feature effort sizings conducted by engineers who could make extensive use of technical requirements. Interestingly, ACUS achieved this improvement without the use of elaborate estimation methods, such as function-points as proposed in (Jones, 1996). By bringing clarity and focus to its requirements ACUS was able to leverage its engineers’ extensive technical knowledge and experience of the product to produce far more accurate project estimations. Again, the change management process prevented unexpected changes and helped to ensure original estimations stayed on target.

Arrow XIII

Individual technical requirements, estimated by the engineers assigned to analyze and later implement them, were aggregated to construct feature estimations and ultimately guide project planning and project estimations. The revised requirements process facilitated a thorough understanding of requirements to aid feature sizing; as one team-lead put it “time spent
during requirements analysis to get finer detail made it possible to have better estimates earlier on.”

**Arrow XI**

Once schedules and resource allocations were established, careful control of change requests prevented project corrections while ensuring the currency of the original estimations. As one manager said, estimations remained on-target via change management because “the process reduced the number of changes sneaking into the product.”

**Reduced Requirements Creep**

Change management was instrumental in preventing unconstrained requirements creep. The success of ACUS1’s software change control board, an integral part of this achievement, confirms the board’s role as a means to control software change (Robertson & Robertson, 2004). Merely by virtue of implementing a formal change management process, engineers were dissuaded from making discretionary changes. Traceability established within the requirement specification which were used by project tracking to monitor resources helped to prevent creep from significantly affecting progress.

**Arrow IX**

The structured nature of requirements artefacts (REP impact of 1.3 on change management from Chart 4.10), helped enable change management, ultimately giving project managers control of requirements creep. Requirements churn that had been so common in past projects was controlled by relying on a rigorous requirements change process that limited all but the most critical changes. One engineer reported that the approval process
itself was significant: “[it] had a big impact: it made people analyze and think twice about the changes they were considering.”

**Arrow VIII**

When changes were necessary, change requests were considered in the context of project progress. Respondents indicated that improvements in project tracking that had occurred because of the revised requirements process enabled “identification of schedule risks”, making it “easier to forecast resource crunches.” (REP impact of 1.6 on project tracking from Figure 4.10, ‘traceability’ from Figure 4.11). Managers said they could effectively assess change requests and that the management process provided “firm control and visibility.”
5.1.2 Quality

Process interactions leading to improved quality

![Diagram showing process interactions](image)

Figure 5.3: Detailed requirements process interactions leading to fewer defects and better feature coverage

Improved Feature Coverage

Although testing prevents defects from reaching customers, it also plays an important role in assuring that the product contains promised features. Many questionnaire responses and comments recorded during interviews suggest that testing was significantly improved due to the test-scenarios conceived during requirements analysis. For example, one tester indicated that the requirements process had helped by “creating a starting point for us as to what to test, there was more of a focus on the actual requirements and not just the feature.” Likewise, another tester specifically identified
the requirement process' role in improving their testing: "test scenarios provided a good base to start writing test cases with".

Another indicated requirement test scenarios were used to validate specific test cases: "we could review the test case to see whether it has met the requirements and what we should be testing." Although these comments indicate a direct impact by the requirements process, some believed other processes were, at least in part, responsible for the apparent payoffs. Respondents also cited the peer-review of test cases and the emphasis of testing features according to requirements as factors contributing to assuring feature coverage.

Arrow I

Testing conducted by ACUS toward the end of the project not only served to detect defects before deployment, but also to validate that the baselined features had been implemented in the software product. The anticipation of such testing likely encouraged developers to assure feature coverage throughout the development project. Developers commented that "the use of rationales and test scenarios gave a broader conceptual view of what was required and how we were to demonstrate that we had met the requirement".

Arrow II

Peer-review inspections of test cases contributed to testers' ability to test coverage. Testing was successful in part because "reviews [were] done with designers and test case authors to validate the 'how' of each test case". The outcome of these reviews were potentially enlightening to both parties, and according to respondents these reviews were successful
in revealing issues: "[during] reviews, test specs came under [scrutiny], the resulting issues were sometimes abundant".

**Fewer Defects**

In the context of the ACUS project, defect rates appeared to have changed significantly under their revised processes. Both internal, pre-deployment defects (system test bugs) and post-deployment defects showed marked decreases; nearly 50% fewer post-deployment defects (see Section 4.4 and 4.6.

One engineer, after being informed there had been fewer internal defects appeared genuinely surprised. This engineer’s explanation for his surprise illustrates the long-term ramifications of more accurate estimations/scheduling and hints at the wide-ranging holistic effects of process change: “Wow - I’m impressed if that is the case as there has been much better testing. Process changes have made engineers more aware of how their piece fits in with the rest. I also feel that the schedules were more realistic so there was no temptation to ‘bang it in’ and move on to the next task – each change was more rigorously unit tested before being integrated.” However, one respondent did indicate that defect entry was made more difficult under the new process, requiring engineers to “... fill in several documents.""", then went on to say that “people are reluctant to enter defects, knowing it will only add to their workload”. Notwithstanding this observation, post-deployment defect rates and customer satisfaction suggest that product quality had increased.

**Arrow III**

Orienting the test effort by emphasizing testing according to require-
ments specifications improved the efficacy of testing. This notion is directly supported by many respondents such as this engineer explaining why the requirements process had improved quality: “targeting the requirements and testing them too [caused] bugs to be detected earlier.”

Arrow IV

In the same way that peer-review improved testing efficacy assuring feature coverage (see description for arrow II), more effective testing also improved defect statistics. When respondents were asked why defects in this project were lower compared to projects in the past, participants suggested that the project had benefitted from “improved peer review”, and due to “the review process and testing inspections related to requirements, which were done as part of the revised requirements process”.

5.1.3 Productivity

Effect: Improved Productivity

Evidence indicates that the revised requirements practice had a significant positive effect on later stages of development. The requirements artefacts produced early in the project were not only important as a developer resource. The analysis and production of requirements itself provided many apparently positive collateral effects. Compared to previous projects, productivity increased: rework lessened, communication effectiveness increased and capacity for informed decision making improved.

Productivity had been increased, in part, by helping engineers as one put it: “[to] focus on individual work items”. The cross-functional involve-
ment during analysis sessions achieved more thorough up-front thinking and "mutually agreed understanding of what was required to be done", and further served as an "ice-breaker" such that developers were more willing to discuss issues with their counterparts during later stages of development: "This improved coding efficiency many-fold".

The increased effectiveness of the communication can be considered from two perspectives. On the one hand, the findings indicate that the improved understanding of details, dependencies and complexities of features significantly decreased the volume of "clarifications". Such clarifications could be considered wasteful or redundant if information is primarily disseminated in such one to one interactions. In contrast, requirements served as a means of low-cost, broad-based knowledge dissemination. In fact, the better-defined and structured requirements specification functioned as a resource to propagate information about the features and their inter-dependencies. As one engineer remarked, "the requirements specification was an invaluable reference to keep track of what needed to be done and to resolve any misconceptions down the track". Project members used requirements as a persistent resource in their daily activities, as a "solid foundation" to seek information on requirements and on which they based their designs and programming decisions on, avoiding unnecessary and redundant communication. It seems likely that informed developer decision making at the individual level would lead to fewer defects, less time spent in fixing bugs and ultimately result in higher developer productivity.

On the other hand, the requirements specification could also serve to
increase the quality of developer communication. The requirements specification provided engineers with a project specific vocabulary and a consistent basis from which to seek further information. Furthermore, cross-functional group interactions that had been established during group analysis sessions set a positive precedent for subsequent communication later in the project. Different departments were brought together to communicate collectively on the particular requirement, providing an opportunity for engineers to understand the responsibilities and concerns of other functional roles in the larger development team. One team lead reported that the process made him "aware of the impact on other areas and think more about the total package rather than in 'tunnel-vision' mode."
Process interactions leading to improved productivity

Reduced Rework

The decomposition of features into detailed requirements produced reliable specifications on which engineers could rely and conform to. In the past, confusing unreliable specifications led to extensive rework when engineer interpretations diverged from one another. When asked how rework was reduced under the revised requirements process, respondents provided evidence to suggest that improved specifications and cross-team communication both contributed.

Arrow V

Basing technical specifications, such as designs or test cases, on accu-
rate requirements specifications provided consistent and informative direction for engineers. Clarifying claims of reduced rework on engineer attributed "better designs ... design does get rid of stupidities". Another said that "less functionality was missed due to constant reinforcement of requirements in [specifications]." In effect, saving the team from having to refit existing development to add missed features. One manager noted that having fewer features dropped after development had started, resulted in less waste.

**Arrow VI**

The organization of teams into cross functional units contributed to cross-pollination of ideas and information. On the rework issue, one manager indicated that cross-functional teams provided "more chances to expose people across the product to everything. More eyes on design and code." In clarifying his position on reduced rework one manager indicated that the "involvement across teams had positive effects. In the past we worked in silos."

**Improved Communication**

Many engineers praised the apparent improvement in communication during this project compared to the past. In many cases they attributed these effects directly to the revised requirements process. For example, one engineer noted that the process had helped him by "reducing meetings and promoting good communication to other teams", another noted that they "felt freer to talk to others". Managers felt that the discussions initiated during requirements analysis sessions set a good precedent for open communication.
Arrow VII

Not only did requirements provide effective common ground for project communication, they also enabled the reorganization of development into cross functional teams. During interviews and in questionnaires, many respondents attributed this change as "reducing meeting times and [contributing to] good communication to other teams" and "providing consistency across functional groups".

5.2 Early Engineer Response

Early indications from data collected at the beginning of the study (described in detail in (Damian et al., 2002)) suggested that while ACUS faced serious challenges, many engineers were somewhat skeptical that the revised requirements process would produce the effects advertised. While observing and interviewing engineers, many respondents voiced their skepticism about the new process. Many thought that the new process emphasized inevitably slow, time-consuming and ultimately ineffective group analysis and negotiation meetings. Developers were concerned about the added overhead of the new requirements process.

Notwithstanding their early doubt, after the requirements phase had progressed and many engineers had participated in requirements analysis sessions, opinions appeared to have shifted. On a subsequent survey, engineers responded more favourably to the revised process. This shift in opinion can most simply be attributable to engineers recognizing the value of the requirements sessions they had participated in. Resistance to
the new process and the change it brings by those it affects is one of the more considerable obstacles to process adoption and software process improvement (Moitra, 1998). This findings suggests that the process quickly began to win the confidence of engineers within the organization, despite their predictable initial cynicism.

5.3 Applicability

The data collected during this research has been used to elucidate our understanding of how requirements activities affect software development. However, the elucidation is not complete. This investigation suffers from the limitation of time, resources and opportunity, in that the collected evidence originates from a single organization. Despite these limitations the research represents one step towards bridging the gap between theory and practice.

First, consider that ACUS does not represent a particularly remarkable software development organization according to characteristics that are commonly used to delineate development organizations. ACUS employs approximately 150 staff, arranged in a fairly flat hierarchy that include small teams led by a team lead and technical managers, all headed by a project manager and a product manager. The relationship ACUS maintains with its parent company's marketing department who function as customers is not unusual. This customer relationship is similar to organizations that receive requirements and funding from separate or internal corporate entities, as well as organizations that build custom solutions for
a single customer. ACUS's software itself forms a product line related to payroll accounting and financial functions, is designed to be customized by resellers or customers and has a well established 20 year history. All and all, these attributes are very characteristic of many other software development organizations.

Second, consider the nature of the improvements that have benefited ACUS, and whether other organizations would benefit from similar progress. As previously described, ACUS suffered from dysfunctional customer negotiations, teams within the organization who operated autonomously and independently of each other, and requirements that were routinely revised without regard to their impact on development. Many companies that would be assessed at CMM level 1 would, by definition, be enduring similar woes. In other words, the evidence collected from ACUS is clearly relevant to organizations who are burdened by similar concerns and wish to improve their internal communication, customer negotiations, change control, and quality control in testing and inspections. Although these are noble accomplishments it would be unwise to expect the same results with all companies.

So while there are many companies that share common characteristics with ACUS, there are also those for whom many of the insights shown here would not apply. Compared to organizations that already exhibit process maturity, ACUS is unique in that it can leverage the steepest, most rewarding region of a process' diminishing returns. In a more process mature company, the addition of such requirements process would have more ambiguous outcomes, particularly improved communication – an
effect that may be realizable using other approaches.

Similarly, companies that are considerably smaller may not ail from internal communication issues merely by virtue of their compact size. This may in part explain why rigorous requirements processes do not appear to produce beneficial effects as readily in small organizations, as in large ones. For example, Emam & Birk (2000) provides evidence that requirements engineering process maturity correlates with an organization's software quality, but this relationship is only true for medium to large organizations.

Finally, requirements management and control comes at a price, costing added overhead in considering change requests and dampening the software's rate of change. This is a desirable outcome for an established software product where features can be included in one version or another. But, for a from-scratch speculative software project, such demands may suffocate the development necessary to produce an innovative new solution.

In summary, these results are likely very applicable to industrial developers who share many of the same organizational and software characteristics, and who also suffer from some of the same problems that ACUS does. Of course, it may appear self-evident that the solution to the problem is generalizable to others with the same problem, the challenge is identifying the problem. Companies may not be fully aware of their own dysfunction until they try different approaches and carefully reflect upon their own development practices.
5.4 Limitations

No research, especially research such as this which endeavours to explain complex human phenomenon can hope to escape the shadow of doubt and the inevitable limitations of such work. The limitations of this research fall under two broad categories: alternative theories and fundamental shortcomings in the research methodology.

Yin (1994) recommends that case studies consider alternative theories that could provide competing explanations for the phenomenon under study. Exploring such theories is not only academically honest, but can enumerate why such theories are unsuitable, thus bolstering support for the main thesis.

Many limitations of the study can lay blame on the way the study has been conducted in the first place. These methodological shortcomings are discussed not only to admit their threat to the validity of the thesis but to highlight where improvements are needed to advance methodological theory and practice.

5.4.1 Alternative Theories

Given my own analysis of feedback received from existing and pending publications on this work, there are four alternative theories that skeptics may argue. First, it is possible that new management at ACUS, a change which occurred just prior to the software process improvement, was in fact responsible for the bulk of changes at ACUS, rather than any particular process. A second possibility, implied in the comments of some engi-
neers, is that evident payoffs were due to the product's maturity, rather than improvements in process. Third, it is possible that other process changes, unrelated to requirements or change management could have been accountable for the positive changes. And fourth, the combination of the Hawthorne or placebo effect may have significantly affected engineer opinion.

**New Management**

There is no question that the change in management at ACUS, namely of their project and product manager at the beginning of the project before the software process improvement initiative had started, was an important factor of success at ACUS. Strong management support and the credible promise of change that comes with new management was most likely instrumental in ensuring that the new requirements process was adopted by engineers. But, although new management helped to assure process adoption, there is no reason to believe these new managers could assure process efficacy. In fact, this is evident by the difficulties ACUS encountered during their subsequent project in which they changed their processes again with a result that was far less successful.

**Product Maturity**

The questioning of engineers occasionally elicited obviously skeptical responses. Of these responses, many identified product maturity as playing a major role in producing the payoffs in question. For example, there were suggestions that this project had fewer defects than previous projects
because during past projects inherent defects in the underlying architecture had been addressed. However, this project was the third point release of their product since its last major revision. Selected managers were specifically asked to compare the significance of this project compared to previous projects and by all accounts this project represented a comparable degree of feature additions and complexity. The consistency in defect rates between the first and second point releases, indicating limited variability between at least the first two point releases suggesting relatively constant defect insertion and removal rates. To accept this theory, however, would require dismissing all other engineer responses that strongly attribute much of ACUS' success to the revised requirements process. In the end, it is impossible to completely discount the effect of code maturity without significant analysis of code changes between previous versions.

Other Processes

Although requirements management had been identified as the most deficient of ACUS' process area, ACUS did make revisions to some of its other processes. However, an inspection of the process documents suggests that these changes pale in comparison to the far more sweeping changes made related to requirements engineering. Some changes only required documentation of what had formerly been without record, for example, many project management processes were altered to require explicit status documentation and report production. In other cases, many relatively minor process changes were made to support the revised requirements process. For example, the formulation of the software change control board
Chapter 5. Discussion

was made a sub-process of the SCM process area, however the board’s purpose was to officiate change requests made necessary by the revised requirements process.

Hawthorn or Placebo Effect

Finally, many interview and questionnaires questions provided ample opportunity for engineers to identify or praise other processes. Although some respondents remained skeptical of the requirements process, no other process or factor had been identified as a more-likely contributor to the until-then-unusual successes of the ACUS project. This is not to suggest that ACUS owes its success to requirements engineering alone, only that the revised requirements process appears to have been the greatest difference between this project and past projects. And, furthermore, that qualitative evidence from respondents does support that the requirements process was a factor of success.

Some critics may suggest that the measurable quantitative effects in estimations, defect count, customer satisfaction or support rates are all subject to the Hawthorne effect (Mayo, 1933). They may go on to contend that qualitative responses from engineers and managers were all subject to the self-fulfilling nature of placebo-like effects. In such case, respondents would attribute success to the most obvious treatment, in this case the revised requirements process.

While it is impossible to fully discount the impact these effects could potentially have, early skepticism among developers suggest otherwise. During the initial phase of research, many respondents voiced their pes-
simistic views of the process revisions. Many engineers felt that the process emphasized slow, time-consuming and ineffective group analysis sessions. Developers were concerned that the increased overhead of the requirements process would put new demands on their limited time. Early doubt seems to suggest that respondents were not overly eager to praise the revised requirements process.

The possibility of other explanations is naturally a concern when considering the validity of research such as this. Although this single study does not definitively prove anything, the preponderance of evidence strongly suggests that requirements engineering has a strong positive effect on risk management, productivity and quality. Ultimately, more further studies such as this one are required to more completely and confidently understand the role of requirements engineering in software development.

5.4.2 Shortcomings in Methodology

Since this was the first time that a requirements process had been rigorously defined at ACUS, historical project data was very limited. This necessitated a reliance on the extensive working experience of ACUS engineers and managers to assess the impact of the requirements engineering process improvements and to provide comparison to previous practice. Our dependence on respondents’ perception and recollection is undeniable, although unavoidable. However, this dependence is not without merit as respondent opinion is a reflection of the satisfaction of process implementation.

Unfortunately, studies such as this one which attempt to understand
the effects of changes in software processes, suffer from the lack of proven instruments for gauging such interactions. In the absence of well-defined instruments we designed our investigation according to the particular case at ACUS. Multiple data collection methods and involvement of a broad range of respondents across the different functional groups at ACUS helped to ensure capture of a comprehensive range of opinion and insight.

5.5 Summary

This chapter has presented the most significant contribution of this thesis by illustrating how the requirements process can affect software development. Considering both the direct payoffs of requirements engineering and the interaction of requirements process and other processes, produces a holistic big-picture perspective of software development and software process. From this vantage point the interaction and interdependence between development processes is evident.

In addition, limitations of the study have been explained, including potentially competing theories and inherent limits of the implementation of this research. Although it is possible that new management, code maturity and other process changes could explain the observed effects, that likelihood is limited. Undoubtedly, this work is limited by the shortcomings of the instruments used to collect and analyze evidence. The development of improved instruments and dissemination of experiences through the publication of these kinds of studies is necessary to advance subsequent inquiries.
Despite the limitations of this work, it still represents significant research inspiring new questions and contributing to our current understanding of the effects of requirements engineering on software development.
Chapter 6

Conclusions

This chapter concludes the thesis by first presenting a few notable insights which stem from my own unique interpretation of the ACUS experience. These ideas offer up potential new directions for requirements researchers to pursue and investigate. This chapter also reviews the research with respect to the original research questions set out in Section 3.1 at the outset of the thesis. Section 6.4 considers specific ways that this research can inform industrial practice. Finally, the chapter concludes with a review of major contributions and concluding remarks.

6.1 Insights

The research presented so far gives rise to a number of insights about the role of requirements engineering. This section argues that communication, information dissemination and collaboration were critically important contributions of ACUS' new requirements process. Moreover, the nature of these contributions suggests that the tangible requirements artefact should not necessarily be considered the primary product of the requirements process. And finally, that components of the requirements process could be prioritized according to their apparent effects.
Each subsection below raises interesting observations leading to ideas that merit the attention of future research. It is apparent that although conducting this research has produced evidence in support of requirements engineering, it also raises fundamental questions about its role in software development organizations.

6.1.1 Communication and Collaboration

It became apparent, after conducting the intermediate questionnaire and analyzing responses from engineers, that one constant pervaded the progress made at ACUS. Communication and collaboration emerged as a constant theme in many engineer responses. It became apparent that many engineers felt the group analysis sessions helped break down barriers between functional teams at ACUS. These requirements analysis sessions were attended by representatives from all functional areas including design, implementation, test and documentation. For ACUS, the revised requirements process' use of group analysis sessions provided the impetus for brokering cross-functional (or cross-departmental) communication. Given comments made by managers and engineers 'breaking the ice' early in development helped to foster continued interaction between departments later in the project. Furthermore, by engaging functional teams during requirements analysis an understanding of requirements was established, providing a basis for subsequent dialogue.

This early interaction cannot be overlooked. It suggests that requirements engineering, one of the first development activities that should occur in a project, has a role to play in orienting the development team and
fostering communication and cooperation.

The importance of communication even transcends organizational boundaries. ACUS' ability to manage expectations is largely an exercise in effective customer communication, in this case with their remote marketing unit. This communication not only occurred during project negotiations but also through the change management process, which kept the marketing unit abreast of project deviations. Clearly, requirements engineering can help enable effective customer-client communication to keep stakeholders informed of important project events.

Communication both internal and external from the organization signify how important collaboration is during software development, both among software developers and stakeholders. Further research must examine the question of how such collaboration can most efficiently be fostered during software development projects. This is especially true when collaboration might otherwise be impaired by distance where stakeholders are physically separated: an issue that seems particularly germane given recent trends in outsourcing. Finally, the role of requirements engineering in this process must be considered. For example, is it necessary to involve the maximum of interaction as early in the project as possible (i.e. during requirements analysis) or can collaborative patterns be established sometime after initial requirements activities have concluded?

6.1.2 Room for Agility

When speculating about the essential elements of effective requirements engineering, the importance of the requirements artefact could be ques-
As discussed in the previous subsection, one of the insights of this study has been the beneficial effects of requirements engineering on communication. Although requirements were articulated in a written document at ACUS, which seemed to provide some value to engineers as a reference, it is possible that the process of analyzing and establishing a requirement is at least as important as writing it down. In other words, the exercise of analyzing requirements is still very useful, even if the tangible artefacts it produces is subsequently ignored.

The possibility of achieving requirements engineering benefits without its artefact could perhaps explain the surprising success of the Agile movement despite its irreverence toward formal documentation. Although it may be debatable whether the success of the Agile movement correlates with the performance of the Agile methodology, there are certainly some empirical claims that aspects of the Agile method can be successful (Williams, Kessler, Cunningham & Jeffries, 2000). Much to the chagrin of proponents of document heavy processes, Agile approaches seem to be successful without the burden of extensive document production. Although Extreme Programming (Beck & Andres, 2004) is but one of the approaches that flies the Agile banner, it depends on customer contact to produce ‘user stories’, which to some, may seem like a somewhat dubious replacement for a requirements specification. But, if this practice, or others specified by the methodology (i.e. pair programming), promote effective communication with customers and collaborative cooperation within the organization, then omitting the development of the specification no longer seems so absurd.
Certainly the correct approach for any organization must depend heavily on the task, but this observation suggests that it becomes necessary to quantify the value in establishing, and then maintaining, a requirements specification. The value of creating the formal requirements artefact during analysis should be investigated.

In ACUS's case, the evidence certainly appears to support the development of the artefact to enable subsequent reference by developers. However, it may be possible to provide this function through alternative (cheaper) methods whether it be informal spontaneous verbal communication or a mechanism that organizes the same information by alternative means.

The utility of the requirements artefact and its subsequent maintenance is an empirical question that deserves more study, but, it is clear that the value of requirements engineering is more than just the specification of requirements. This suggests, at least in some circumstances, that there may be room for optimization of 'traditional' requirements practices.

6.2 Implications for Practitioners

Considering feedback from the initial and intermediate phases of this research, it becomes apparent that some requirements engineering components have a greater effect than others. For organizations of similar requirements engineering maturity as ACUS, this is a valuable result that can be used to prioritize the adoption of requirements components. For research-oriented academics this insight provides impetus for further research into requirements engineering phenomena.
Industrial practitioners face two opposing options when adopting a requirements engineering process for their next project: too little requirements engineering, or too much requirements engineering. Such practitioners are either hoping to enjoy the fruits of requirements engineering by adopting more requirements practices, or they are optimizing their production costs by reducing non-essential spending on such activities that do not directly contribute to the final product. These are both laudable goals that have merit in the right circumstances. The prioritization of requirements components helps to establish which components companies should implement or retire first, depending on their objectives.

Based on the evidence collected in this study, practitioners that are establishing a requirements engineering process in their organization should do so by first adopting group analysis sessions, which for ACUS, proved to be very effective. This effort should coincide with a formalized management protocol for controlling requirements change through a centralized review committee. I believe that these two efforts will move an otherwise undisciplined development project a long way towards more reliable practices. By conducting group analysis sessions that are attended by representatives from various functional areas, collaborative barriers are broken down while a collective understanding of requirements is established. The requirements management process solidifies that collective understanding by limiting subsequent changes. Even when changes are necessary, they are then visible to all of those involved in the review process. Establishing and maintaining requirements requires transparency and openness to ensure that requirements are a foundation that developers can depend on.
From ACUS' experience it is apparent that some requirements components were less successful than others. Namely the structured requirements document, the sentence template and the context diagram (used to structure requirements specifications) did not receive extensive acclaim from engineers. Therefore, I speculate that practitioners should consider these components only after more effective components have been successfully implemented. Unfortunately, this is only speculation. Although the evidence does not definitively lead to the conclusion that these components are unnecessary or subordinate compared to others, respondents did appear mixed in their assessment. For projects such as this one, where developers are experienced in developing such software, it is understandable that significantly constraining the style of documentation might have limited utility. However, it is also possible that respondents were simply not aware of how useful these restrictions were. Further experimentation and study is necessary to add clarity to this issue.

Providing further insight into the independence and interdependence of the components of requirements engineering and other development processes should be the concern of further academic inquiry. If anything, this study illustrates that there are differences between the components that constitute a requirements engineering process, that their efficacy likely depends on the maturity and needs of the organization using them. The relationship between requirements engineering components and their effectiveness need to be elucidated to inform tool and methodology design. Such understanding may be critical in improving both the efficiency, effectiveness and adoption of requirements engineering practices in industry.
6.3 Reviewing the Research

Having presented the findings, building a model of process interactions, and describing the implications of these findings, it is appropriate to review how the study's original research questions have been addressed:

**How does requirements practice impact the early stages of development?**

According to these findings, requirements engineering clearly has a role in improving negotiations by providing the impetus for stakeholders to agree on acceptable requirements. During stakeholder negotiation and requirements analysis, vital requirements information can be elicited and absorbed to inform early planning activities and subsequent development. It has been shown that using requirements early can dramatically improve project effort estimations and schedules. Estimations feed back into negotiations customer needs can be prioritized, serving align stakeholder expectations and assure project feasibility. Together these improvements all help to address and control project risk.

**How does requirements practice affect downstream development?**

This study clearly demonstrates how important requirements change management can be in controlling change and preventing uncontrollable feature creep. Early conception of test scenarios, as a component of requirements practice, can be effectively used by testers near the end of the project to ensure quality and validate that the requirements have been satisfied.
Analysis of requirements by cross-functional teams can bring about collaborative communication patterns among developers, promoting effective communication and common understanding. Requirements artefacts themselves function as a foundation for communication among developers and a medium through which to deliver technical information. These improvements appear to produce highly [customer] satisfying software of higher quality with less wasted development rework.

How could the interaction between the requirements engineering process and other processes have contributed to the effects that had been observed during the previous phases of the study?

The specific details of process interaction presented in this study not only corroborate the findings and conclusions described above, but also provide additional unique insights. The importance of traceability in requirements is clearly evident, particularly for project planning activities, project tracking and to testers. Furthermore, it is shown how requirements engineering can be useful to testing processes which validates product functionality, and how requirements can be used positively to facilitate peer-review. The interaction between requirements engineering practices and other processes suggests that requirements engineering can have project-wide effects and may depend heavily on the particular nature of other processes currently in use.
6.4 Recommendations for Practitioners

It is encouraging how much progress ACUS was able to achieve despite making relatively inexpensive, straight-forward changes to their process. ACUS did not use any elaborate requirements methodologies, but depended on structured negotiations, engineer estimations and a well-defined change management procedure. These changes alone addressed many of issues.

Although it is doubtful that any one method can be universally applied, organizations that find themselves in similar circumstances may find success by adopting strategies that worked for ACUS. In particular, organizations who have extensive technical knowledge of their product and receive features from remote proxy customers (marketing unit or otherwise), but suffer from similar requirements process maturity may benefit the most.

These factors, and strong management support, made ACUS an organization ripe for change. Perhaps the real challenge is identifying these problem areas. Organizations may not be fully aware of their own dysfunction until they try different approaches and carefully reflect upon their own development practices.

The strategies described below address improving risk management, quality and productivity. Addressing risk management requires establishing a foundation for customer-supplier cooperation to help promote mutual expectations while encouraging the development and the validation of accurate requirements. Quality can be improved by considering validation of requirements and feature testing early during the require-
ments engineering. Establishing analysis sessions which are attended by members of every functional area can help foster cooperation and efficient communication.

**Iterative Feature Decomposition:** Committing to high-level features from customers without input from development organizations is likely an invitation for unrestrained feature creep and missed deadlines. Instead, stakeholders should understand that provision of high-level features is only the beginning. A process should promote systematic feature analysis and decomposition by a development team who can then seek clarifications from and provide effort estimations to marketing. This is an iterative processes that empowers stakeholders to reach an agreeable outcome despite their geographical separation.

**Frequent Negotiation Sessions:** Understanding the needs of the customer and the limits of the supplier are prerequisites for reaching an agreement. A requirements process that facilitates frequent negotiations among remote stakeholders is an effective way to overcome mistrust and power-struggles. Rigorous effort estimations can empower the development organization to inform requirement prioritization to align development capability with corporate strategy and market needs.

**Change Management:** Change is inevitable, but a process can assure that change affects the project in a controlled manner. A process can motivate stakeholders to get appropriate requirements established by specifying an approval process which takes effect when project scope is agreed to and baselined. Requiring stakeholder approval, irrespective of location, ensures that all parties will remain fully informed of requirements change.
Early Test Scenarios: Test scenarios are necessary for conducting system testing and requirements validation over the development period. Conceiving of test scenarios as early as possible, during the development and analysis of requirements, assures that the product's final realization is considered early. This practice can expose developers to validation issues and testers to development issues, helping to flush out potential problems earlier and lead to higher quality software.

Cross Functional Requirements Analysis Sessions: Analyzing requirements is a necessary component of any requirements process. Conducting such analyses as a group helps to ensure sufficient exploration of ideas. Attendance by representatives from major functional departments establishes a common understanding among groups, helps break down social barriers and ensures that requirements are considered from a variety of perspectives. Cooperation and collaboration within the organization promote a productive team environment.

These strategies represent a small but important contribution toward providing detailed advice to practitioners. By providing supportive empirical evidence, these strategies are framed by the context in which they are known to have been successful. One hopes that this comprehensive coupling of strategy, evidence and context provides effective, adoptable approaches for improving software risk management, quality and productivity.
6.5 Directions for Future Research

The evidence presented here provides the basis for an empirical model of requirements engineering process and how requirements engineering’s beneficial outcomes in risk management, quality and productivity are realized through interaction with other processes (as illustrated in Figure 5.1). Such a model would validate existing recommendations made in the requirements engineering literature (e.g. (Boehm, 1991; Jones, 1996; Robertson & Robertson, 2004)). The accumulation of empirical evidence would ideally be captured with a much-needed scientifically validated instrument to lend credibility to our understanding of tangible benefits and process interactions. Validating the relationships between requirements process and risk management activities should therefore be on the agenda for future studies.

Notwithstanding the importance of firmly establishing the benefits of requirements process, it is also necessary to measure the cost of realizing such improvement. The costs of training, organizing and developing requirements should be compared to other productive activities such as design and implementation. For example, instead of analyzing requirements, would ACUS (and its customers and users) have been equally well off spending those eight months doing additional design, implementation and testing?

Finally, ACUS made practical use of electronic support in conducting its JAD-like negotiation sessions among distributed stakeholders. Their success suggests the need for further study to explore how distance collaborations such as these can be improved through technological support,
rather than only process support.

6.6 Contributions

The work presented in this thesis is important for its contributions toward building a more complete understanding of how requirements engineering relates to and how it affects software development. Of those contributions the most significant are listed below:

- The evidence collected during this study provides important data toward supporting existing claims in requirements research. This evidence is particularly significant because it has been systematically collected from multiple sources so that qualitative developer responses have been corroborated using objective project data (such as estimations and defect rates).

- This evidence is used to illustrate, in detail, the apparent effects of the requirements engineering process in improving risk management, quality and productivity throughout the development life cycle. The evidence that is provided not only establishes the extent of these effects, but also confirms the requirements process’ causal role in producing the said effects.

- The investigation and analysis of the above evidence serves to untangle the complex interactions of software development processes to reveal the relationship between requirements processes and other planning and development processes.
Chapter 6. Conclusions

- The development of the interaction map (Figure 5.1) combines both the evidence of payoffs with the data collected regarding the interaction of processes. This interaction map shows the interdependencies of processes and contributes toward a theory of requirements engineering.

- I have proposed a number of unique research opportunities which have the potential to add to requirements theory and improve requirements engineering practice. These research opportunities include: the role of collaboration and communication in requirements and software development; the demphasization of tangible requirements artefacts; and a proposed requirements process component prioritization so that practitioners may plan adoption and optimize software development.

- Exploring the limitations of this work not only admits its shortcomings, but reveals the fundamental limits of existing instruments for conducting this type of empirical research.

6.7 Closing Remarks

This research has shown how requirements engineering can affect software development project according to the experiences of one software development organization. It shows how the requirements engineering process is part of a larger dynamic organization of software development processes which are to some extent all related and dependant on each
other. This thesis has enumerated the measurable effects of requirements engineering and to what extent the revised requirements process benefited other planning and development processes. Although this research is but one example of a positive experience in requirements engineering, it provides much-needed empirical basis for advancing our understanding of how requirements engineering relates to software development practices. Clearly, further research is necessary to corroborate these findings and add to current theory in requirements engineering.
Bibliography


Bibliography


Bibliography


Bibliography


Appendix A

Questionnaire 2

Q1 How important do you feel the revised requirements process was at ACUS? Why?

Very Important □ Important □ Intermediate □ Somewhat Important □ Not Important at All □

Q2 How did the revised requirements process help you in your work?

Q3 With respect to the ‘requirements phase’, in the future, would you spend more or less time and/or effort on this phase of development? (Why?)

Far More □ More □ Same □ Less □ Far Less □

Q4 The revised requirements process consisted of several components as shown below. Please indicate which ones were most important/useful in your work in activities such as design, implementation, testing or documentation:
## Appendix A. Questionnaire 2

<table>
<thead>
<tr>
<th>Components of the REP</th>
<th>Design</th>
<th>Implementation</th>
<th>Testing</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross-Functional Teams</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Group Analysis Sessions</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Sentence Template</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Structured Req Spec</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Traceability to Req Rationale</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Traceability to Test Scenarios</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Definition of Test Scenarios</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Context Diagram</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
</tbody>
</table>

**Q5** Do you feel that the improved requirements process revealed further details, inter-dependencies and complexities of features?

<table>
<thead>
<tr>
<th>Strongly Agree</th>
<th>Agree</th>
<th>No Effect</th>
<th>Disagree</th>
<th>Strongly Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
</tbody>
</table>

**Q6** Have you, during design, implementation, testing or documentation, made use of the technical requirements...

<table>
<thead>
<tr>
<th>Strongly Agree</th>
<th>Agree</th>
<th>Neutral</th>
<th>Disagree</th>
<th>Strongly Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
</tbody>
</table>

- to re-familiarize yourself with the feature
- to gain deeper understanding of the feature
- to facilitate rational design
- to understand motivation behind the feature
- to ensure complete coverage / compliance

**Q7** Compared to projects in the past: has there been more or less rework during development (but before deployment):

<table>
<thead>
<tr>
<th>Far More</th>
<th>More</th>
<th>Same</th>
<th>Less</th>
<th>Far Less</th>
</tr>
</thead>
<tbody>
<tr>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
</tbody>
</table>
Appendix A. Questionnaire 2

Q8 Did understanding help...  

<table>
<thead>
<tr>
<th>Question</th>
<th>Strongly Agree</th>
<th>Agree</th>
<th>Neutral</th>
<th>Disagree</th>
<th>Strongly Disagree</th>
<th>Don't Know</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facilitate more informed decisions?</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
<tr>
<td>Improve your sense of ownership?</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
<tr>
<td>Inspire you to be more creative?</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
<tr>
<td>Improve communication with customers?</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
</tbody>
</table>

Q9 Based on an early assessment the requirements process improved communication among developers. Did communication generated by these GAS continue into later stages of development?

<table>
<thead>
<tr>
<th>Response</th>
<th>Much Improved</th>
<th>Improved</th>
<th>Unsure</th>
<th>No Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
</tbody>
</table>

Q10 In an earlier assessment it was reported that engineers often only vaguely understood the requirements and often had to “walk to others’ cubicles and ask for clarifications”. To what extent do you believe the revised requirements process prevented this phenomenon in the current project?

Q11 How do you believe the communication inspired by the requirements analysis sessions improved or deteriorated in the current project (a) productivity or (b) product quality?

Q12 How important was the use of features and technical requirements in the estimation process?

<table>
<thead>
<tr>
<th>Importance</th>
<th>Very Important</th>
<th>Important</th>
<th>Unsure</th>
<th>Not Really Important</th>
<th>Not Important at All</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
</tbody>
</table>

Q13 In your design, coding testing or documentation activities, how important is it to understand the features and technical requirements
Appendix A. Questionnaire 2

Q14 How important was the use of technical requirements in estimation for:

<table>
<thead>
<tr>
<th></th>
<th>Very Important</th>
<th>Important</th>
<th>Neutral</th>
<th>Not Really Important</th>
<th>Not Important at All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Documentation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q15 What was the source of failure of inaccuracy in the above-mentioned areas (resource allocation and planning)?

Q16 In thinking about your estimates, can you think of reasons for discrepancies? With respect to design, implementation testing and documentation?

Q17 How could you have improved your estimations?

Q18 Please provide an estimate of the number of defects you addressed so far in this project and the percentage of those defects that could be traced requirements, design and implementation problems.

Change Management

Q19 To what extent did the new requirements engineering process enable your organization to manage requirements: a) to analyze risk and cost? b) to assess impact of changing requirements?

Marketing Unit Negotiations (Managers Only)

Q20 Do you believe that the articulation of technical requirements (and more thorough analysis) empower your organization in their decision making with the marketing unit (MU)?

Q21 Do you believe that your organization able to more "comfortably” accept or reject MU requests?
Q22 Do you believe that the refinement of features into technical requirements result in further detailed rationale behind requirements to be available from the MU?

Q23 Do you believe that the improved MU process was responsible for this result?

Q24 Did the requirements analysis session part of the RE process lead to a better understanding of MU (or customer) expectations?

Q25 How useful were [feature] estimates during negotiations with the MU?

Q26 How were the requirements (tech and customer) used to analyze the impact of changes?
Appendix B

Questionnaire 3

The following is the paper equivalent of questionnaire 3 which was administered exclusively via the web. A screen shot of the actual web form is provided below that demonstrates the precise interface used by respondents.

<table>
<thead>
<tr>
<th>Development processes</th>
<th>Rate RE impact on process</th>
<th>Requirements process component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Planning and Tracking</td>
<td>-3 ○ ○ ○ ○ ○ ○ ○ ○  +3</td>
<td></td>
</tr>
<tr>
<td>feature sizing</td>
<td>-3 ○ ○ ○ ○ ○ ○ ○ ○  +3</td>
<td></td>
</tr>
<tr>
<td>risk assessment</td>
<td>-3 ○ ○ ○ ○ ○ ○ ○ ○  +3</td>
<td></td>
</tr>
<tr>
<td>scheduling</td>
<td>-3 ○ ○ ○ ○ ○ ○ ○ ○  +3</td>
<td></td>
</tr>
<tr>
<td>resource planning</td>
<td>-3 ○ ○ ○ ○ ○ ○ ○ ○  +3</td>
<td></td>
</tr>
<tr>
<td>change management</td>
<td>-3 ○ ○ ○ ○ ○ ○ ○ ○  +3</td>
<td></td>
</tr>
<tr>
<td>responsibility allocation</td>
<td>-3 ○ ○ ○ ○ ○ ○ ○ ○  +3</td>
<td></td>
</tr>
<tr>
<td>requirements tracing</td>
<td>-3 ○ ○ ○ ○ ○ ○ ○ ○  +3</td>
<td></td>
</tr>
<tr>
<td>Software Quality Assurance</td>
<td>-3 ○ ○ ○ ○ ○ ○ ○ ○  +3</td>
<td></td>
</tr>
<tr>
<td>tracking</td>
<td>-3 ○ ○ ○ ○ ○ ○ ○ ○  +3</td>
<td></td>
</tr>
<tr>
<td>SQA team</td>
<td>-3 ○ ○ ○ ○ ○ ○ ○ ○  +3</td>
<td></td>
</tr>
<tr>
<td>meetings and reports</td>
<td>-3 ○ ○ ○ ○ ○ ○ ○ ○  +3</td>
<td></td>
</tr>
<tr>
<td>deviations</td>
<td>-3 ○ ○ ○ ○ ○ ○ ○ ○  +3</td>
<td></td>
</tr>
<tr>
<td>Software Configuration Mgmt</td>
<td>-3 ○ ○ ○ ○ ○ ○ ○ ○  +3</td>
<td></td>
</tr>
<tr>
<td>baselining</td>
<td>-3 ○ ○ ○ ○ ○ ○ ○ ○  +3</td>
<td></td>
</tr>
<tr>
<td>change management</td>
<td>-3 ○ ○ ○ ○ ○ ○ ○ ○  +3</td>
<td></td>
</tr>
<tr>
<td>change metrics</td>
<td>-3 ○ ○ ○ ○ ○ ○ ○ ○  +3</td>
<td></td>
</tr>
<tr>
<td>role adherence</td>
<td>-3 ○ ○ ○ ○ ○ ○ ○ ○  +3</td>
<td></td>
</tr>
<tr>
<td>sw change control board</td>
<td>-3 ○ ○ ○ ○ ○ ○ ○ ○  +3</td>
<td></td>
</tr>
<tr>
<td>SCM tools</td>
<td>-3 ○ ○ ○ ○ ○ ○ ○ ○  +3</td>
<td></td>
</tr>
</tbody>
</table>

For each, optionally chose one of:
- decomposition of features
- requirements traceability
- group req analysis sess.
- cross functional teams
- structured requirements doc
- defn of test scenarios
# Appendix B. Questionnaire 3

## Development Processes

<table>
<thead>
<tr>
<th>Development Processes</th>
<th>Rate RE Impact on Process -3 to 0 to +3</th>
<th>Requirements Process Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development</td>
<td>-3 ooooooo  +3</td>
<td></td>
</tr>
<tr>
<td>team reorganization</td>
<td>-3 ooooooo  +3</td>
<td></td>
</tr>
<tr>
<td>split leads</td>
<td>-3 ooooooo  +3</td>
<td></td>
</tr>
<tr>
<td>planning</td>
<td>-3 ooooooo  +3</td>
<td></td>
</tr>
<tr>
<td>conformance</td>
<td>-3 ooooooo  +3</td>
<td></td>
</tr>
<tr>
<td>inspection</td>
<td>-3 ooooooo  +3</td>
<td></td>
</tr>
<tr>
<td>Testing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>automation</td>
<td>-3 ooooooo  +3</td>
<td></td>
</tr>
<tr>
<td>testing against requirements</td>
<td>-3 ooooooo  +3</td>
<td></td>
</tr>
<tr>
<td>peer-review</td>
<td>-3 ooooooo  +3</td>
<td></td>
</tr>
<tr>
<td>repeated testing</td>
<td>-3 ooooooo  +3</td>
<td></td>
</tr>
</tbody>
</table>

For each, optionally choose one of:
- decomposition of features
- requirements traceability
- group req analysis sess.
- cross functional teams
- structured requirements doc
- defn of test scenarios