THE EMERGENCE OF CHANGE AT THE SYSTEMS ENGINEERING AND SOFTWARE DESIGN INTERFACE

:: AN INVESTIGATION OF IMPACT ANALYSIS ::

THIS DISSERTATION IS
SUBMITTED FOR THE DEGREE
OF DOCTOR OF PHILOSOPHY

MALIA SOFIA KILPINEN

To my mom

DECLARATION

This dissertation is the result of my own work and includes nothing that is the outcome of work done in collaboration except where specifically stated in the text. This thesis has not been submitted in whole or in part for consideration for any other degree qualification at this university or any other institution of learning. This thesis contains 126 figures, 34 tables, and less than 85,000 words.

Malia Sofia Kilpinen
August 2008

ABSTRACT

Systems engineers and software designers collaborate throughout product development to delineate and fulfil software requirements. Given the flexibility of software, the effects of software requirement and detail design changes must be understood in order to reduce the risk of ripples of further, unanticipated modifications. As such, these designers both perform *impact analysis* to estimate the consequences of design changes. This dissertation examines the impact analysis performed by systems engineers and software designers as prescribed by literature and as practised within industry and then investigates the implications of impact analysis improvement on the design process.

Literature only mentions the application of impact analysis in practice and often conflicts in describing how impact analysis should be performed. In turn, two extensive empirical studies within two global companies explore and characterise the challenges of implementing prescribed impact analysis techniques. Through the discussion of specific changes and the associated impact analysis applied, interviewees provide suggestions for practical improvement strategies to address these challenges. The effect of impact analysis improvement on the design process through such strategies is then further investigated through modelling and simulation using system dynamics. The simulation of a collaborating company's design process shows that the time to complete design work can be reduced by about 60% through marginally improving IA and also is about twice as sensitive to IA improvement than other improvement strategies (*e.g.* optimising design task scheduling or improving requirements management). As a result, heuristics for impact analysis improvement are developed. These heuristics indicate that the quality of impact analysis results and when the analysis is performed during change processes can significantly influence the rate of design project progress. The evaluation of the heuristics by the industry collaborators suggests the utility of strategically reforming impact analysis practice to provide for software design process improvement. Consequently, this dissertation focuses on the effects of impact analysis improvement and argues for elevation of the importance of impact analysis within software design processes.

ACKNOWLEDGEMENTS

Much *mahalo* to:

Professor John Clarkson
for his guidance

Dr. Claudia Eckert
for her support

The collaborators from the aerospace and telecommunications companies
for contributing their thoughts and insights

The Change Management and Process Management Groups,
especially Anja Maier, Rene Keller, and David Wynn
for proof-reading this dissertation

and

Tido Eger
for his companionship, encouragement, and love

CONTENTS

1 :: INTRODUCTION 1

1.1 HISTORY OF THE SYSTEMS ENGINEERING AND SOFTWARE DESIGN INTERFACE 4
1.2 SCOPING THE SYSTEMS ENGINEERING AND SOFTWARE DESIGN INTERFACE 7
1.3 EMERGENT CHANGES IN COMPLEX SYSTEM DESIGN 9
1.3.1 The Effects of Unexpected, Emergent Changes 10
1.3.2 The Increasing Importance of Effective Change Management 11
1.3.3 Change Management Research 12
1.4 IMPACT ANALYSIS TO MANAGE EMERGENT DESIGN CHANGES 12
1.5 RESEARCH METHODOLOGY 14
1.6 RESEARCH QUESTIONS 17
1.7 DISSERTATION STRUCTURE 19

2 :: LITERATURE ON SYSTEM DESIGN AND SOFTWARE CHANGES 21

2.1 THE LITERATURE SEARCH PROCESS 21
2.1.1 Literature Published within Engineering Disciplines 22
2.1.2 Literature Focusing on the Design Process 24
2.2 LITERATURE REVIEW DISCUSSION STRUCTURE 25
2.3 DESIGN CHANGE CHARACTERISATIONS 26
2.4 PRESCRIBED DESIGN AND CHANGE PROCESSES 28
2.4.1 Systems Engineering Process Models 29
2.4.2 Software Design Process Models 31
2.4.3 Change Process Models 36
2.5 IMPACT ANALYSIS TECHNIQUES 38
2.5.1 Traceability Impact Analysis 39
2.5.2 Dependency Impact Analysis 43
2.5.3 Experiential Impact Analysis 45
2.6 IMPACT ANALYSIS IN PRACTICE 47
2.6.1 Traceability Impact Analysis in Practice 48

2.6.2 Dependency Impact Analysis in Practice *49*
2.6.3 Experiential Impact Analysis in Practice *49*
2.7 SCARCITIES AND DISPARITIES IN IMPACT ANALYSIS LITERATURE 50
2.8 SUMMARY 51

3 :: EMPIRICAL STUDIES ON SYSTEM DESIGN AND SOFTWARE CHANGES 53

3.1 METHOD OF THE EMPIRICAL STUDIES 54
3.1.1 Grounded Theory *54*
3.1.2 Data Collection and Analysis *55*
3.1.3 Phase 1 – The Exploratory Empirical Study *59*
3.1.4 Phase 2 – The Impact Analysis Empirical Studies *59*
3.2 EMPIRICAL STUDIES DISCUSSION STRUCTURE 61
3.3 INTRODUCTION TO THE AEROSPACE AND TELECOMMUNICATIONS FIRMS 62
3.4 FINDINGS FROM THE AEROSPACE COMPANY 63
3.4.1 The Systems Engineering and Software Design Interface *63*
3.4.2 The Prescribed Change Processes and Impact Analysis Techniques *66*
3.4.3 The Change Processes in Practice *70*
3.4.4 The Impact Analysis Techniques in Practice *72*
3.4.5 The Strategies for Improving the Design Process *73*
3.5 FINDINGS FROM THE TELECOMMUNICATIONS COMPANY 74
3.5.1 The Systems Engineering and Software Design Interface *75*
3.5.2 The Context of Implementing Agile Development Methods *75*
3.5.3 The Prescribed Change Processes and Impact Analysis Techniques *76*
3.5.4 The Change Processes and Impact Analysis Techniques in Practice *77*
3.5.5 The Strategies for Improving the Design Process *78*
3.6 SUMMARY 78

4 :: IMPACT ANALYSIS CHARACTERISATIONS 80

4.1 IMPACT ANALYSIS TECHNIQUES AND TASKS 80
4.2 IMPACT ANALYSIS QUALITY 81
4.3 RIGOUR OF IMPACT ANALYSIS TECHNIQUES AND TASKS 81
4.4 INFLUENCES ON IMPACT ANALYSIS TECHNIQUES AND TASKS 84
4.4.1 Technique Influences *87*
4.4.2 Task Influences *93*
4.5 EVALUATION OF THE IMPACT ANALYSIS CHARACTERISATIONS 98

4.5.1 Comparison of the Characterisations with Literature ... 98
4.5.2 Evaluation of the Characterisations by Interviewees ... 101
4.6 SUMMARY ... 104

5 :: THE APPLICATION OF IMPACT ANALYSIS IN PRACTICE ... 106

5.1 AN ELICITATION METHOD FOR IMPACT ANALYSIS PRACTICE ... 107
5.1.1 Development of the Elicitation Method Structure ... 108
5.1.2 Change Attributes for Impact Analysis Technique Elicitation ... 111
5.1.3 Application of the Elicitation Method ... 115
5.2 IMPACT ANALYSIS IN PRACTICE AT THE AEROSPACE COMPANY ... 116
5.2.1 Data Collection ... 116
5.2.2 Key Observations of Impact Analysis Practice ... 120
5.3 IMPACT ANALYSIS IMPROVEMENT STRATEGIES ... 127
5.3.1 Including More Impact Analysis within Change Processes ... 127
5.3.2 Improving the Input Quality to Impact Analysis ... 129
5.3.3 Advancing the Impact Analysis Techniques Performed ... 130
5.4 EXTRACTION OF PRACTICAL IMPACT ANALYSIS IMPROVEMENT STRATEGIES ... 131
5.5 REVISITING THE RESEARCH QUESTIONS ... 132
5.6 RESEARCH QUESTION REFINEMENT ... 133
5.7 SUMMARY ... 134

6 :: INVESTIGATING THE IMPLICATIONS OF IMPACT ANALYSIS IMPROVEMENT ... 136

6.1 LITERATURE ON ANALYSING DESIGN PROCESS IMPROVEMENT ... 136
6.1.1 System Dynamics ... 137
6.1.2 Software Process Dynamics ... 143
6.2 MODELLING AND SIMULATION METHOD AND DISCUSSION STRUCTURE ... 145
6.3 MODELLING THE DESIGN PROCESS VIA THE ADAPTED REWORK CYCLE ... 146
6.3.1 Rework Discovery Time Delay ... 147
6.3.2 Impact Analysis Quality ... 149
6.3.3 Change Packaging ... 153
6.3.4 Limitations ... 154
6.3.5 Evaluation of the Adapted Rework Cycle Model ... 157
6.4 BEHAVIOUR OF THE ADAPTED REWORK CYCLE ... 159
6.4.1 Qualitative Behaviour of the Adapted Rework Cycle ... 160

6.4.2 *Quantitative Behaviour of the Adapted Rework Cycle* *163*
6.4.3 *Reference Modes for the Adapted Rework Cycle Simulation* *167*
6.5 SUMMARY 172

7 :: **SIMULATING IMPACT ANALYSIS IMPROVEMENT 174**

7.1 SIMULATION OF THE ADAPTED REWORK CYCLE 174
7.1.1 *Simulation Design* *174*
7.1.2 *MATLAB Implementation* *182*
7.2 SIMULATION VARIABLE VALUES AND INITIAL CONDITIONS 183
7.3 TESTING THE ADAPTED REWORK CYCLE SIMULATION 185
7.3.1 *Reference Modes* *186*
7.3.2 *Extreme Conditions* *188*
7.3.3 *Sensitivity Analysis* *189*
7.4 EVALUATION OF THE IMPACT ANALYSIS IMPROVEMENT STRATEGIES 192
7.4.1 *Simulation of the Impact Analysis Improvement Strategies* *192*
7.4.2 *Simulation Results* *194*
7.4.3 *Interpretation of the Impact Analysis Improvement Strategies* *200*
7.5 EXAMINING PROCESS IMPROVEMENT THROUGH CHANGE PACKAGING 202
7.5.1 *Simulation of Change Packaging Strategies* *202*
7.5.2 *Simulation Results* *204*
7.5.3 *Interpretation of the Change Packaging Strategies* *206*
7.6 SUMMARY 207

8 :: **HEURISTICS FOR PROCESS IMPROVEMENT 209**

8.1 REVISITING THE REFINED RESEARCH QUESTIONS 210
8.2 IMPACT ANALYSIS IMPROVEMENT AND CHANGE PACKAGING HEURISTICS 211
8.2.1 *Extending the Adapted Rework Cycle Model* *211*
8.2.2 *Extraction of Heuristics for Impact Analysis Improvement* *215*
8.2.3 *Extraction of Heuristics for Change Packaging* *225*
8.3 EVALUATION OF THE HEURISTICS 226
8.3.1 *Falsehood* *227*
8.3.2 *Utility* *229*
8.3.3 *Transferability* *230*
8.3.4 *Applicability* *230*
8.4 SUMMARY 231

9 :: CONCLUSION 234

9.1 KEY CONCLUSIONS STEMMING FROM THE RESEARCH QUESTIONS 234
9.2 CONTRIBUTIONS 238
9.3 FUTURE WORK 239
9.4 SUMMARY 241

1 :: Introduction

Harnessing the power of software flexibility demands careful management. Without changing any physical hardware, software can modify a product or service to address different markets or customer needs and deliver the desired functionality. However, such software design changes also can cause unexpected product or system performance, revealing the challenge of change management. Many "software failures" have been publicised extensively, but actually have been instigated by changes to the overall product or system design, which the software design was intended to accommodate (Sommerville 2001: 31). The knock-on effects of making design changes may not be taken into account within the high-level design, and, thus, may not be implemented within the software design, causing a failure. Alternatively, system design modifications can lead to software changes, which can inadvertently cause unintended and unspecified system behaviour. The software is consequently reported to fail, but, arguably, the system design also contributes to the failure.

The Therac-25 cancer radiation therapy machine accidents exemplify change mismanagement and are often rated as some of the worst software-related failures (Haskins 2006: 3.14). Between June 1985 and January 1987, the Therac-25 caused at least six accidents either resulting in patient death or serious injury due to radiation overdoses. The Therac-25 was largely developed based on prior designs, namely, the Therac-6 and Therac-20, and advanced the software control system used in these versions. The software control system enhancements also provided some of the functionality previously contained within the hardware design. In particular, the hardware interlocks, used to physically stop the machine from delivering high radiation doses or performing unsafe operations, were re-implemented within the software design. In turn, this redesign of the machine functionality relied only on the software interlocks to prevent malfunctioning (Leveson and Turner 1993).

Although no single cause can be attributed to the Therac-25 failures, or accidents in general, the interlock design change arguably contributed to the risk of radiation overdoses. An error in the Therac-20 software design was carried over into the

Therac-25 machine, and this software design fault became active and affected operation only after the removal of the hardware interlocks. This latent dependency between the hardware and software interlock designs was not recognised, and, in turn, the necessary changes to the software interlocks were not initiated, leading to at least two of the patient accidents (Leveson and Turner 1993). Since the complete impact of the interlock design modification on the system performance was not determined, the system design failed to some extent (Neumann 1995: 73). Furthermore, given that the Therac-25 was also plagued by poor software design processes and inadequate testing, leaving such hidden software faults (Leveson and Turner 1993), the software design also contributed to the failure by not meeting the intended system requirements (Neumann 1995: 73).

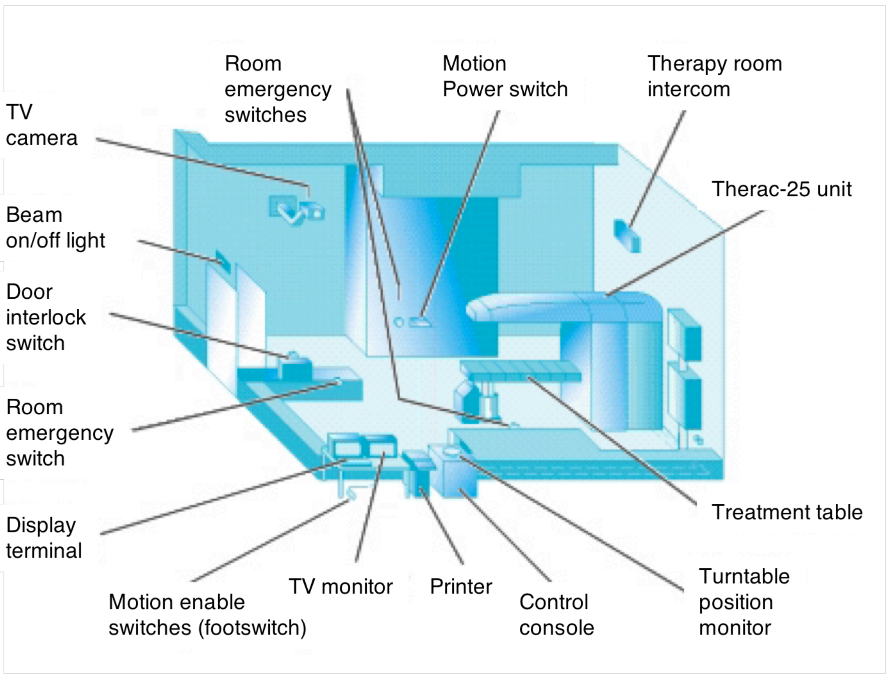


Figure 1.1: Components of the Therac-25 system (Miller *et al.* 2000)

Leveson and Turner (1993) note that the Therac-25 accidents should not only be classified due to a "software failure", but also should be viewed from a systems engineering perspective. The "software failure" is only a symptom of the Therac-25 accidents, and the system dependencies must also be understood to address the fundamental causes of the design failure. By recognising that an error within the software interlock design could singularly cause a failure in the Therac-25 machine, measures to ensure safe operation could occur at the system-level by adding hardware or software redundancy. Thus, the investigation of the Therac-25 accidents specifically suggests that evolutionary design changes can have serious consequences on system performance and that the impact of design changes must be thoroughly analysed to provide for the desired product and software behaviour and reduce the risk of unsafe operation.

Although implementing effective change management can reduce the risk of product or project failures in other product or service domains, such as space, avionics, transportation, and defence systems (Neumann 1995: 12-95), medical devices epitomise the importance of change management due to their direct effect on human health and life. In particular, the complete impact of design changes must be determined and implemented in such products to allow for the safe treatment of patients. Research by Wallace and Kuhn (2001) indicate that the challenges of change management continue to transpire in this industry. By investigating 15 years of medical device recall data from 1983 to 1997 collected by the U.S. Food and Drug Administration (FDA), they conclude that at least 23 devices were taken off the market specifically due to "change impact" faults. The potential for failures in these devices was caused by the unanticipated and unaccounted side effects of software changes during product development. More recent studies indicate that the number of FDA recalls due to "software errors" is sharply rising, suggesting that change management may still require improvement (Johnson 2006). Given that this medical device software is reviewed and approved by the FDA, this occurrence of design flaws indicates that testing, verification, and validation may not be sufficient to identify all errors in safety-critical systems or other product designs. The complete impact of changes must be actively determined during the development process as opposed to passively found as design faults during design evaluation stages.

While the impact of the design flaws in the recalled medical devices did not cause loss of life or significantly affect the health of any patients since they were found prior to any accidents, the recall of the equipment inevitably produced financial losses for the medical companies. In turn, effective change management can reduce the risk of design failure as well as unanticipated rework and design costs. Similarly, if necessary changes are detected during the design process, design tasks also can be scheduled efficiently, avoiding potential delivery delays. Most software projects inevitably have to deal with these risks to some extent (Rosenberg 2007), and when any of these risks are taken to an extreme, blockbuster project failures can occur (Charette 2005; Flowers 1996; Peterson 1995). Moreover, as software is increasingly incorporated into products (Huhn and Schaper 2006), understanding the impact of system and software design changes becomes ever more critical for many engineering projects.

This challenge of managing design changes has surfaced throughout literature on the systems engineering and software design interface. As such, an exploration of the discourse on this interface's history establishes the context for this research (Section 1.1). In turn, this dissertation sets the scope for investigating the systems engineering and software design interface for this research project (Section 1.2). The implications of improving change management, as suggested by the discussion of the medical device industry, are then considered further (Section 1.3), and a perspective for investigating design change is suggested (Section 1.4). This chapter concludes by describing the research methodology and questions (Section 1.5 and Section 1.6, respectively) and by outlining the remainder of this dissertation (Section 1.7).

1.1 HISTORY OF THE SYSTEMS ENGINEERING AND SOFTWARE DESIGN INTERFACE

The fundamental concepts of *systems engineering*[1] emerged in Bell Telephone Laboratories in the early 1900s. The Director of Systems Engineering at Bell, Mr. Gilman, formalised these concepts and began teaching the first course on systems engineering at MIT in 1950. Throughout the next decades, industry and government practitioners, notably the RAND Corporation and the U.S. Department of Defense (DoD), implemented systems engineering and advanced the processes, techniques, and tools in this discipline (Buede 2000: 6).

In one of the first formal texts on systems engineering written in 1962, Hall stresses that systems engineering should be applied across the phases of product development, including problem definition and the design, integration, and testing of system components (Buede 2000: 6). While the definitions of systems engineering have evolved, a holistic perspective on planning and designing a product has remained central to the discipline. In 2004, the International Council on Systems Engineering (INCOSE), a leading professional society focused on developing and disseminating systems engineering practices, defined "systems engineering" as:

> *an interdisciplinary approach and means to enable the realization of successful systems. It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, then proceeding with design synthesis and system validation while considering the complete problem: operations, performance, test, manufacturing, cost and schedule, training and support, and disposal. Systems Engineering integrates all the disciplines and speciality groups into a team effort forming a structured development process that proceeds from concept to operation (INCOSE 2004b).*

Although no single definition of systems engineering exists in literature, systems engineering intends to coordinate the many disciplines involved throughout product

[1] The term *system design* is used interchangeably with *systems engineering* in this dissertation.

development. Descriptions by Sage (2000: 3), Kossiakoff and Sweet (2003: 4-5), and Blanchard (2003: 29-40) highlight this integration of specialist design disciplines (*e.g.* mechanical, electrical, and software engineering) as well as design process principles (*e.g.* configuration management and concurrent engineering).

As systems engineering theories and practices have matured, *software engineering*[2] has become increasingly integrated into product design (Peterson and Hiles 1976), and means to integrate software design into systems engineering processes have been investigated[3]. Sage wrote one of the first in-depth discussions of this interface in <u>Software Systems Engineering</u>, originally published in 1989 and as a second edition in 1990. Sage argues for the need to understand and carefully implement this interface of systems engineering and software design by citing the budget and schedule overruns frequently experienced in practice as well as the poor quality and reliability of software at the time (Sage 1990: 84-88). He specifies that complete, consistent, and feasible requirements developed by systems engineers should guide the software design and allow for the seamless integration of software into product designs (Sage 1990: 240). As such, requirements continue to be a key artefact of the systems engineering and software design interface.

In 1993, Andriole and Freeman analysed the systems engineering and software design interface and acknowledged a lack of literature regarding this disciplinary interaction. They conclude that this interface requires a multi-disciplinary approach to education and research efforts (Andriole and Freeman 1993). The concept of unifying systems engineering with software design processes has since been acknowledged in literature within several design disciplines, including systems engineering (Thomé 1993), software engineering (Bate 1998; Sommerville 1998; White 2005), requirements engineering (Gonzales 2005), and design process research, such as concurrent engineering (Kaindl 2005), as well as has been mentioned in design standards[4], such as the ISO/IEC 15288 standard on systems engineering (ISO/IEC 2002), the IEEE 1220 standard on the application and management of systems

[2] The <u>IEEE Standard Glossary of Software Engineering Terminology</u> (1993) defines "software engineering" as "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software".
[3] Literature does not suggest software engineering to replace systems engineering in any way, but synthesise the software design with the overall product development. Hence, this dissertation refers to this interaction of these design disciplines as the *systems engineering and software design interface*.
[4] Sheard (2001) describes the evolving relationships among these design standards and others. Since Sheard's publication, INCOSE in conjunction with the Electronics Industry Association (EIA) developed ISO/IEC 15288, based on EIA 632 (Harwell 1998; Martin 1998). IEEE (2005a) then adopted this standard into IEEE 15288, leading to some commonality between the systems and software engineering communities. In addition, the terminology used in the latest version of CMMI (2006: 532) draws from ISO/IEC 15288 and corresponds with ISO/IEC 15504, which similarly describes process maturity levels.

engineering (IEEE 2005b), and the Capability Maturity Model Integration (CMMI)[5] for integrated product and process development (CMMI 2006). These publications identified are the only found to date to specifically address and elaborate on the systems engineering and software design interface. However, since they do not thoroughly analyse handling design changes, a scarcity of research at this interface persists.

Nevertheless, this literature on the interface of systems engineering and software design suggests design changes can be problematic. For instance, Sommerville (1998) addresses the practice of modifying software within a system design and the consequences on the development schedule and budget, writing:

> *Budget over-runs and delays in delivery are inevitable when the developers of the software must constantly accommodate change… We must accept that other problems of systems engineering will inevitably mean that there will be demands for software changes at a late stage in the development process. Cost and schedule overruns are the price we have to pay for system flexibility.*

As suggested, the flexible nature of software can provide for system design changes, but may have implications on project management. However, Sommerville does not tackle specific change management improvements in this text.

In turn, Bate (1998) argues that requirements engineering practices at the systems engineering and software design interface fundamentally make design modifications difficult to handle and states:

> *For many years, there has been an artificial barrier between these two disciplines. Software engineers often receive incomprehensible or inadequate requirements and specifications, frequently at the last moment. They are expected to turn out the software quickly and rapidly adjust to any problems or changes late in the development cycle.*

Similarly, if the system requirements given to software engineers to implement include errors, these modifications can cause further, unanticipated changes, leading problems in managing the change process. Sage (1990: 125) writes:

> *Errors introduced at the onset of system and software requirements specifications, if not detected early, will continue to propagate throughout the entire design and development lifecycle. They will not be picked up early through the programming phase, as this activity occurs only after the errors are inserted into the detail design. Thus, there is a need for validation of system and software requirements specifications early in the design effort.*

As such, Bate and Sage indicate that improving requirements engineering practices at the interface can provide for better change management and interaction of the

[5] Boehm (2000) and Gibson (2003) highlight the systems engineering and software design interface with respect to CMMI.

systems and software engineering disciplines. However, they do not consider alternative means for improving the handling of design changes.

While such remarks on the challenges of design changes and potential improvement strategies have been intertwined with the evolving discussion of the systems engineering and software design interface, no panacea to managing design changes at the interface has been found or may exist. Likewise, no in-depth study on the specifics of handling design changes at the systems engineering and software design interface has occurred either. This research project builds on this history of analysing the systems engineering and software design interface and investigates design change, thereby, filling a gap in the literature to date and addressing the continued challenges of change management in industry, as suggested by the opening discussion of medical devices in this dissertation.

1.2 SCOPING THE SYSTEMS ENGINEERING AND SOFTWARE DESIGN INTERFACE

Since systems engineering and software design must be coordinated throughout product development, encompassing the interactions of these disciplines succinctly in a definition can be problematic. For the purpose of this research project, the systems engineering and software design interface is delineated with respect to design changes. As observed in empirical studies (Chapter 3), such modifications stem from causes, including *new functionality requirements*, *design errors*, *design improvements*, or the *adaptation of legacy systems*. However, even with this focus, the word *interface* may be ambiguous. The Oxford English Dictionary defines "interface" (1989) as:

> *a means or place of interaction between two systems, organizations, etc.; a meeting-point or common ground between two parties, systems, or disciplines; also, interaction, liaison, dialogue.*

Based on this definition, identifying the "means or place" of a design change transmitted between the systems and software engineering "disciplines" describes the interface. However, similar to the term "interface", the words "means" and "place" can have many different interpretations and appear similarly ambiguous.

In order to distinguish the interface addressed, this research project interprets the definition of interface in terms of people (*i.e.* the "disciplines" enabling design change), product (*i.e.* the "place" of design change), and process (*i.e.* the "means" for design change). The people performing the systems and software engineering inevitably affect how design changes are handled at this interface. A designer's education, knowledge, and experience all influence the implementation of design changes. Similarly, team dynamics and communication contribute to the effective management of changes (Chudge and Fulton 1996; Crowston 1997; Koffi 2005; Wiegers 1994). This research project does not intend to specifically address these people aspects of handling design changes at the systems engineering and software design interface.

The products or artefacts of design changes also characterise this interface. For example, a change to a system requirement may link to modifications in the associated software design and code. The systems and software engineering disciplines interact through the objects affected by this design change, including *software requirement documentation*, *change requests*, *system and software design models*, *software design documentation*, or *code*, as observed in empirical studies (Chapter 3). In effect, these artefacts define an aspect of the systems engineering and software design interface. However, this research project does not focus on the style, semantics, or management tools used for such design change artefacts at the interface.

Instead, this research project views the systems engineering and software design interface in terms of change processes, and, more specifically, investigates *impact analysis* tasks within change processes (Section 1.4). These tasks, which scope the effects of design modifications, can be performed using design objects, such as analysing the documented traceability relationships between system requirements and software specifications. Design reviews held between systems and software engineers are another type of impact analysis task (Section 2.5)[6]. In turn, design-related procedures (*e.g.* configuration management) influence the change processes implemented and, consequently, the type of impact analysis practised. Hence, this dissertation investigates the systems engineering and software design interface[7] through first addressing the prescribed change processes and related procedures and then focusing on impact analysis tasks given this perspective.

[6] As suggested by these examples, the process interface between systems engineering and software design cannot be completely decoupled from the people and product aspects. People and company culture affect the process of design reviews, and the tools for and representations of design artefacts influence the change process applied. While this research focuses on change processes, links are made to people and products where relevant within this dissertation.

[7] The systems engineering and software design interface is abbreviated as the *systems-software* interface in the remainder of this dissertation.

1.3 EMERGENT CHANGES IN COMPLEX SYSTEM DESIGN

Design changes can significantly affect the progression of product development processes. For instance, during highly iterative design processes, rework may consume a large percentage of the development effort as changes evolve the state of the design. In this case, implementing effective change processes is vital to the overall design process. Similarly, change processes may also have a significant role in product customisation processes. Existing products may be reused with slight alterations to meet specific customer needs or adapt to new purposes. While most of the design may not change, the necessary modifications require effective management of design effort and rework. In effect, these product development processes are driven by the change processes implemented (Eckert *et al.* 2005).

Within such design processes, the successful planning of modifications provides a primary means for projects to stay on budget and schedule, as indicated by Sommerville (1998) and Bate (1998) within the literature focusing on the systems-software interface (Section 1.1). However, products can often have many complex interdependencies between and within disciplinary designs, and a design change can initiate additional modifications through these interactions. Design modifications can cause further, unexpected changes, leading to rework, additional costs, and schedule delays, as suggested by the initial discussion of the medical device industry. As such, managing this unanticipated *propagation* or *emergence* of additional changes also provides for effective product development processes.

Eckert *et al.* (2004) define two types of modifications, *emergent* and *initiated* changes. As termed, emergent changes are "caused by the state of the design, where problems occurring across the whole design and throughout the product lifecycle can lead to change". In other words, unanticipated problems or knock-on effects require additional, emergent design modifications. Given this perspective, design errors due to human mistakes, unlike more fundamental flaws in the design, are not considered emergent changes since they are not explicitly caused by the state of the design. In turn, Eckert *et al.* define initiated changes as "arising from an outside source, typically a new requirement from customers or certification bodies, or initiated by the manufacturer". Such initiated changes may be difficult to control, although, in some cases, negotiation with customers and manufacturers can help to manage these modifications. If the complete impact of an initiated change is not determined, unexpected, emergent changes can certainly arise.

Given that the management of emergent changes can shape the progression of design change processes, Eckert *et al.* (2004) describe two basic scenarios for the implementation of design modifications. Either initiated design changes and the associated emergent modifications can be implemented within a planned schedule, or these changes exceed the allotted amount of time. In the first situation, changes can *ripple* or *blossom*. A ripple pattern characterises the overall decrease in the number of changes and, therefore, the change effort. Thus, emergent changes are effectively managed in this scenario in that the rework generally diminishes with time. In turn, changes may blossom when large-scale modifications are required, meaning that significant redesign effort is required, but is possible to complete on schedule. Change propagation is also managed in this case since emergent changes are identified and implemented systematically. In the second case, Eckert *et al.* characterise the pattern of changes not completed on schedule as an *avalanche*, in which unexpected, emergent changes occur and require more design effort than budgeted. Figure 1.2 illustrates these patterns of change processes.

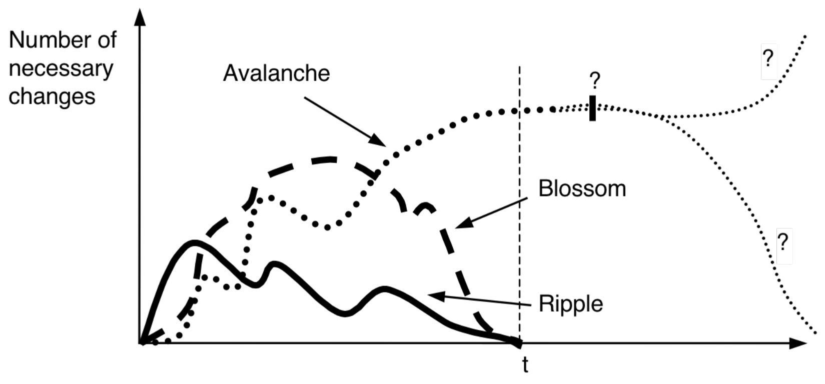


Figure 1.2: Characterisation of change processes (Eckert *et al.* 2004)

Although ripples, blossoms, and avalanches are described in terms of scheduling, the anticipation of the rework involved also can affect the budgeted development costs.

1.3.1 THE EFFECTS OF UNEXPECTED, EMERGENT CHANGES

While Eckert *et al.* (2004) qualitatively describe the potential consequences of unanticipated design modifications on change processes, the additional costs, schedule delays, and probability of product failure caused by such changes cannot be quantified in general. Many influences contribute to these same effects, including unrealistic project goals, inaccurate estimates of resources required, poorly defined system requirements, ineffective communication between stakeholders, stakeholder politics, and sloppy development practices (Brooks 1995; Charette 2005; Yourdon

1999). All of these elements must be addressed to provide for the holistic mitigation of these risks.

Nevertheless, given that software projects continue to face budget and schedule overruns, improving change management practices can support and enable more effective design processes. Charette (2005) describes 31 high-profile software projects from 1992 to 2005, which failed to meet customer requirements, budget, and schedule, and depicts a dismal future for software design. He suggests that 5 to 15 percent of software projects within a £500 billion worldwide market per year are abandoned shortly before or after release, while many others deliver over-budget and late products and services. Previously, in 2002, the U.S. National Institute of Standards and Technology pinned software failures as costing the U.S. economy alone approximately $59.5 billion every year (Rosenberg 2007). Although these statistics may be notional, reducing the notorious risks of such software development problems can more generally improve the operation of the businesses and governments that sponsor software projects. Effective software design change management contributes to this overall improvement.

1.3.2 THE INCREASING IMPORTANCE OF EFFECTIVE CHANGE MANAGEMENT

Software is incorporated more and more into product designs ranging from fast-moving consumer goods to large, complex systems, such as cars and airplanes. For instance, mobile phones currently have about 2 million lines of code, and, by 2010, they are expected to have 20 million lines of code. Also in 2010, the control systems in cars are expected to exceed 100 million lines of code (Charette 2005). The addition of more software into products increases the contribution of the software design to the functionality of the product. In turn, Huhn and Schaper (2006) estimate that software design will account for 15 percent of the revenue generated by a car in 2015 as opposed to only 4 percent in 2003. Similarly, Bill Gates (2007), founder of Microsoft, also suggests that robotics, which relies on software for intelligence, is fast becoming reality within product design and must be supported by software development tools and techniques.

Given that software projects have clearly been challenged by successfully delivering products and services on budget and schedule, the increase in incorporating software into products inevitably compounds the risks of additional costs, delays, and failures. Huhn and Schaper (2006) generally suggest focusing on improving the software design processes implemented in order to cope with the escalation of software within products. However, they specifically note that changes, due to design flaws, made to the software embedded in car control systems are much more expensive to implement than modifications for electronic hardware design errors. As such, improving design change management processes can decrease this cost of software and contribute to process improvement in light of the increase of software in products.

1.3.3 CHANGE MANAGEMENT RESEARCH

Change management for technical systems[8] is a developing field of research. While literature often mentions changes tangentially, research devoted to understanding and analysing changes and change processes is just emerging within engineering design and systems engineering publications (Section 2.1.2). This research project stems from work on change management undertaken at Cambridge Engineering Design Centre (EDC). Empirical studies on change processes beginning in 1999 have characterised the nature of design modifications and change processes within the development of complex products, such as helicopters, diesel engines, and jet engines (Eckert *et al.* 2005). As a result, the Change Prediction Method (CPM) has been developed to address the difficulties encountered in understanding the consequences of a change. Using a model of the dependencies between components within a product, the CPM yields a prediction and visualisations of the components potentially affected by a change. In turn, previously unanticipated changes can be systematically identified and analysed in order to plan a change process (Clarkson *et al.* 2004). This research project extends this work in change management of Cambridge EDC, which has focused on mechanical design, by considering other techniques used to examine the knock-on effects of initiating design modifications within systems and software engineering.

1.4 IMPACT ANALYSIS TO MANAGE EMERGENT DESIGN CHANGES

In order to address change management at the systems engineering and software design interface, this research project investigates means to completely identify required changes, thereby, reducing unanticipated, emergent modifications. As suggested by the scope set for the systems-software interface (Section 1.2), this topic is interpreted through change processes. Within change processes, *impact analysis* (IA)[9] is the primary means for evaluating the effects of changes.

[8] Change management within business literature is considered a separate field of research. This literature primarily discusses means of implementing organisational change or transformation (Jarratt 2004: 24) and is not addressed by this research project.
[9] Lindvall (1997b) has also used the abbreviation of "IA" for "impact analysis".

Bohner and Arnold (1996: 3) define IA in the pivotal book Software Change Impact Analysis as: "identifying the potential consequences of a change, or estimating what needs to be modified to accomplish a change". Based on their definition, IA determines the scope of modifications in terms of the design details and can occur throughout the software development lifecycle. By first using IA to indicate the design elements potentially affected by a change, designers then determine the design artefacts that actually require modification.

Other definitions of IA focus on particular phases of the lifecycle. For example, Lindvall (1997a: 4) investigates IA within the requirement development and planning phase. Furthermore, IA is also conceptualised beyond the impact of changes on design details. For example, Pfleeger and Atlee (2006: 526) focus on the risk of making modifications and state that IA is: "the evaluation of the many risks associated with the change, including estimates of effects on resources, effort, and schedule". Consequently, these definitions influence the techniques that can be considered IA[10].

This research project focuses on managing knock-on effects to detail designs. Improving the assessment of changes within the design details is the first step to controlling emergent modifications, which can be aggravated by the influence of resources, effort, and scheduling as well as other effects, such as safety criteria. Thus, the definition of IA by Bohner and Arnold suits this perspective to reduce the risk of emergent changes, and Figure 1.3 illustrates the definition IA for this research project.

Impact Analysis (IA):
"identifying the potential consequences of a change, or estimating what needs to be modified to accomplish a change"
(Bohner and Arnold 1996: 3)

Figure 1.3: Definition of IA used in this research project

Bohner and Arnold (1996: 1) depict IA as an integral part of a change process during new product development and software maintenance. Given that IA can be applied in these different cases, both of these contexts are considered in this research (Section 2.4). Bohner and Arnold (1996: 8) also indicate that designers perform IA in response to a *change request* (also known as an *engineering change* or *engineering change request*

[10] For example, failure modes and effects analysis (FMEA) could be considered as IA by Pfleeger and Atlee's definition and not by Bohner and Arnold's definition.

(Jarratt 2004: 24-26)) to scope the modification that is approved by a *change control board* (Section 2.4.3). While formal change control can occur throughout the software lifecycle, informal change processes also can take place in practice (Sommerville 2007: 500). In turn, this research uses the term *design change* since more specific terms within change processes (*e.g.* change request or engineering change) may limit the types of IA that are considered[11]. The analysis techniques considered as IA by Bohner and Arnold are extended based on this perspective and through a literature review (Section 2.5).

1.5 RESEARCH METHODOLOGY

This project initially identified relevant literature on design change at the systems-software interface (Chapter 2). While this literature review suggests that this interface can be problematic with respect to design change, the literature found often only tangentially discusses the difficulties encountered, as initially suggested in Section 1.1. Furthermore, the points of view in this literature often conflict with each other in terms of how change processes and IA should be performed at this interface. These scarcities and disparities found as well as interest in the general research topic by two industry collaborators indicated that additional inquiry would be useful.

The Design Research Methodology (DRM) proposed by Blessing *et al.* (1995) and further described by Blessing and Chakrabarti (2002) indicates that a researcher should aim to set measurable criteria for success. A first descriptive study follows and investigates current industry practice. The researcher then prescribes a desired scenario through the development of a method or tool. A second descriptive study evaluates the method or tool and ultimately measures the method or tool against the success criteria. Iteration between the descriptive and prescriptive studies to develop the criteria may occur. In turn, these research phases may be covered in more or less depth during each iteration and also depending on the project resources available. Figure 1.4 outlines DRM.

Eckert *et al.* (2003) propose another methodology through their spiral of applied research framework and indicate that design research can begin with questions and that the research objectives subsequently can be derived and tailored from empirical studies. However, the framework stipulates that research may also begin in any of the other 8 types of research and that research may backtrack through these stages or

[11] For instance, design reviews performed independently of a formal change process may not be considered a form of IA (Section 2.5.3), even though they are effective means to detect design flaws. Design flaws or problems, corrected with associated changes, are more fundamental than design mistakes due to human error, which reviews may also find.

progress in parallel or reverse order as necessary. The spiral formed by repeatedly undertaking the 8 research types leads to insights, which sculpt the progression of the research. Figure 1.5 illustrates this research methodology, highlighting the phases implemented in this research project.

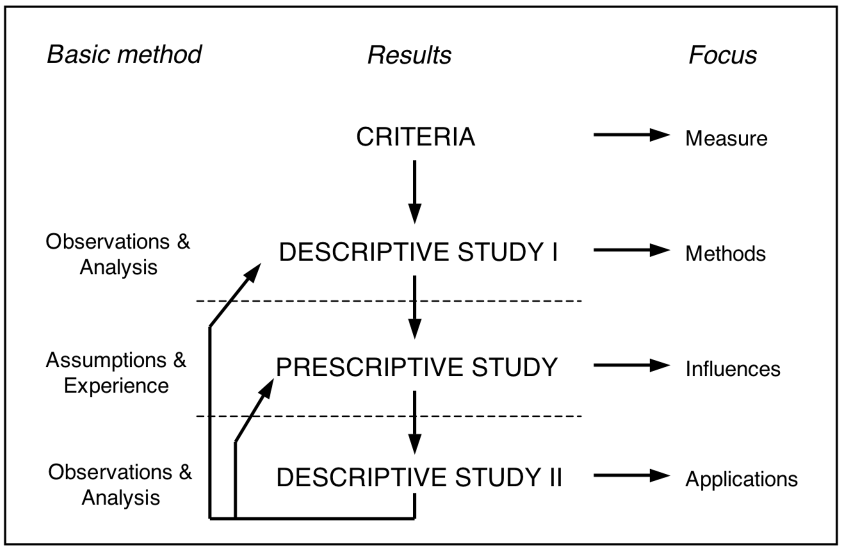


Figure 1.4: DRM (Blessing and Chakrabarti 2002)

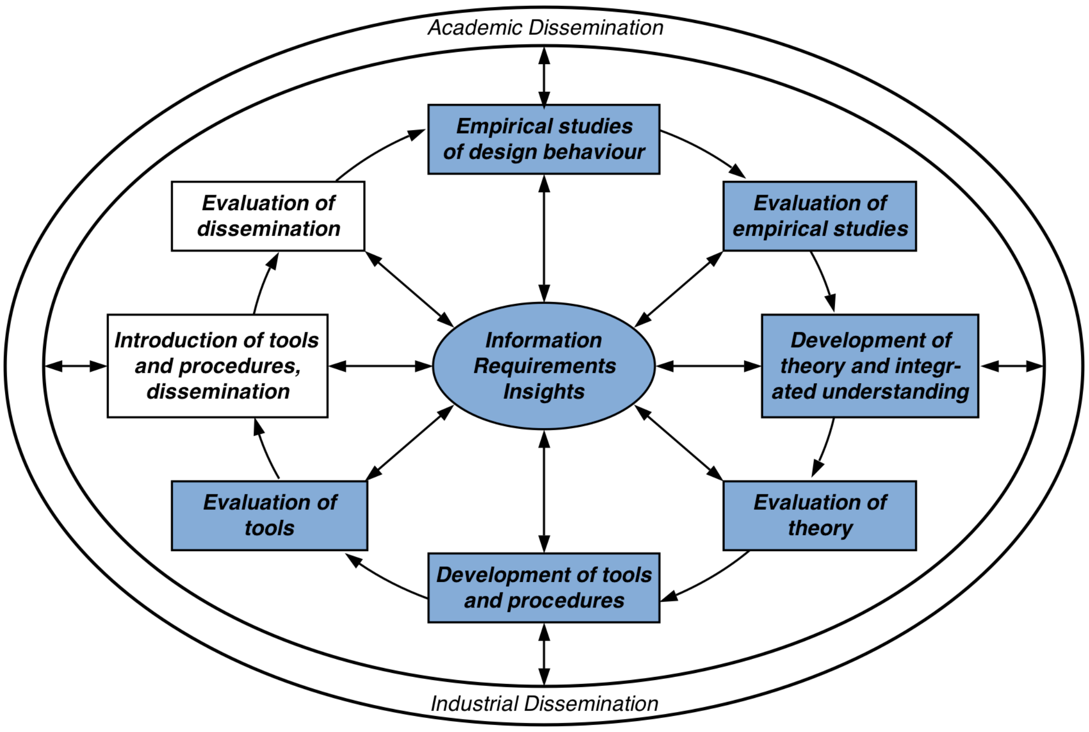

Academic Dissemination
Empirical studies of design behaviour
Evaluation of dissemination
Evaluation of empirical studies
Introduction of tools and procedures, dissemination
Information Requirements Insights
Development of theory and integr-ated understanding
Evaluation of tools
Evaluation of theory
Development of tools and procedures
Industrial Dissemination

Figure 1.5: The spiral of applied research (Eckert *et al.* 2003)

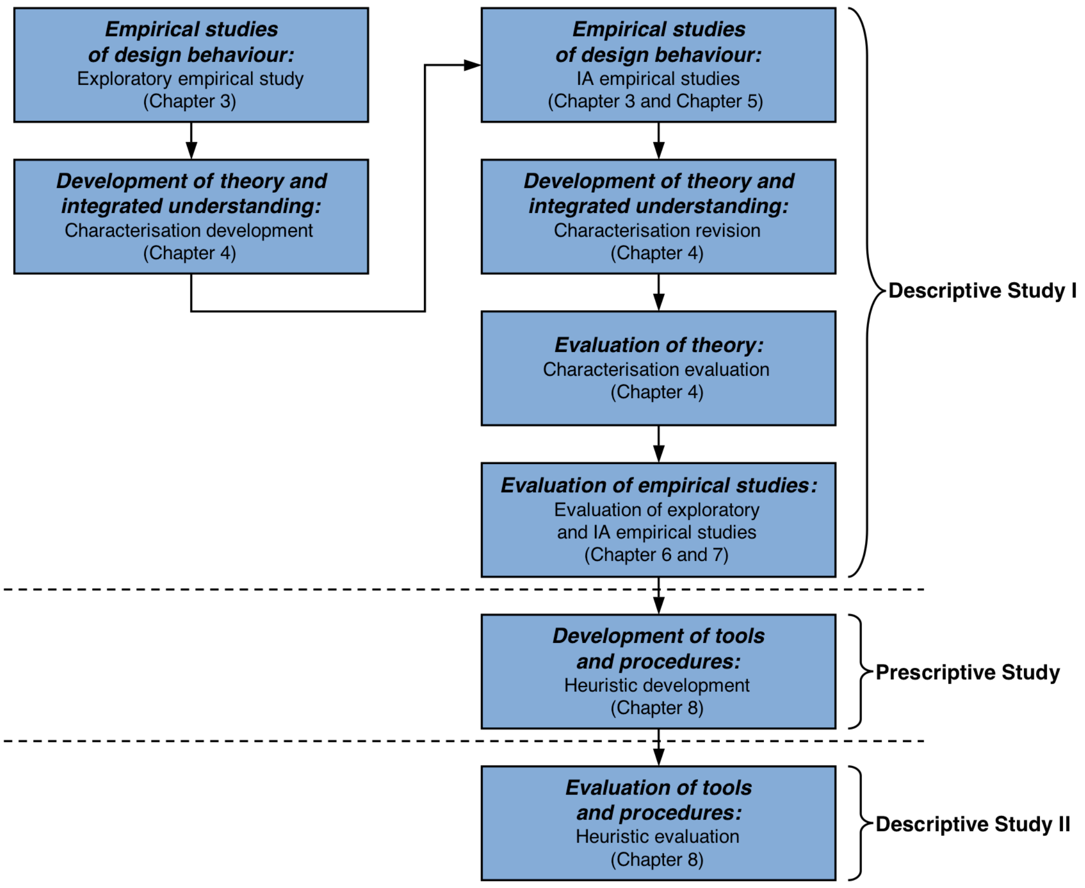

Empirical studies of design behaviour: Exploratory empirical study (Chapter 3)
Development of theory and integrated understanding: Characterisation development (Chapter 4)
Empirical studies of design behaviour: IA empirical studies (Chapter 3 and Chapter 5)
Development of theory and integrated understanding: Characterisation revision (Chapter 4)
Evaluation of theory: Characterisation evaluation (Chapter 4)
Evaluation of empirical studies: Evaluation of exploratory and IA empirical studies (Chapter 6 and 7)
Descriptive Study I
Development of tools and procedures: Heuristic development (Chapter 8)
Prescriptive Study
Evaluation of tools and procedures: Heuristic evaluation (Chapter 8)
Descriptive Study II

Figure 1.6: Phases of research performed

Figure 1.6 details the research phases performed from the spiral of applied research framework and maps them to DRM, showing how both methodologies are satisfied. As shown in Figure 1.6, after an initial literature review identifying the scarcities and disparities in research on IA at the systems-software interface (Chapter 2), this project entered the spiral at *empirical studies of design behaviour* during an exploratory empirical study at an aerospace company. This empirical study aimed to tailor research questions to suit the research gaps identified in the literature review and meet the interests of this industry collaborator. Specifically, the research questions were modified to focus on the systems engineering and software design interface, as opposed to the interaction of mechanical engineering and software engineering, which was also considered at the onset of the empirical study. In turn, an appropriate set of initial research questions was developed, which correspond with the research aims and criteria described by DRM.

After the exploratory study, characterisations describing the nature of IA at the interface were created in the *development of theory and integrated understanding* phase. These two stages of the spiral were repeated by conducting further empirical studies more specifically on IA at the aerospace company and at a telecommunications firm, which led to the refinement of the characterisations. *Information, insights and requirements* were identified throughout these research stages. An extended literature review and reflection by the industry collaborators occurred after this iteration around the spiral and provided a means to assess the characterisations in the *evaluation of theory* phase. In turn, examination of the IA characterisations and observations from the IA empirical studies responded to several of the research questions. Analysis of quantitative data from the aerospace company and simulation work addressed the remaining research questions as well as allowed for the assessment of industry-elicited IA improvement strategies through the *evaluation of empirical studies* stage. These research phases fit into the first descriptive study stage in DRM in that design behaviour was observed and analysed, allowing for development of a support tool.

Specifically, the simulation work was used to create heuristics for improving the application of IA within change processes in the *development of tools and procedures* research phase. This set of heuristics is a tool to guide and improve design practice, corresponding with the prescriptive study in DRM. Finally, the heuristics developed were discussed with some of the key industry collaborators from the empirical studies, providing for the *evaluation of tools* and the second descriptive study in DRM.

1.6 RESEARCH QUESTIONS

In order to contribute to improving the design process, this research project aims to analyse the management of design changes by specifically investigating IA. This research project addresses this high-level objective through the following succession of research questions. The first three questions stem from the gaps found in the literature review (Chapter 2) and are addressed through the empirical studies. The fourth research question examines design process improvement through IA and is tailored in Chapter 5 to confront the challenges of improving IA as elicited from industry.

Understanding change processes at the systems-software interface sets the context for investigating IA, and the literature review reveals different prescriptions and standards for implementing change processes and IA. In turn, the empirical studies (Chapter 3) address how the industry partners account this variety in prescribed change processes and IA techniques through company policies and procedures. Thus, the first research question (Figure 1.7) pertains to how design modifications should be managed using IA within change processes.

How are change processes and impact analysis prescribed at the system and software engineering interface within industry?

Figure 1.7: Research question 1

The second research question (Figure 1.8) focuses on the application of IA in practice. The literature review suggests that industry often does not implement the IA techniques available, increasing the risk for emergent changes. However, no study specifically on IA uptake or research on the collective influence of different IA techniques on managing emergent changes has occurred to date. As such, the empirical studies (Chapter 4) investigate the application of a spectrum of IA techniques in practice.

Does impact analysis influence the management of emergent changes in practice? If so, how?

Figure 1.8: Research question 2

The third research question (Figure 1.9) addresses the challenges influencing the implementation of IA in practice. While the literature review implies that IA techniques are often not applied as prescribed, the specific reasons for this behaviour have not been established yet. The empirical studies reveal the challenges of performing IA in practice (Chapter 4). In turn, understanding the effectiveness of IA on managing design changes and the disparity between prescribed and practised IA can illuminate areas for improvement (Chapter 5).

What are the challenges in using impact analysis to manage emergent changes at the systems and software engineering interface?

Figure 1.9: Research question 3

Finally, the fourth research question (Figure 1.10) indicates the need to understand the relative effect of IA improvement on the overall design process given that change management is only one element of product development. Implementing improvement strategies can incur significant costs and effort, and the potential benefits must be understood by industry to make appropriate management decisions.

Can the application of impact analysis improve the design process? If so, how?

Figure 1.10: Research question 4

1.7 DISSERTATION STRUCTURE

The structure of this dissertation (Figure 1.11) follows the application of the spiral research methodology (Figure 1.6). Chapter 2 summarises the literature review conducted and highlights the change processes and IA techniques prescribed and practised. Areas of scarcity and disparity in literature provide several areas suitable for empirical research. With this basis, Chapter 3 outlines the observations in the empirical studies on the change processes and IA prescribed and practised. In Chapter 4, characterisations of IA synthesise the insights gained in the empirical studies. This theoretical framework addresses the differences between prescribed and practised IA and confronts the influences for the variation. The IA tasks applied in practice and ideas for IA improvement from the aerospace firm are described and analysed in Chapter 5. This discussion leads to the refinement of research question 4 to specifically address how modelling and simulation can show the effects of IA improvement on design processes, setting up further inquiry into IA. Chapter 6 describes the background of system dynamics as a research discipline for process modelling and simulation analysis and details the adaptation of *the rework cycle* (Cooper 1993a) to model IA. In turn, Chapter 7 explains the simulation design and implementation of the adapted rework cycle and evaluates the elicited ideas for IA improvement using this model. Chapter 8 develops heuristics to guide the improvement of IA within change processes based on the simulation work. The industry evaluation of these heuristics is also discussed. Finally, Chapter 9 addresses the research questions, contributions, and future work of this research project.

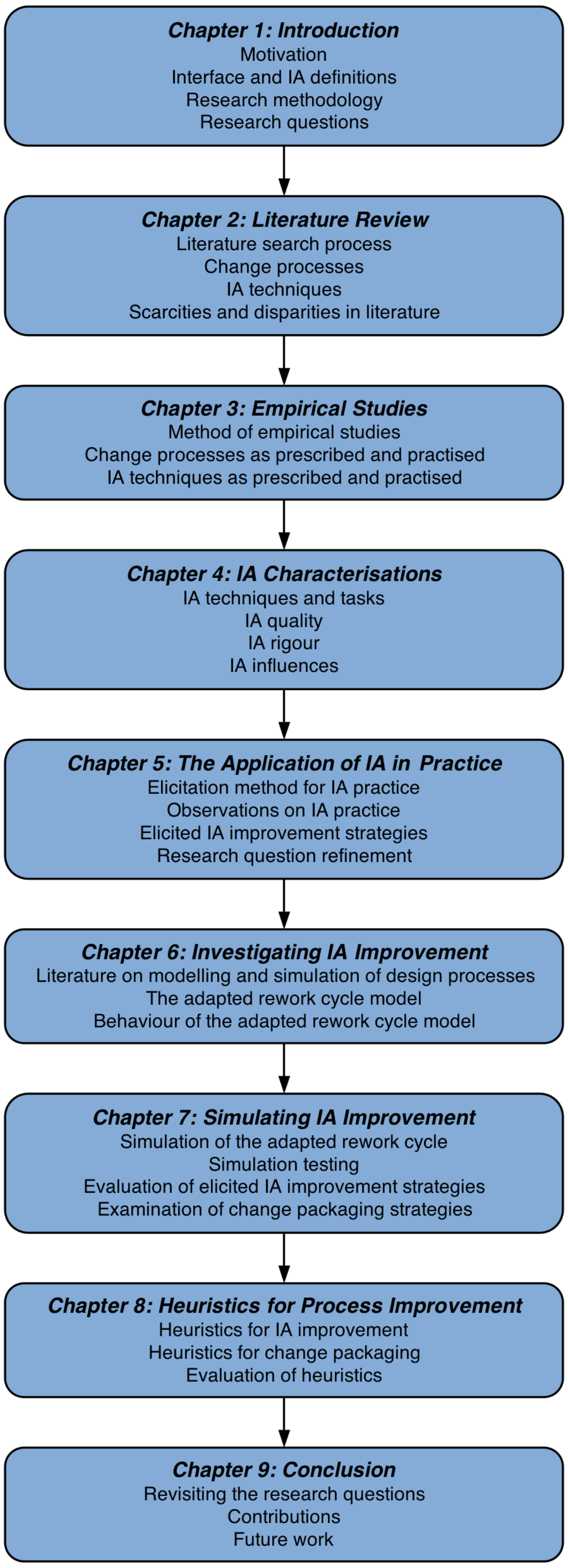

Chapter 1: Introduction
Motivation
Interface and IA definitions
Research methodology
Research questions
Chapter 2: Literature Review
Literature search process
Change processes
IA techniques
Scarcities and disparities in literature
Chapter 3: Empirical Studies
Method of empirical studies
Change processes as prescribed and practised
IA techniques as prescribed and practised
Chapter 4: IA Characterisations
IA techniques and tasks
IA quality
IA rigour
IA influences
Chapter 5: The Application of IA in Practice
Elicitation method for IA practice
Observations on IA practice
Elicited IA improvement strategies
Research question refinement
Chapter 6: Investigating IA Improvement
Literature on modelling and simulation of design processes
The adapted rework cycle model
Behaviour of the adapted rework cycle model
Chapter 7: Simulating IA Improvement
Simulation of the adapted rework cycle
Simulation testing
Evaluation of elicited IA improvement strategies
Examination of change packaging strategies
Chapter 8: Heuristics for Process Improvement
Heuristics for IA improvement
Heuristics for change packaging
Evaluation of heuristics
Chapter 9: Conclusion
Revisiting the research questions
Contributions
Future work

Figure 1.11: Dissertation structure

2 :: Literature on System Design and Software Changes

An analysis of literature sets the basis for the investigation of IA at the systems-software interface. This review extracts specific scarcities and disparities that exist within literature regarding design change processes and IA. These insights depict the need for further inquiry to examine these gaps and disparate perspectives.

2.1 The Literature Search Process

While some literature specifically addresses the systems-software interface, none of this literature found to-date targets change processes or IA to manage design changes (Section 1.1). However, research directed at design change could appear instead within a variety of research areas relating to the systems-software interface. Thus, several *engineering disciplines* that are associated with this interface were investigated. Namely, the literature review explored the fields of systems engineering, software engineering, requirements engineering, mechatronics[12], control system design, and engineering design. More specifically, given the large volume of literature that exists within software engineering, the review focused on plan-driven (*e.g.* waterfall processes) (Boehm 2002), agile, and software maintenance processes within this domain as well as literature on software modelling techniques and software inspection and review methods used for IA.

Although this disciplinary design literature often describes design and change processes, additional literature specifically focusing on *design processes* was also surveyed to determine other perspectives on design change at the systems-software interface. This body of literature includes change management, configuration management, and concurrent engineering. The relationships between the *engineering*

[12] The International Federation for the Theory of Machines and Mechanism (IFTMM) defines mechatronics as "the synergetic integration of mechanical engineering with electronic and intelligent computer control in the design and manufacturing of industrial products and processes" (Comerford 1994).

disciplines and *design process* areas are noted from both literature points of view (Section 2.1.1 and Section 2.1.2, respectively).

Each literature area was explored in books, journals, conference proceedings, and websites for specific concepts on design change, including:

- Type, taxonomy, or characterisation of design changes,
- Prescribed and practised change processes,
- Design processes standards, and
- Prescribed and practised IA.

Bibliographies of references were additionally used to find the most relevant work. These concepts were selected since they encompass investigating the system-software interface in terms of change processes and IA (Section 1.2).

2.1.1 LITERATURE PUBLISHED WITHIN ENGINEERING DISCIPLINES

Systems, software, and requirements engineering literature yields information in characterising design changes, change processes, process standards, and IA. In turn, mechatronics (Bolton 1999; Centinkunt 2007) and control system design (Englander 2000; Gorsline 1986) literature primarily focuses on technical engineering details, but occasionally also refers to design or change processes at the interface, while literature in engineering design contains technical details and also extensive information on design processes. However, within engineering design, extensive work centring on design changes or change processes for products containing software has not been found to date. Table 2.1 maps a representative selection of literature found in mechatronics, control system design, and engineering design to the design change topics examined.

Based on the mappings in Table 2.1, systems, requirements, and software engineering literature provides the most guidance on developing software products. As such, this literature review concentrates on the perspectives from each of these disciplines on the systems-software interface. However, even these areas of research do not fully address IA at this interface. The perspectives from each of these disciplines can often be prescriptive in nature and, specifically, do not fully describe the implementation of IA in practice.

Table 2.1: Mapping of a selection of mechatronics, control system design, and engineering design literature to design change at the systems-software interface

Concept from Literature	References	Mapping to Design Change at the Interface
Mechatronics and control system design literature suggests that mechatronics is a subset of systems engineering.	Bradley *et al.* (2000: 79) separately draw from engineering design and software process models to set the context for mechatronic design and suggest that mechatronics is a subset of systems engineering.	Mechatronics and control system design literature suggest that systems engineering provides a means for implementing multi-disciplinary design and change processes.
	Homkes (2006) indicates that mechatronics occurs within systems engineering processes.	
	Auslander *et al.* (2002: 9) identify systems engineering as a method for coordinating mechanical and software designs.	
Mechatronics literature proposes that this discipline can fulfil concurrent engineering practices.	van Brussel (1996) describes how mechatronics as a discipline provides the technical foundation for concurrent engineering.	Mechatronics literature does not specifically describe appropriate design or change processes as applied within a concurrent engineering framework, but focuses on technical details.
	Comerford (1994) indicates that mechatronics simulation is a resource to incorporate into concurrent engineering in which designers can share results instead of working in parallel without complete understanding of the entire design.	
Engineering design literature prescribes design processes for product development.	Pahl *et al.* (1996: 128-134) as well as French (1998: 2) indicate that the design process for primarily mechanical products involves generating requirements and then developing a conceptual, embodiment, and detail design.	Engineering design tends not to differentiate itself from systems engineering or specifically address interfacing with software design processes. However, research on change management is often published within this discipline (Clarkson *et al.* 2004; Eckert *et al.* 2004).
	Zhang and Li (1999) describe a method to design a mechatronic system by specifically considering the control system.	
Mechatronics, control system design, and engineering design literature indicate that modelling of a system design provides a means to analyse the interface between disciplines.	Jalkio (2006) discusses the importance of requirements modelling such that the partitioning of the design between disciplines can be determined.	Modelling of a system design may also help to understand the scope of changes. However, the literature found does not specifically address IA for design changes.
	Hallin *et al.* (2003) suggest to model information between hardware and software system elements to provide for better information sharing.	
	Gelle (2005) discusses a method to use the Unified Modelling Language (UML) to model system interfaces.	
	Mrozek (2002) describes how UML can be used to design mechatronic systems. He notes that the mechatronic design process is iterative, and the simulation of UML models can aid managing redesign.	
	Kanoun and Ortalo-Borrel (2000) suggest that the interface between hardware and software should be modelled in order to understand system dependability properties.	

2.1.2 LITERATURE FOCUSING ON THE DESIGN PROCESS

Literature on change management, configuration management, and concurrent engineering was also examined to gain insight into design changes the systems-software interface. As previously indicated in Section 1.3.3, this research project has developed from research into change management of technical systems at Cambridge EDC. This investigation of IA specifically draws from research regarding the characterisation of design changes and change processes (Eckert *et al.* 2004) and managing change propagation (Clarkson *et al.* 2004), published within engineering design. Other research lead by Fricke and published within systems engineering mentions software change management, but does not focus on the systems-software interface (Fricke *et al.* 2000; Fricke and Schultz 2005). de Weck is also pursuing further research in change management and has investigated data mining and analysing change propagation in recorded software change data sets from industry (de Weck 2007; Giffin *et al.* 2007). Similar work has focused on correlating (Chmura *et al.* 1990) and visualising (Eick *et al.* 2002) such data sets. However, this research does not account for the principle of change propagation (Section 1.3). Consequently, this research contributes to change management literature, given that no research has specifically explored IA within change processes.

Configuration management literature also contributes to understanding design change at the systems-software interface. Although design standards (Chrissis *et al.* 2007; CMMI 2006; IEEE 2005a; ISO/IEC 2002) prescribe applying configuration management, these standards do not elaborate on the implementation details, as does literature specific to this field. Configuration management literature describes processes for implementing design changes. Furthermore, this discipline discusses the management of requirements, design models, and other forms of documentation used for IA with respect to these change processes (Hass 2003) and software code file management within project teams (White 2000: 184-185). Some of this literature distinguishes the implementation of configuration management depending on the nature of the design process implemented (*e.g.* a sequential waterfall approach versus a more iterative method) (Hass 2003). In effect, configuration management literature does not explicitly address the systems-software interface, yet influences the processes of using of requirements, models, and other documentation as key artefacts of the interface. Thus, configuration management literature provides a perspective into change processes and IA at the interface.

As suggested by Table 2.1, systems engineering offers guidance in managing concurrent design processes across multiple disciplines (Blanchard 2003: 29-40;

ISO/IEC 2002). Similarly, literature specifically focusing on the systems-software interface refers to concurrent engineering (Kaindl 2005). However, these literature areas do not discuss performing multiple changes or instances of IA concurrently within a design process in detail and only prescribe the implementation of concurrent engineering. In turn, concurrent engineering literature occasionally mentions systems engineering (Kusiak 1993: 537), but does not specifically address change processes or IA. Thus, the remainder of this chapter draws from change management and configuration management literature.

2.2 LITERATURE REVIEW DISCUSSION STRUCTURE

This literature review synthesises the perspectives gathered on characterising design changes, change processes, process standards, and IA dispersed throughout the indicated engineering disciplines and design process research. Figure 2.1 illustrates the structure of the discussion of this literature relevant to IA.

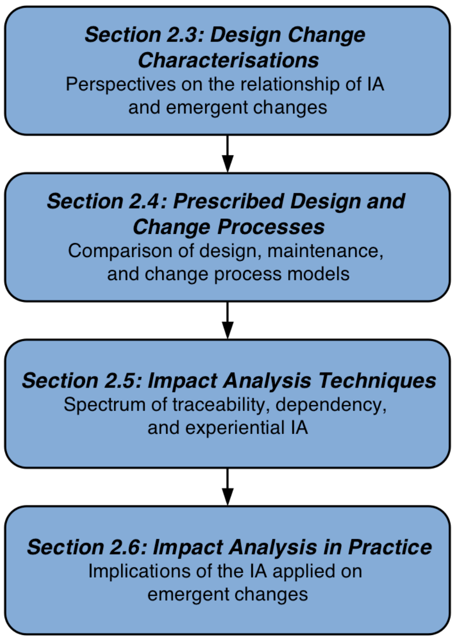

Section 2.3: Design Change Characterisations
Perspectives on the relationship of IA and emergent changes
Section 2.4: Prescribed Design and Change Processes
Comparison of design, maintenance, and change process models
Section 2.5: Impact Analysis Techniques
Spectrum of traceability, dependency, and experiential IA
Section 2.6: Impact Analysis in Practice
Implications of the IA applied on emergent changes

Figure 2.1: Literature review discussion structure

This discussion first focuses on and analyses two characterisations of design changes, one from change management and the other from software engineering, which address emergent changes and IA (Section 2.3). These characterisations suggest a scarcity in research in that they specify a limited selection of IA techniques and only allude to influences that can degrade IA results, leading to emergent changes. Secondly, given that IA occurs in both design and maintenance processes, systems engineering and software design process models during these project phases are examined in conjunction with design process standards and requirements engineering. These design process models are then linked to change process models described within configuration management (Section 2.4). With this basis, different types of IA are described as prescribed (Section 2.5) and practised (Section 2.6). This discussion highlights several conflicts in the literature regarding IA implementation and scant information on how IA is actually used in practice. These insights into these scarcities and disparities are summarised and related to the research questions (Section 2.7).

2.3 DESIGN CHANGE CHARACTERISATIONS

Systems and requirements engineering literature frequently highlights the difficulties in managing requirement changes or implementing late requirements in design processes. These discussions often suggest that making requirement modifications can ripple throughout product designs (Eisner 2002: 236-237; Endres and Rombach 2003: 16) and lead to unexpected, emergent changes (Kossiakoff and Sweet 2003: 267-268). However, no research found to date in these literature areas explicitly defines the characteristics of requirements changes that cause such knock-on modifications during the design or maintenance phases of a product.

In contrast, within change management literature, Eckert *et al.* (2004) explicitly address the characteristics of emergent changes and change processes and conceptualise design change from many different angles. Although this research includes elements specific to mechanical design changes (*e.g.* design manufacturing), the basic characterisation of emergent and initiated changes provides a foundation to distinguish the nature of software modifications during design and maintenance (Section 1.3). The CPM developed from this research (Section 1.3.3), as a form of IA, addresses the need to control the risk of emergent design changes. However, this research into change management only centres on this form of IA and does not describe the range of IA available (Section 2.5) to support controlling emergent changes in mechanical or software engineering. As such, this research project extends this work.

Software engineering and configuration management literature often focuses on change during maintenance and conceptualises design modifications as "corrective", "adaptive", and "perfective" maintenance, as originally proposed by Swanson (1976). Corrective maintenance removes faults; adaptive maintenance allows a software program to operate in another environment; and, perfective maintenance encompasses changes due to user requests. ISO/IEC and IEEE have since adopted this classification of maintenance changes. However, these organisations have published definitions that differ with each other and from Swanson's definitions, causing further research into these change characterisations (Chapin *et al.* 2001). ISO/IEC includes "preventative" maintenance to correct latent design faults, and IEEE uses "emergency" maintenance to describe unscheduled changes to correct faults, allowing for system operation (Canfora and Cimitile 2001: 94-95). Most significantly, Buckley *et al.* (2003) extend Swanson's characterisation and include emergent changes and IA within software design during design and maintenance.

While Eckert *et al.* (2004) describe modifications based on a variety of attributes, such as the source of a change and propagation properties, and then discuss the CPM as IA, the characterisation by Buckley *et al.* (2003) expresses change characteristics in terms of the "mechanism of change", which is defined as the tools or features of tools used to assess and implement the modification. In effect, these tools effectively can perform IA. Buckley *et al.* propose four major themes, "temporal properties", "object of change", "system properties", and "change support", each with underlying dimensions to characterise software changes (Figure 2.2).

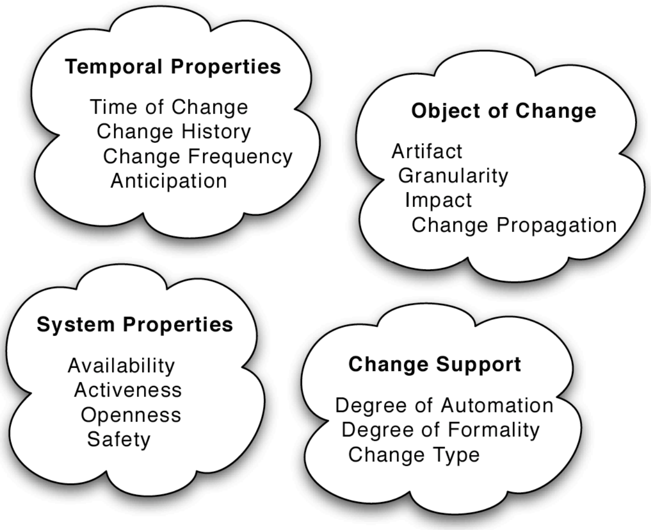


Figure 2.2: A software change taxonomy (Buckley *et al.* 2003)

Of particular interest, Buckley *et al.* define "change history" or the timing of changes in terms of software versioning tools and techniques as used to assess modifications and distinguish between performing synchronous and asynchronous changes. Synchronous changes are described as "convergent" and can be integrated into the tasks of the software design process since versioning tools constantly share all data. In contrast, asynchronous changes may be "divergent" when work commences in parallel with no data sharing, requiring rework tasks for design integration. Emergent changes, as described by Eckert *et al.* (2004), can occur in this latter case and, thus, are related to the task synchronisation within change processes. In effect, Buckley *et al.* suggest that performing IA with complete information can allow for the effective synchronisation of design tasks, minimising knock-on changes.

Furthermore, within the "change propagation" dimension of this characterisation, Buckley *et al.* describe specific tools (*e.g.* refactoring tools[13]) for automating change implementation as the "mechanism of change". They note that these tools perform IA and cite the fundamental IA text by Bohner and Arnold[14] (Section 1.4). However, as defined by Bohner and Arnold and this research project, IA allows for the scoping of possible changes, which may or may not be implemented, while Buckley *et al.* only focus on performing detail design changes automatically. In turn, both reduce the risk of unanticipated change propagation and emergent modifications, but from fundamentally different perspectives. The taxonomy of change by Buckley *et al.* neglects the spectrum of IA techniques available for use.

Comparing the characterisations by Eckert *et al.* and Buckley *et al.* suggests two areas for further inquiry. Firstly, additional process characteristics that cause emergent changes (*e.g.* synchronisation) have not been explored. Secondly, managing emergent changes using the spectrum of IA techniques available has not been addressed. Both of these areas of scarcity are confronted in the research questions (Section 2.7), and the remainder of the literature review concentrates on discussing IA as prescribed by literature and practised within industry.

2.4 PRESCRIBED DESIGN AND CHANGE PROCESSES

Systems and software process models suggest how IA should be performed to assess design changes. Systems engineering process models (Section 2.4.1), supported by design standards and requirements engineering literature, indicate that documenting requirements is a crucial element of the systems-software interface and can be used to execute IA throughout the design process. Some software process models (Section 2.4.2), based on this systems engineering perspective, also describe IA using requirements during design and maintenance. However, agile software design processes do not document requirements as explicitly and, consequently, have an alternative perspective on IA. The range of agile development methods, including

[13] Refactoring tools identify patterns in software code and modify the code structure into a more robust form. The changes made automatically propagate throughout the software code (Section 2.5.2).
[14] Bohner and Arnold (1996) propose a different classification of design changes, including modifications due to incorrect information, clerical errors, errors of omission, inconsistencies, and ambiguities, based on a specific taxonomy used by the U.S. National Aeronautics and Space Administration (NASA). However, this characterisation does not link to IA, emergent changes, or rework effort.

Extreme Programming (XP) and Scrum, generally focuses on iterative development to adapt to changing customer needs. Emphasis is placed on working software rather than supporting design documentation and following rigid development processes (Beedle *et al.* 2001). In turn, configuration management offers change process models (Section 2.4.3) detailing how IA tasks should occur within both requirements-based and agile design methods. As a result, this range of process models describes the context of implementing IA at the systems-software engineering interface.

2.4.1 SYSTEMS ENGINEERING PROCESS MODELS

Systems engineering literature often discusses implementing software design changes through requirements during product development. An adaptation of the Vee model for the systems-software interface clearly illustrates this interaction through requirements (Figure 2.3). This model specifies that systems engineers develop software specifications based on identifying customer and functional requirements. While software engineers should have input during this design phase, they primarily focus on producing the software detail design from the specifications received from systems engineers.

Similarly, other systems engineering process models from the DoD, NASA, and the U.S. Coast Guard depict the flow-down of system requirements to software designers for implementation (Eisner 2002: 236-237). The IEEE 1220 standard also defines a process for requirements engineering of a system design and discusses the transformation of requirements into a detail design solution, as shown in Figure 2.4. Although this standard does not explicitly mention software design and generally recognises subsystem design, it references implementing design processes according to the IEEE standard for software lifecycle processes (Shaaf 2005).

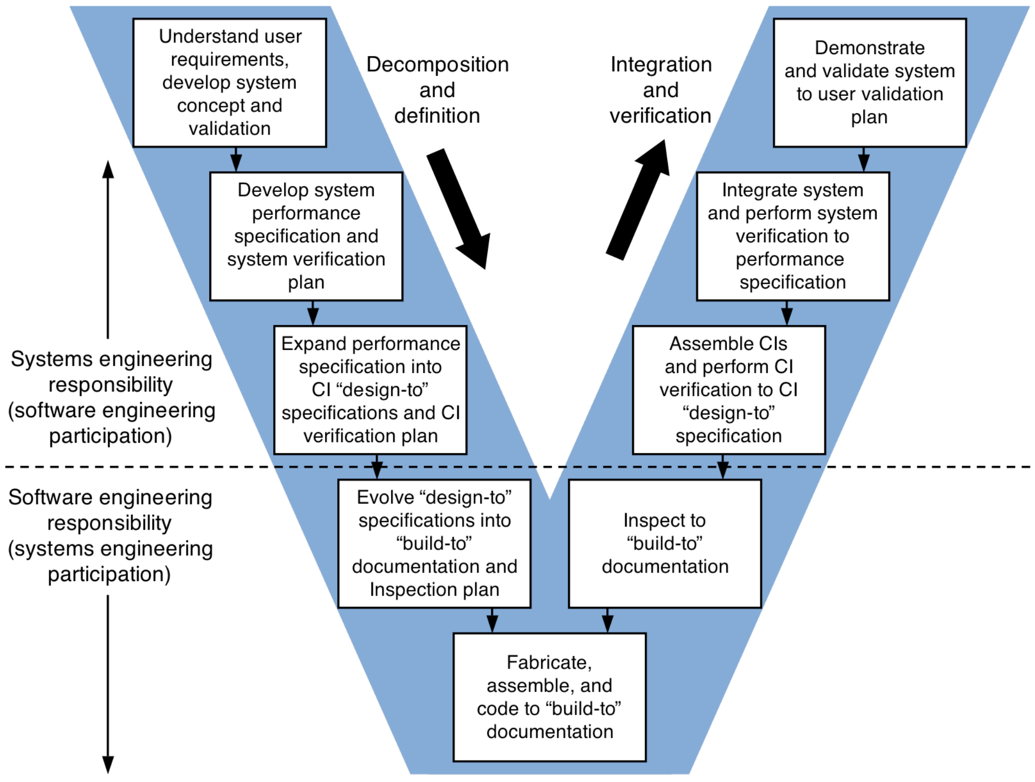

Understand user requirements, develop system concept and validation
Develop system performance specification and system verification plan
Expand performance specification into CI "design-to" specifications and CI verification plan
Evolve "design-to" specifications into "build-to" documentation and Inspection plan
Fabricate, assemble, and code to "build-to" documentation
Inspect to "build-to" documentation
Assemble CIs and perform CI verification to CI "design-to" specification
Integrate system and perform system verification to performance specification
Demonstrate and validate system to user validation plan
Decomposition and definition
Integration and verification
Systems engineering responsibility (software engineering participation)
Software engineering responsibility (systems engineering participation)

Figure 2.3: The Vee model of the systems engineering and software design interface (Blanchard 2003: 27)

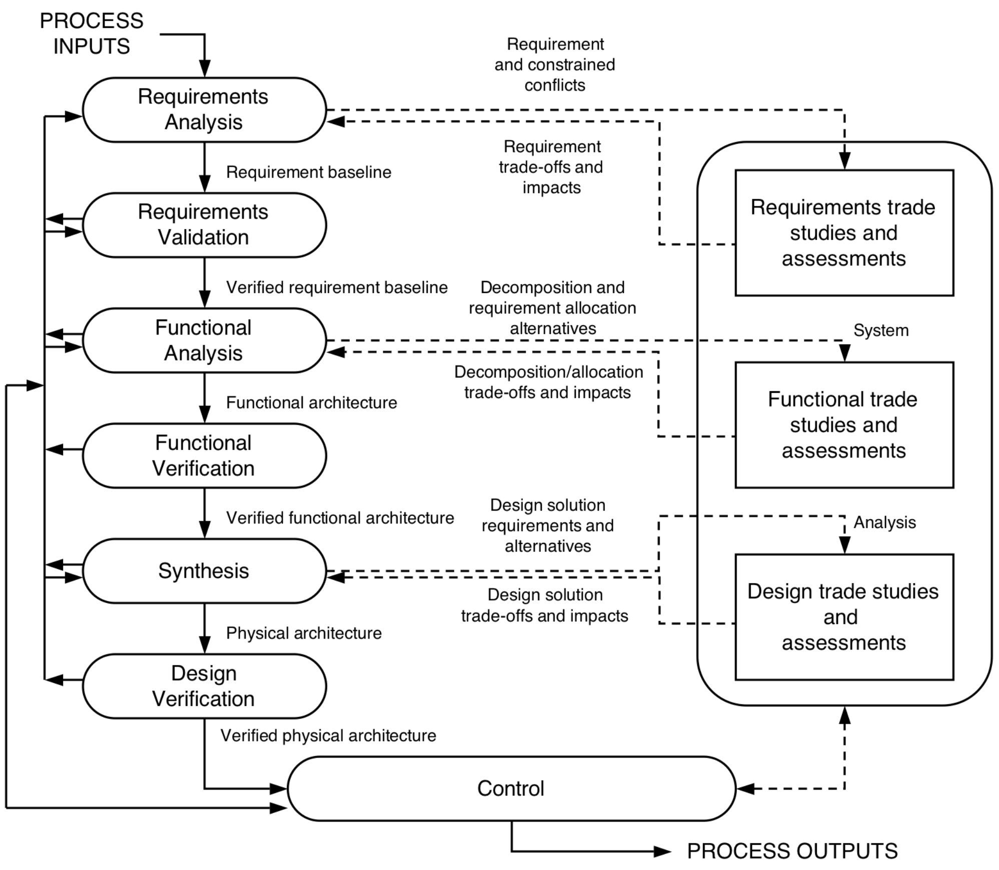

PROCESS INPUTS
Requirements Analysis
Requirement baseline
Requirements Validation
Verified requirement baseline
Functional Analysis
Functional architecture
Functional Verification
Verified functional architecture
Synthesis
Physical architecture
Design Verification
Verified physical architecture
Control
PROCESS OUTPUTS
Requirement and constrained conflicts
Requirement trade-offs and impacts
Decomposition and requirement allocation alternatives
Decomposition/allocation trade-offs and impacts
Design solution requirements and alternatives
Design solution trade-offs and impacts
Requirements trade studies and assessments
System
Functional trade studies and assessments
Analysis
Design trade studies and assessments

Figure 2.4: IEEE 1220 systems engineering process model (IEEE 2005b: 12)

IEEE 1220 aligns with ISO/IEC 15288 in terms of concepts and definitions. However, ISO/IEC 15288 does not graphically model the overall systems engineering design process as does IEEE 1220, but instead describes specific systems engineering sub-processes with defined purposes, inputs, activities, and outcomes. ISO/IEC 15288 covers a broad range of these detail process definitions, such as acquisition and supply, implementation, integration, verification, validation, operation, maintenance, and disposal. In turn, the INCOSE Systems Engineering Handbook also conforms to ISO/IEC 15288 and graphically depicts the systems engineering process through the Vee model (Haskins 2006: 3.3-3.6). CMMI (2006: 396), given the purpose to measure processes, also does not specifically describe systems engineering processes. However, CMMI does suggest that requirements are an interface between systems and software engineering.

Although these systems engineering process models and standards depict different design process details, systems engineering uniformly prescribes requirements as a key artefact of the systems-software interface. As expected, requirements engineering similarly discusses using requirements at the interface of systems engineering and software design (Kotonya and Sommerville 1998: 14). As such, systems engineering, requirements engineering, and design standards often prescribe capturing the relationships between system requirements developed by systems engineers and the related detail software specifications created by software designers through document *traces*. In effect, traces can define the decomposition of requirements across multiple disciplines and particularly link the system and software designs. In turn, the potential consequences of a change to a requirement can be visualised using these captured dependency relationships, known as *traceability IA* (Section 2.5.1). This form of IA can be applied throughout the software lifecycle just as new requirements and requirement modifications can occur. The prescription of working with requirements at the systems-software interface influences the frequent recommendation of using traceability IA within literature (Strens and Sugden 1996). Nevertheless, literature also suggests performing other forms of IA (Section 2.5.2 and Section 2.5.3).

2.4.2 SOFTWARE DESIGN PROCESS MODELS

Some software engineering literature similarly depicts the interface with systems engineering through requirements. Plan-driven design processes, such as the waterfall method and as opposed to agile design processes (Boehm 2002), develop system requirements, which software then details and implements (Figure 2.5).

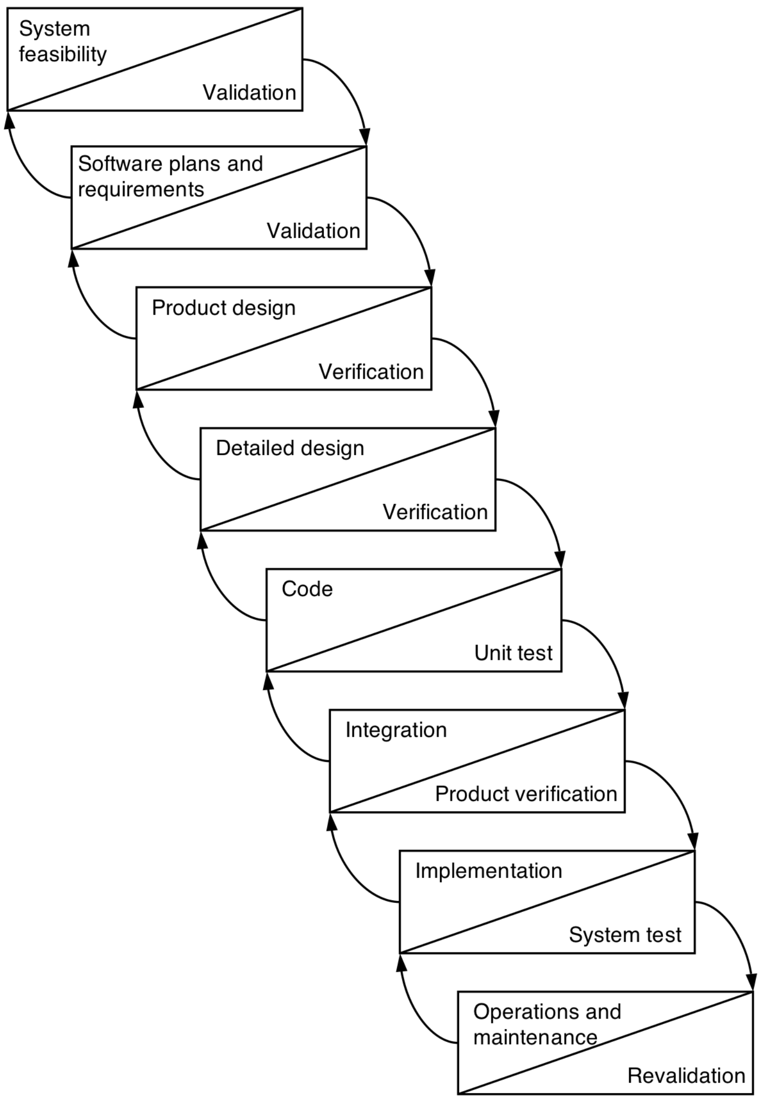


Figure 2.5: The waterfall software design process (Boehm 1988)

Although Boehm and Turner (2005) accept that requirements allow systems engineers to coordinate mechanical and hardware designs, which often operate sequential design processes, with software development, they also argue that software engineering should adopt more iterative and agile design processes. Accordingly, systems and software engineering need to reconcile these different design process approaches. The spiral software lifecycle model (Figure 2.6) and the Rational Unified Process (RUP) suggest such iterative design processes, which include the allocation of system requirements to software design. Given that systems engineering and iterative software design processes include requirement development phases, process planning can synthesise these product development models in theory. As such, Turner (2007) proposes means to incorporate more concurrency and iterative software design into systems engineering. Subsequently, traceability IA can still be implemented through the captured requirement relationships.

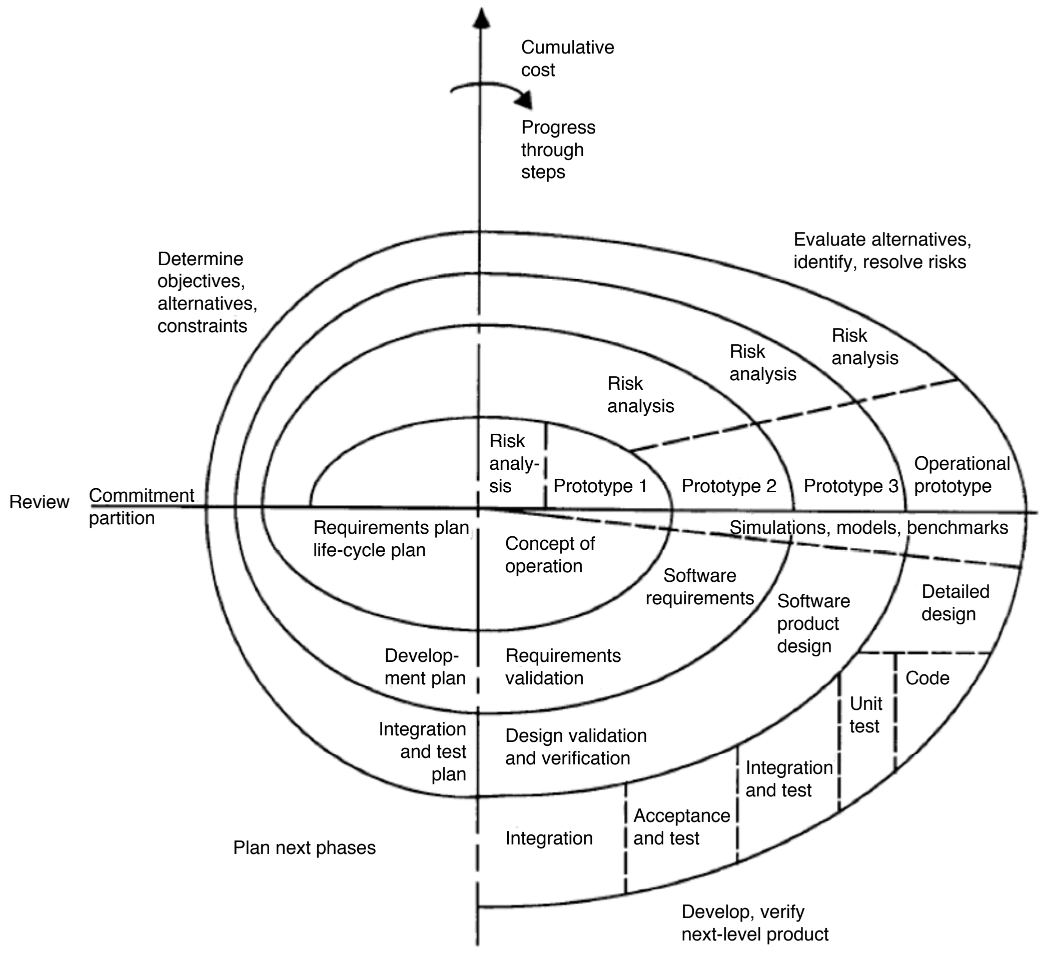


Figure 2.6: The spiral software lifecycle model (Boehm 1988)

Notably, in the spiral model, IA tasks (*e.g.* when traceability IA is applied) are not distinguished for the design iterations depicted, but can be conceptualised based on the change process models presented in Section 2.4.3. These models indicate that IA should occur to scope design modifications prior to making change requests (Figure 2.10) or, in turn, prior to fixing the changes to incorporate into the next prototype iteration (*i.e.* in the *risk analysis* phases in Figure 2.6)[15]. Moreover, IA also should be performed while implementing changes (Figure 2.11) and, consequently, also during the incorporation of modifications analysed through the evolving prototypes into the design (*i.e.* in the development and revision of *software requirements*, *software product design*, and *detailed design* in Figure 2.6).

Agile software development processes, such as Extreme Programming and Scrum, often do not formally write, analyse, or review the complete set of requirements before commencing software design. Davies (2005) describes requirement

[15] As highlighted in the definition of IA for this dissertation (Section 1.4), IA is not considered a type of risk analysis. However, IA fits into this stage of the spiral model since this risk analysis phase drives the changes made to evolve prototype designs in successive iterations.

documentation, analysis, and freezing before design as an "ivory tower". Agile methods expect requirements to change frequently and emerge during development. As a result, requirements may be captured simply in colloquial text within a prioritised list or backlog database or may be part of story cards that describe scenarios of product operation. Thus, agile processes can be more difficult to synchronise with systems engineering, which relies on requirement development and partitioning, and presents another perspective on IA given that requirement traces may not exist (Section 2.5).

Software maintenance literature refers to design work occurring after the release of a product and often conforms to the systems engineering approach through requirements. For instance, Pfleeger and Atlee (2006: 527) specifically name traceability IA (Section 2.5.1) as an essential task to handling such rework and design changes and suggest that requirements are central to controlling maintenance costs. Their maintenance process model also suggests that traceability IA is a means to account for ripples of changes or emergent modifications (Figure 2.7).

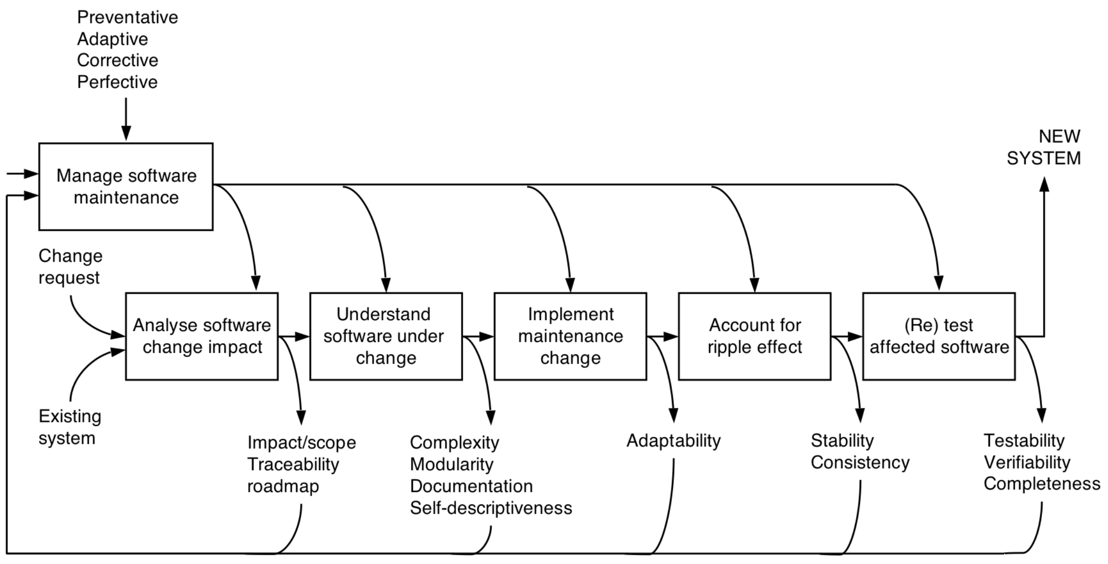


Figure 2.7: A software maintenance model (Pfleeger and Atlee 2006: 527)

Similarly, Sommerville (2007: 499) explicitly identifies IA within his maintenance process model and suggests implementing traceability IA to determine the scope of changes (Figure 2.8).

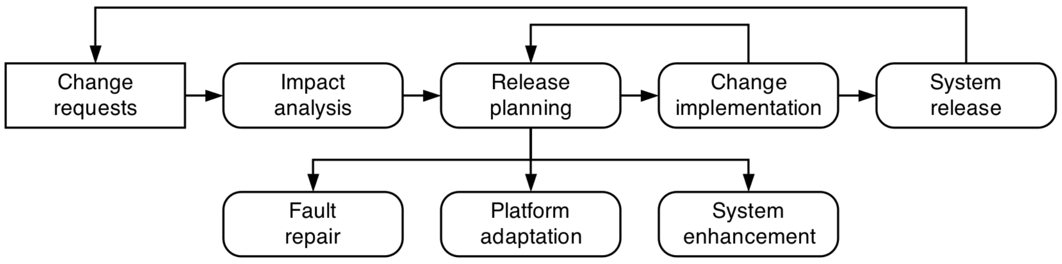


Figure 2.8: A software maintenance model (Sommerville 2007: 499)

Although formal change requests initiate these change processes (Figure 2.7 and Figure 2.8), as configuration management literature suggests should occur (Section 2.4.3), Sommerville (2007: 499) also notes that changes can often occur informally, especially if urgent, and that requirements are rarely consulted or updated in this case. However, he states that many changes made directly to the software code can introduce design inconsistencies, create a chaotic design structure, and cause the software to "age". Ambler (1999: 191) similarly depicts these problems produced by "emergency" changes. In addition, Yourdon (1989: 447) also describes the temptation to make "quick-and-dirty" changes to software code and notes the importance of keeping requirements documentation up-to-date. The updating of supporting design documentation may be neglected in practice since this administrative work is given lower priority than implementing code changes. In turn, van Vliet (2000: 474-475) illustrates this practice through his "quick-fix" model of software maintenance (Figure 2.9) in which code is updated initially and other design artefacts may only be updated later, if time and budget permit.

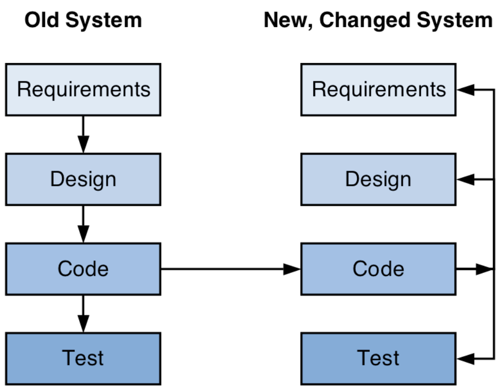


Figure 2.9: The quick-fix model of software maintenance (van Vliet 2000: 474-475)

Using requirements to maintain a system and determine knock-on changes through IA may be impossible, if they are not up-to-date or do not even exist. Ebner and Kaindl (2002) indicate that requirements, design, or design rationale documents rarely exist in general for legacy software. In this case, the quick-fix model may accurately depict the maintenance process performed, focusing on directly using code for IA.

2.4.3 CHANGE PROCESS MODELS

Configuration management literature describes the details of change processes during product development. This literature area often refers to placing a baseline design under formal configuration control. The baseline design contains all design artefacts (*e.g.* requirements, design models, software code, other documentation, etc.) and is used as a reference point for future design modifications (Haskins 2006: 5.12-5.13). In turn, defined change processes handle the incorporation of proposed modifications into a baseline design. Hass (2003) indicates that configuration management change processes can occur within systems engineering or agile development processes and describes archiving practices for this variety of processes accordingly. As a result, the implementation of configuration management processes can influence designers' access to up-to-date information and, therefore, IA techniques and affect the outcome of the IA performed.

Figure 2.10 exemplifies the structure of change processes found within configuration management literature. A *change request* (also known as an *engineering change* or *engineering change request*) initiates a change process by documenting the scope of the design modification and potentially estimating the cost and development time. A *change control board* (CCB), which may include projects managers and stakeholders, accepts or rejects the change request. The CCB may perform IA during this decision process (Whitgift 1991: 135). However, IA inevitably also occurs during the creation of the change request (Leishman and Cook 2003). If a change request is accepted, then the design modification is implemented and verified. While Figure 2.10 illustrates a change process focused on software design modifications, a CCB can oversee modifications for system designs as well.

In practice, change requests also may be revised or trigger alternative requests before acceptance or rejection by a CCB, and, moreover, a CCB may leave change requests open or, in other words, not take action to accept or reject a design modification throughout a design project. Figure 2.10 notably does not illustrate these scenarios.

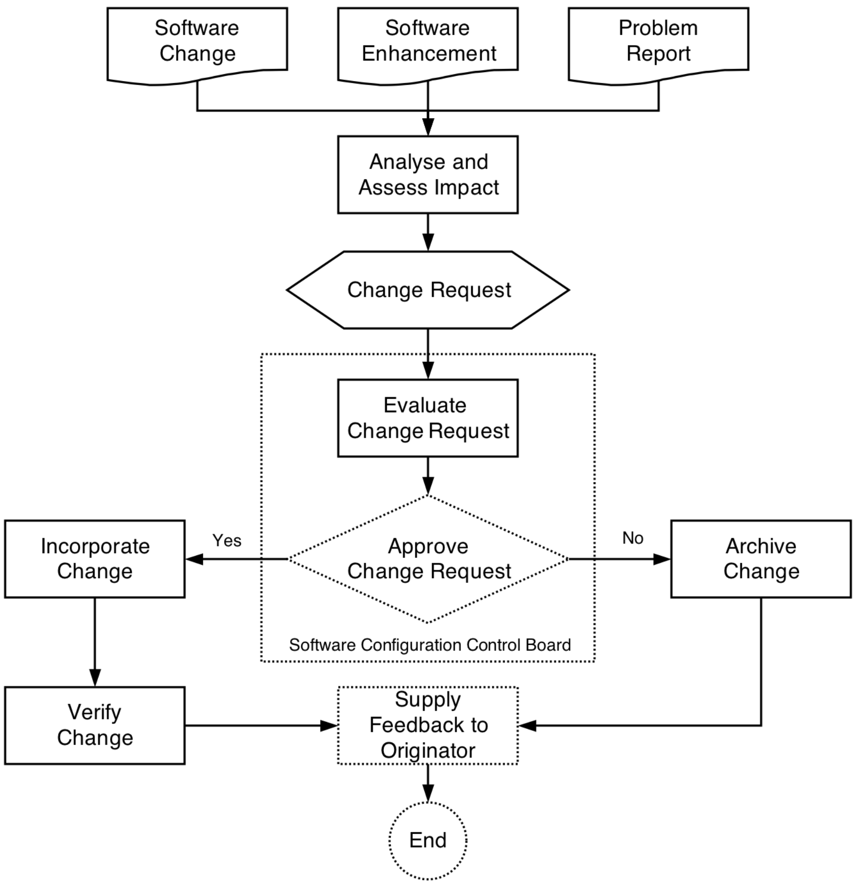


Figure 2.10: A prescribed software change process (Leishman and Cook 2003)

Bohner and Arnold (1996: 7) note that IA also should occur in conjunction with the implementation of the modification to ensure that the full scope of the change is performed, reducing the risk of unexpected, emergent changes. Figure 2.11 models this continual use of IA during change processes.

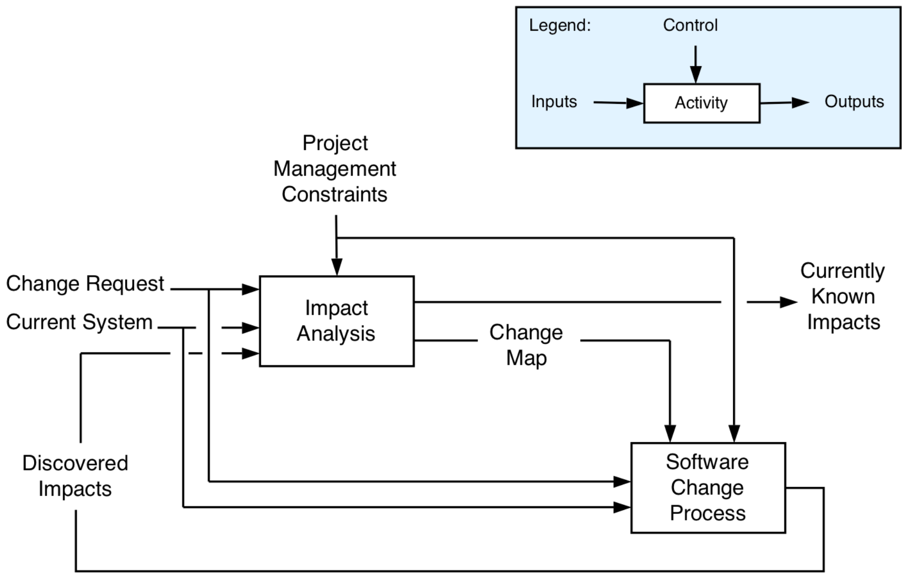


Figure 2.11: Impact analysis within change processes (Bohner and Arnold 1996: 6)

When configuration management change processes take effect during product development, all modifications should be tightly recorded and documented through change requests. However, the delineated point at which such configuration management is implemented can vary from project to project (Haskins 2006: 5.12-5.13). In effect, change processes prior to configuration management can be less formal in that change requests may not be documented or reviewed by managers or stakeholders, and, hence, the times when IA is applied can vary, but should occur, nevertheless.

In summary, the process models reviewed throughout this section indicate a variety of perspectives on how and when IA should be applied during product development and specifically contrast performing IA through a requirement-engineering approach and within agile software development. The following section details the IA techniques used within these processes.

2.5 IMPACT ANALYSIS TECHNIQUES

Systems, requirements, and software engineering have developed a spectrum of IA techniques to perform within more or less formal change processes. Bohner and Arnold (1996) define two classes of IA, *traceability* and *dependency* IA, and this research identifies a third type, *experiential* IA, through a comparison of plan-driven and agile software development literature (Figure 2.12).

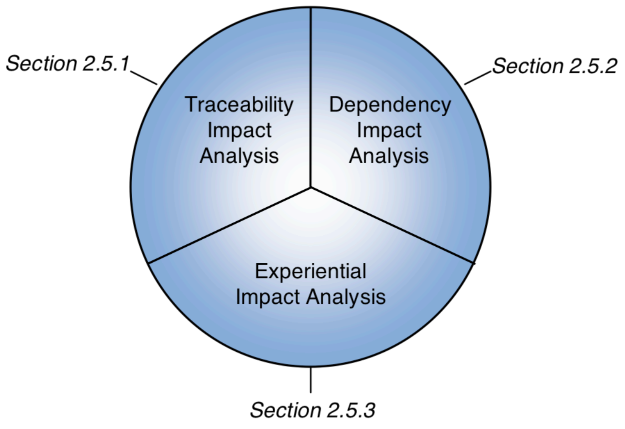

Section 2.5.1
Section 2.5.2
Traceability Impact Analysis
Dependency Impact Analysis
Experiential Impact Analysis
Section 2.5.3

Figure 2.12: Classification of IA techniques

Although any of these IA techniques can be applied as prescribed by change processes (Section 2.4.3), experiential IA may also be performed as defined by design review procedures (Ambler 1998: 488). In effect, design reviews, a form of experiential IA, can initiate change requests and the associated processes, if they detect design flaws or errors.

2.5.1 TRACEABILITY IMPACT ANALYSIS

Systems and software engineering literature as well as design process standards, including IEEE 1220 (IEEE 2005b: 17), ISO/IEC 15288 (ISO/IEC 2002: 21-28), and CMMI (2006: 413), requirements engineering (Hofman 2000: 79; Hull *et al.* 2004: 151), and configuration management (Hass 2003: 100; Whitgift 1991: 150) literature frequently prescribe capturing the relationships between system requirements and software specifications, known as requirement *traceability*. This literature suggests that maintaining these associations in a *requirements traceability matrix* (Haskins 2006: 4.7; Kossiakoff and Sweet 2003: 452) affords completeness in design documentation throughout product development (DeGrace and Stahl 1993: 81-82).

Given a requirement or specification change, the existing traceability relationships extending from the affected element can be identified, indicating additional requirements or specifications that may need modification. In turn, these traces allow engineers to perform *traceability IA* across the systems-software interface (Eisner 2002: 236-237; Nuseibeh and Easterbrook 2000). Assuming that the traceability relationships captured are complete and up-to-date, following these traces can ensure that an initiating design change and its related modifications are implemented fully, reducing the risk for unanticipated, emergent changes (Ambler 2002: 244; Robertson and Robertson 1999: 186). Leveson and Weiss (2004) even argue that traceability is fundamental for managing design changes in safety-critical systems in order to prevent accidents and unanticipated product behaviour.

In contrast to hardware, software changes are deterministic to some extent in that, without implementing modifications fully (*e.g.* variable usage or name changes), software may not compile and run or fail basic system tests. Hardware modifications can be more probabilistic since emergent system properties can occur (*e.g.* vibration), affecting product operation. As such, initiating software changes and the associated *first-order*, knock-on changes can be treated in terms of *completeness* to meet the desired functionality. First-order, knock-on changes are directly related to an initiating design modification, while higher-order, knock-on modifications indirectly occur and may be spawned through interacting requirements or other simultaneous changes. Notably, since software changes can be implemented in different ways to meet the same functionality, completeness only describes the chosen implementation.

While design and maintenance processes highlight requirement traceability (Section 2.4), traces can extend beyond these relationships. Traces can also map system requirements to software detail designs (*i.e.* system requirements are linked to *software lifecycle objects* or *work products*, such as models, code files, or other design documentation) (Bohner and Arnold 1996: 3). *Vertical* traceability[16] refers to the relationships within a type of software lifecycle object (*e.g.* between system requirements), and *horizontal* traceability captures the associations between different types of work products (*e.g.* between system requirements and software specifications) (Pfleeger and Atlee 2006: 526). Figure 2.13 illustrates vertical and horizontal traceability relationships.

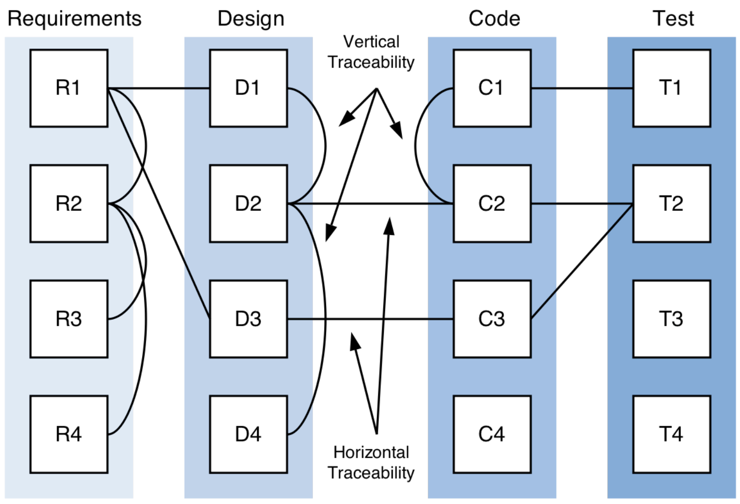


Figure 2.13: Vertical and horizontal traceability (Lindvall and Sandahl 1998)

As a result, traceability IA can indicate the effects of changes on the system requirements, software specifications, and specific software lifecycle objects, which essentially can document the entire systems-software design (Dick 2005).

As described by Bratthall *et al.* (2000), documenting design rationale and inserting traces to this variety of artefacts improves the IA capabilities by encapsulating a wider scope of the change consequences. Similarly, Lacaze *et al.* (2006) have created a tool to support performing IA by tracing rationale to modelled design artefacts; Boehm and Kitapei (2006) also prescribe using traceability links to design rationale in the IA performed during negotiating requirement changes; finally, Bass *et al.* (2006) outline how traces to design rationale can be captured "causally" and "structurally" and feed into IA. Furthermore, Hull *et al.* (2004: 133-134) suggest that additional logic or "rich traceability" can be added to traceability relationships, which are traditionally binary in that they exist or do not exist. By including "and" and "or" logical operators, traceability can be used to show the conditions under which

[16] Literature also provides a variety of other naming conventions for the mapping between artefacts (Blanchard 2003: 84; Grady 2006: 58; Jalote 2000: 58; Lindvall 1997a: 20-21; Sommerville 2001: 142-145).

system requirements satisfy user requirements. By including this additional information, further analyses can be performed using the traceability relationships. For instance, "validation" analysis can ensure that the functional requirements meet the customer requirements (Dick 2005).

Requirements engineering literature also suggests creating specialised traceability databases for design variables, called *data dictionaries* (Bray 2002: 334). Data dictionaries store information about variables used in system design models, software models, or even code. By capturing the relationships between variables and their use in models and code, a change to the name, meaning, or use of a variable can be propagated throughout the design (Sommerville and Sawyer 1997: 181-184).

Although requirements may not be formally documented within agile software development (Section 2.4.2), taking the perspective of Boehm and Turner (2005), merging traditional requirement documentation with other requirement artefacts, such as backlog databases and story cards, should be considered. Daniels and Bahill (2004) exemplify this perspective and suggest a method to integrate requirement and use-case documentation, while Alexander (2002) discusses the implementation of supporting tools for this integration. In turn, using multiple forms of requirement capture and tracing between these work products provides abounding design documentation, which can be used for traceability IA (Jacobson *et al.* 1999: 4).

However, literature on agile development processes does not necessarily support capturing traceability or using this form of IA, arguing that maintaining traces has a high cost and is labour-intensive (Ambler 2002: 244). Kotonya and Sommerville (1998: 132) give an overview of the costs of traceability and write:

> *The fundamental problem with maintaining traceability information is the high cost of collecting, analysing, and maintaining that information. There is a high initial cost involved in assessing dependencies. Additional costs are incurred to update this information every time a requirements change is made. When projects are working to a tight time schedule, other work must sometimes take higher priority. All too often, traceability information is not updated. The information becomes progressively less useful so there is little incentive to use it and keep it up to date. Within a relatively short time, it is discarded and change analysis is carried out informally.*

Although the cost of maintaining traces arguably can be reduced through effective, automated management tools (Grady 2006: 58), traceability relationships, nevertheless, require continual resources during the design process. Palmer (2002) argues that these costs outweigh the risk of avoided rework and the potential for product failure. In turn, Grady (2006: 58) contends that simply relying on human memory (*i.e.* termed experiential IA in Section 2.5.3) cannot effectively determine the consequences of design modifications in the wake of changing program personnel, the length of complex design projects, and the sheer volume of design information and recommends absorbing the costs of traceability IA. In addition, Parnas (1986) directly confronts the reality of evolving requirements and suggests "faking" the requirement development process. He argues that requirements naturally change during design and that not all requirements exist at the beginning of design. As such, requirement documentation should be updated as new requirements emerge and errors are found. The design can be updated rationally as opposed to in an *ad hoc* manner. With this perspective, traceability IA may provide insufficient results at the beginning of the design process and more reliable results towards the end of development (Gilb and Graham 1993: 12; Neill and Laplante 2003). Thus, less incentive also may exist in the short-term for designers to maintain traces, even though it can be useful throughout product development (Grady 2006: 58; Palmer 2002).

To address the costs and potential lack of incentive for maintaining traces, Kotonya and Sommerville (1998: 132-134) suggest creating practical, useable procedures to enable traceability IA. They propose that capturing some traceability information is more beneficial than recording none or some in an *ad hoc* manner. This concept of lightweight traceability forms a middle ground between systems engineering and agile design methods and also initiates another debate on the essential information that should be encapsulated in this case. Neither Kotonya and Sommerville or other literature offer explicit suggestions on such traceability policies.

Many commercial computer tools allow for the semi-automation of traceability IA by structuring the relationships among software lifecycle objects. Within INCOSE's taxonomy of systems engineering tools website, 22 traceability tools are listed (INCOSE 2004a), such as Borland Caliber-RM™, Telelogic DOORS®, and IBM Rational® RequisitePro®. IA results produced by these tools may still require manual analysis since modifications may not propagate across all traceability relationships, as described by Bohner and Arnold (1996: 8). Research continues to aim to improve the results of these tools to virtually eliminate this work by hand through enhancing the relationship modelling (Ebner and Kaindl 2002; von Knethen 2002), the visualisation of traceability links (Marcus *et al.* 2005), the traceability IA algorithms (Lock and Kotonya 1999; O'Neal 2003; O'Neal and Carver 2001), and the automation of maintaining traces (Hayes *et al.* 2005; Marcus *et al.* 2005). However, even without tool automation, traceability IA can be performed manually by searching for and following references between work products (Brown *et al.* 1999: 98), although this search process may be difficult to do exhaustively (Dick 2005).

2.5.2 DEPENDENCY IMPACT ANALYSIS

Although systems, requirements, and software engineering literature emphasises requirements as a primary work product at the interface, *dependency IA* techniques can also assess design modifications. Bohner and Arnold (1996: 2) describe performing dependency IA by identifying the linkages between variables, logic, modules, etc. within software architecture or code. If a design change from systems engineering is known to affect a specific software artefact, dependency IA, performed using the actual software code, can depict additional areas that also require modification. Bohner and Arnold recognise that dependency IA is limited to low-level design, compared to traceability IA, which broadly analyses relationships between work products. As such, dependency IA does not indicate further changes required of requirements or other work products. For instance, dependency IA may not be able to diagnose the timing and synchronisation of software functions, requiring a high-level analysis of the distribution of software requirements and timing constraints. Nevertheless, traceability IA can be used in conjunction with dependency IA for more detailed analysis of software designs (Rierson 2000).

Program slicing and *refactoring* are examples of such low-level, code-based, dependency IA. In program slicing, the dependencies of a variable at a certain point in a program are determined. This *slice* or connections within the program include all of the statements in the program that might affect this variable. Thus, the potential set of knock-on changes to this variable can be assessed (Binkley *et al.* 2007; Horwitz *et al.* 1990; Sridharan *et al.* 2007; Weiser 1984). Based on the concept of program slicing, more specific forms of this dependency IA have stemmed. Static algorithms identify relationships within the program structure (Buckner *et al.* 2005; Han 1997; Kung *et al.* 1994; Rajlich 1997; Rajlich 2000; Rajlich and Gosavi 2004; van den Berg 2006); dynamic methods gather information about program behaviour at run-time (Apiwattanapong *et al.* 2005; Breech *et al.* 2005; Law and Rothermel 2003); and, hybrid algorithms incorporate both static and dynamic characteristics (Ren *et al.* 2006; Ren *et al.* 2005). Similar support for dependency IA has been based on algorithms using probabilities (Tsantalis *et al.* 2005), heuristics (Hassan and Holt 2004), and other coupling measures (Briand *et al.* 1999). Unlike these dependency IA algorithms, refactoring often automatically changes the software code and is frequently used within Extreme Programming. Refactoring tools make modifications to the internal code structure and propagate these changes without modifying the external code behaviour (Fowler 2004). In turn, the changes initiated are often restricted to follow certain patterns and focus on producing more robust code. IBM Rational® RequisitePro® and Metallect IQ (Figure 2.14) are two commercial software tools available for dependency IA that use such analysis algorithms to map relationships within software code.

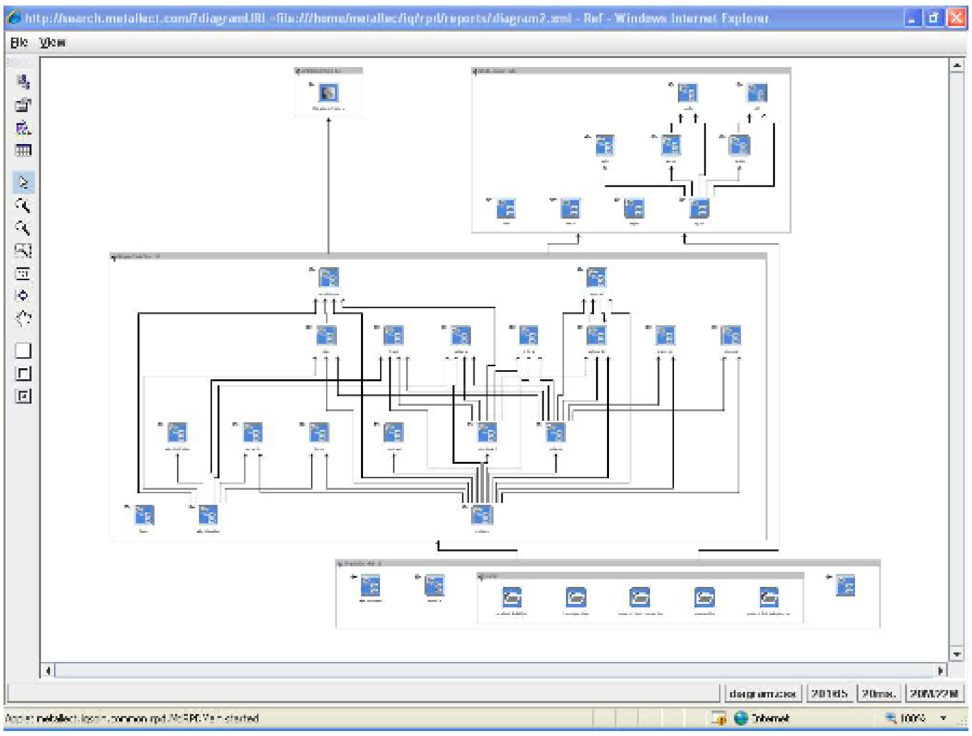

http://search.metallect.com/?diagramURL=file:///home/metallect/iq/rpd/reports/diagram2.xml - Ref - Windows Internet Explorer
File View
Internet
100%

Figure 2.14: Screenshot of code dependencies from Metallect IQ (Metallect 2007)

Extending Bohner and Arnold's definition of dependency IA, literature also describes techniques implemented through system and software design models. Models can map the dependencies between model elements and be used to scope a design change. The impact of a change identified on the model components can be translated into the consequences on the actual design. Specifically, UML models[17] can visualise system or software requirements in terms of the details of a software design (Holt 2001: 235-240), and specialised algorithms have been developed to perform IA on UML diagrams (Briand *et al.* 2003; Briand *et al.* 2005; Lee *et al.* 2002). Other modelling formats can be used in a similar manner to perform IA (Ajila 1995). Such analysis of models can be considered dependency IA in that modelled relationships depicted are often more detailed than those used for traceability IA and

[17] INCOSE has extended UML for systems engineering modelling capabilities and created the Systems Modelling Language (SysML), allowing for the clear depiction of system requirements for software design (Haskins 2006: 7.7-7.8). Furthermore, SysML can support integrated software and hardware designs, granting IA to be performed across these disciplines. Advancing the techniques for interdisciplinary IA could provide a means to avoid accidents, such as the Therac-25 discussed in the introduction to this dissertation.

can be animated, actively changing software code in some cases. UML modelling programs, such as ARTiSAN Studio®, support such dependency IA, and, since systems engineering and agile development methods promote using UML, both can implement such dependency IA (Ambler 2002).

Although the specific dependency IA techniques described thus far often can rely on automated algorithms, dependency IA can also be performed through manual searches of variables or logic in software models or code. Systematically following dependencies can indicate the consequences of making a change. van Vliet (2000: 459) suggests that a top-down study of program text can effectively identify knock-on effects of changes and notes that only localised study of variables and program logic can cause unanticipated problems. The code should be first analysed at a high-level for dependencies between code modules, functions, etc. and then investigated in progressively more detail until low-level relationships between specific variables within the program logic are reached. However, similarly to employing traceability IA by hand, manual searches can be difficult to implement thoroughly for large, detailed software designs.

2.5.3 EXPERIENTIAL IMPACT ANALYSIS

Literature on IA often refers to using alternative techniques to aid in understanding the consequences of changes. For example, Bohner and Arnold (1996: 7) write that if design relationships are not or cannot be captured for traceability or dependency IA (*e.g.* early in design processes), then designers can use review techniques, such as inspections, walkthroughs, and desk checking, to identify requirement and variable relationships and, in turn, infer the impact of changes. While Bohner and Arnold do not classify such techniques as IA, literature supports the argument that reviews[18] can identify defects and potential problems in design, which necessitate modifications. van Vliet (2000: 87) indicates that expert design knowledge employed during reviews quickly and efficiently detects errors. Similarly, Endres (2003: 17) states that reviewing also can cheaply spot changes early, as opposed to detecting design errors late and then making design modifications at a high cost. In addition, Humphrey (2000: 150) argues that reviews are essential because it is impossible to exhaustively test software systems to find all errors and make appropriate changes. Furthermore, testing should not be assumed to be able to find all design errors (*e.g.* high-level design flaws, such as not meeting user needs, may not be found during

[18] Bohner and Arnold do not appear to distinguish between these different reviewing techniques. While walkthroughs, scenarios, inspections, peer reviews, and the like have different motivations and methodologies (Hofman 2000: 70-77), the generic term *review* is used in this research for simplification, as does Ciolkowski (2003).

low-level testing) (Ellims *et al.* 2006). Designers must use their logical reasoning to interpret the impact of changes in these cases. Similarly, traceability and dependency IA techniques may require designer interpretation to determine the relevant results and consequences of changes, as suggested by Bohner and Arnold's definition of IA (Section 1.4) and as indicated by research into software "information leaks" (George and Bohner 2004). Hence, IA can involve using engineering judgement and knowledge.

Although review methodologies attempt to support the systematic evaluation of designs through assigned roles, checklists, and reviewing rules and processes (Ambler 2002: 473; Bush 2004; Gilb and Graham 1993: 43-62), reviews still depend on designer knowledge and experience to identify design errors and the associated knock-on changes. As such, engineers can perform IA by simply thinking about the implications of a change and using "general knowledge" (Sommerville 2001: 145) or informally discussing proposed modifications within the design team as done in agile processes. Applying engineering judgement, just as formal reviews, can identify tacit dependencies and implications of design changes not found through traceability or dependency IA techniques. Furthermore, capturing and then reviewing tacit relationships between design artefacts through design rationale can enhance this judgement (Boehm and Kitapei 2006). Using such design rationale, which may also be captured for traceability IA (Section 2.5.1), can enhance both techniques. Ambler (2002: 244) specifically argues for using engineering judgement instead of traceability IA and writes:

> *Warning: Think Very Carefully Before Investing in a Requirements Traceability Matrix…This is not travelling light… The benefits of having such a matrix is that it makes it easier to perform an impact analysis that pertains to a changed requirement because you know what aspects of your system will potentially be affected by the change. However, if you have one or more people familiar with the system, which you want to have anyway, and if you want to be effective at enhancing the system, then it is much easier and cheaper to simply ask them to estimate the change. Traceability matrices are highly overrated because the cost to maintain such matrices, even if you have specific tools to do so, typically far outweigh the benefits.*

Given that agile design methods focus on working software and emphasise communication, rather than documentation, they may prefer implementing IA through informal discussions and individual engineering judgement.

In turn, this research defines *experiential IA* to include review processes, informal discussions, and the application of engineering judgement. Experiential IA can identify implicit design dependencies and, therefore, mechanisms for change propagation through expert knowledge. However, this form of IA may be unsystematic by nature, potentially neglecting means of change propagation.

Nevertheless, performing experiential IA does not exclude the implementation of traceability and dependency IA, and vice versa.

To summarise the specific IA techniques cited in this section, Figure 2.15 provides example references for the traceability, dependency and experiential classification (Figure 2.12). The effort to implement each type of IA is further investigated through modelling and simulation in Chapters 6, 7, and 8 using the adapted rework cycle model (Figure 8.2). This work focuses on analysing the completeness of IA for design modifications and the associated *first-order*, knock-on changes (Section 2.5.1 and Section 6.3.2).

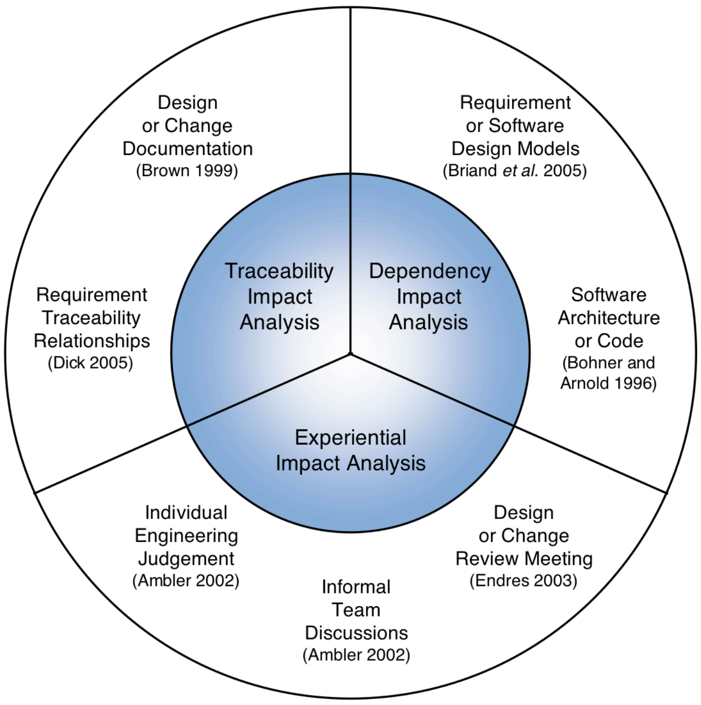

Design or Change Documentation (Brown 1999)
Requirement or Software Design Models (Briand et al. 2005)
Traceability Impact Analysis
Dependency Impact Analysis
Requirement Traceability Relationships (Dick 2005)
Software Architecture or Code (Bohner and Arnold 1996)
Experiential Impact Analysis
Individual Engineering Judgement (Ambler 2002)
Design or Change Review Meeting (Endres 2003)
Informal Team Discussions (Ambler 2002)

Figure 2.15: The range of IA techniques

2.6 IMPACT ANALYSIS IN PRACTICE

Even though literature promotes traceability, dependency, and experiential IA as effective means to identify the scope of changes (Section 2.5), literature also suggests that these IA techniques are not always implemented in practice. The following discussion highlights the disparities between the prescribed and practised forms of traceability (Section 2.6.1), dependency (Section 2.6.2), and experiential (Section 2.6.3) IA. Attributes of practised change processes, such as documentation and model maintenance, review processes, and only relying on memory and experience, can cause these IA techniques to deliver less than perfect results, increasing the risk of unanticipated, emergent changes.

2.6.1 TRACEABILITY IMPACT ANALYSIS IN PRACTICE

Literature suggests that requirement management and traceability maintenance processes can cause the disuse of traceability IA in practice. Frequent or late requirement changes particularly produce this effect (Blanchard 2003: 4; Humphrey 2000: 111; Jalote 2000: 56). If changes occur frequently, designers may not keep traceability information always up-to-date. If late requirements emerge, traces may not be maintained during the implementation of emergency or quick-fix change processes (Section 2.4.2). Similarly, if requirements are inconsistent or incomplete dependency traces may not be reliable (Heimdahl and Leveson 1996; Sage 1990: 240). In turn, the traceability IA performed with such inaccurate information does not provide useful results, potentially leading to abandonment of this technique.

Although some literature indicates that requirement management improvement can occur in practice, increasing productivity (Damian *et al.* 2005), other literature confirms that frequent and late requirement changes continue to plague design projects (Emam and Hoeltje 1997; Jones 2006). In a special issue of *IEEE Software* devoted to large-scale studies on software engineering practice, research indicates that documentation is not maintained (Lethbridge *et al.* 2003) and traceability between work products is not always captured during design processes (Graaf *et al.* 2003). While out-of-date requirements and traces can occur due to the logistics of tool configuration and the sharing of information (Kotonya and Sommerville 1998: 134), these large-scale studies indicate that this is not the case since the information is simply not maintained consistently. In effect, the IA results obtained can be out-of-date, at best, and without the capture of traces, the analysis cannot even be performed in practice.

In addition, literature suggests that maintaining documentation and traceability can be challenging based on the nature of the design process implemented. For example, Rottman (2006) notes that requirement management is particularly difficult when outsourcing software design. Coordinating information between dispersed teams developing system requirements and programming the software affects the handling of requirements and changes. As opposed to "throwing requirements over a fence", responsibility and participation in IA is required of both parties. However, even for co-located design teams, highly parallel design processes can cause problems and design errors (Perry *et al.* 2001). Specifically, concurrent requirement changes can lead to out-of-date, inaccurate traces and unhelpful IA results (Buckley *et al.* 2003). Thus, the context of the design process (*i.e.* outsourcing or high concurrency) can influence the implementation of traceability IA in practice.

2.6.2 DEPENDENCY IMPACT ANALYSIS IN PRACTICE

The implementation of dependency IA can face similar challenges to traceability IA, even though most of the literature on dependency IA found focuses on algorithm development. While dependency IA may be difficult to implement based on the context of the design process (*i.e.* outsourcing or high concurrency) since dependency models have to be maintained, as do traceability relationships, no literature has been found to specifically address this concern. Nevertheless, literature suggests that the lack of model maintenance has lead to erroneous IA results in practice. In particular, literature indicates that performing IA on incomplete and inconsistent models can produce misleading results. For example, Pap *et al.* (2001) discuss the importance of creating complete and consistent UML models for design specifications and cite incidences in which ambiguous specifications have caused software failures. In turn, Pap *et al.* develop methods to check UML diagrams for these properties. Graff *et al.* (2003) similarly report the inconsistency of UML models in practice during a large-scale study of the software engineering industry, potentially causing inaccurate IA results. Furthermore, incompleteness and inconsistencies within software design and code can occur independently (Nuseibeh *et al.* 2000), and, consequently, dependency IA results either based on models or code can both be incorrect and unhelpful.

2.6.3 EXPERIENTIAL IMPACT ANALYSIS IN PRACTICE

Finally, literature notes that experiential IA is not necessarily implemented as intended. Ciolkowski *et al.* (2003) report from a large-scale study on software engineering practice that reviews are often performed using unsystematic techniques and do not specifically look for design faults and the associated necessary changes. Using reviews only to educate others on a design or to follow procedures misses the opportunity to evaluate the design quality. Ciolkowski *et al.* also indicate that standard processes for reviews often do not exist. In turn, the results produced by reviews can vary greatly, making this form of experiential IA unreliable to detect knock-on changes. Similarly, Weinberg (2003) and Chao *et al.* (2004) separately acknowledge that reviews are often not rigorously implemented within industry. They indicate that cosmetic reviews occur in that there is no real intent to find defects, and the reviews simply provide a false sense of security that design progress has been made. In contrast, Bush (2004) notes that review processes have been both fervently adopted as well as abandoned in companies. She attributes these differences to organisational culture. Consequently, there is no overwhelming evidence that reviews are a key form of IA used in practice.

Studies have also shown that the reliance on experiential IA techniques using engineering knowledge and judgement produces inaccurate estimations of change. Specifically, Lindvall and Sandahl (1998) experimentally demonstrate that engineers predict incomplete sets of changes when only using their judgement, producing an ineffective means of IA. Similarly, von Mayrhauser *et al.* (1997) report on observations that changing software code can cause memory overloading for designers. In these experiments conducted, designers ended up relying on provided tool support to obtain information then implementing changes. Thus, the implementation of experiential IA is not necessarily a proven method for detecting changes in practice.

2.7 SCARCITIES AND DISPARITIES IN IMPACT ANALYSIS LITERATURE

In this chapter, several scarcities and disparities within literature on IA have been illuminated. Firstly, the search process (Section 2.1) indicates that the systems-software interface is treated only tangentially within disciplinary and design process literature, and no abundance of research focuses on the nature of this interface (Section 1.1). Of the literature found specific to the interaction of systems engineering and software design, no research concentrates on change management, even though much of this literature mentions the difficulty of handling design modifications. As a result, this research project as a whole addresses this general lack of research.

Secondly, the review of change characterisations suggests a potential scarcity of research at the interface in that only one process characteristic, synchronisation, is mentioned to influence the management of emergent changes using IA (Section 2.3). Although other literature on traceability IA practice corresponds to this characterisation in that outsourcing and highly concurrent processes can affect the sharing of information and application of IA (Section 2.6.1), other influences may also exist and have yet been explicitly defined. In effect, the reasons for the differences between the prescribed and practised forms of IA have not been characterised. Research question 3 (Figure 1.9) refers to this concern of scarcity in literature, and the empirical studies describe the influences on implementing IA in practice (Chapter 4).

Thirdly, in the discussion of design processes (Section 2.4), systems engineering processes and the standards associated with these processes (*e.g.* ISO/IEC 15288, IEEE 1220, and CMMI) conflict with literature on agile software development methods (Boehm and Turner 2005). These different perspectives have not been fully interpreted in terms of implementing IA. The discussions on traceability (Section 2.5.1) and experiential (Section 2.5.3) IA highlight this discrepancy, but often the literature cited only prescribes or devalues certain IA techniques from a theoretical, academic perspective. Research question 1 (Figure 1.7) is derived from examining these disparities in literature and noting that no other research found to date has examined these different perceptions of IA within industry. In turn, the empirical studies investigate the prescription of IA by company policies and procedures from systems engineering and agile methods perspectives (Chapter 3).

Fourthly, literature suggests that prescribed IA techniques are not always implemented in practice (Section 2.6). For instance, research on requirement and model maintenance practice suggests that traceability and dependency IA may not produce useful results and, in turn, not be applied in practice. Similarly, literature on experiential IA shows that the adoption of design reviews for IA varies within industry. However, no research found identifies the spectrum of IA available and succinctly describes the difference between these IA techniques as prescribed and practised. Thus, a scarcity on the influence of IA on emergent design changes in practice appears to exist within literature, as described in research question 2 (Figure 1.8). The empirical studies do not aim to perform a large cross-sectional study across many companies and industry sectors to answer this question. They intend to provide a characterisation and theoretical argument to depict the influences of different types of IA on emergent change management in practice (Chapter 4).

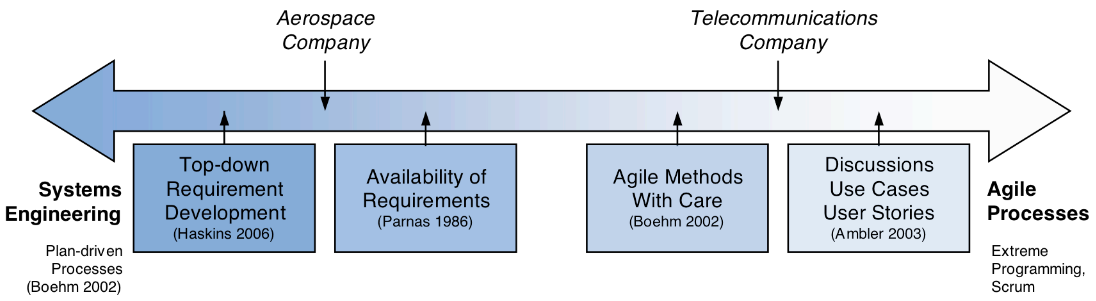
Finally, no literature found suggests improvement strategies for implementing IA. Many of the descriptions of IA simply prescribe the techniques, but do not address the challenges of applying them in practice. This scarcity of improvement strategies is addressed in research question 4 (Figure 1.10). Ultimately, this research question is grounded in the desire of the industry collaborators to understand the effect of IA on product development.

2.8 SUMMARY

Given the scarcity of literature directly confronting the systems-software interface (Section 1.1), a literature search uncovers several research areas noting aspects of this disciplinary intersection. A review of change characterisations, design, maintenance, and change process models, and IA techniques illuminates a disparity in perspectives on how design changes should be handled at the interface. Scarcities within the literature found are also identified in that the difficulties of implementing IA in practice have not yet been fully examined, and the usefulness of IA to manage emergent changes in practice has not been determined. These scarcities and disparities (Section 2.7) identified through the literature review reflect the research questions presented at the beginning of this dissertation (Section 1.6) and are addressed through empirical studies.

3 :: EMPIRICAL STUDIES ON SYSTEM DESIGN AND SOFTWARE CHANGES

This research project conducted empirical studies at two large engineering firms in the aerospace and telecommunications industry sectors to identify prescribed and practised change processes and IA techniques implemented at the systems-software interface. In the aerospace company, embedded software supports the functionality of a primarily mechanical product. This firm advocates using a systems engineering design approach in which system designers develop the high-level software requirements and software designers implement these specifications. In contrast, the telecommunications company develops products integrating hardware and software, and the software delivers most of the functionality of these products. Agile software design processes are implemented this latter company. The selection of these companies for participation in the empirical studies was opportunistic in that both had interest in the research project and were willing to grant access to company resources. Nevertheless, these industry partners characterise IA practice with respect to the systems engineering and agile processes spectrum discussed in the literature review (Chapter 2). Figure 3.1 maps a selection of this literature on prescribed and practised change processes and IA and the industry partners' processes to this spectrum.



Aerospace Company
Telecommunications Company
Systems Engineering
Plan-driven Processes (Boehm 2002)
Top-down Requirement Development (Haskins 2006)
Availability of Requirements (Parnas 1986)
Agile Methods With Care (Boehm 2002)
Discussions Use Cases User Stories (Ambler 2003)
Agile Processes
Extreme Programming, Scrum

Figure 3.1: The systems engineering and agile processes spectrum

This chapter first describes the method used to conduct the empirical studies (Section 3.1). Accordingly, this discussion provides the background for presenting the results of these empirical studies in the remainder of Chapter 3 as well as Chapters 4 and 5 (Section 3.2).

3.1 METHOD OF THE EMPIRICAL STUDIES

Systems and software engineering research increasingly employs empirical studies to investigate design practice (Ott *et al.* 1999; Perry *et al.* 2000; Seaman 1999). Several different approaches to empirical research exist, including (1) biography, (2) phenomenology, (3) grounded theory, (4) ethnography, and (5) case study (Cresswell 1998). The investigation of IA was conducted using an approach informed by grounded theory. As opposed to other empirical research methodologies, grounded theory is data-driven and focuses on developing new theories and explaining phenomenon based on systematically collecting and analysing information. Employing this methodology provides an impartial approach to describing IA practice as observed within the aerospace and telecommunications companies.

3.1.1 GROUNDED THEORY

Grounded theory often only relies on data collection through interviews. However, other data collection methods can also be used within a grounded theory framework. For example, informal discussions, observation, and documentation review can be recorded through memos (Goulding 2002: 64-66). Using such a variety of techniques can obtain additional information not available through interviews, allowing for the *triangulation* of data in that multiple sources can corroborate the information collected (Bratthall and Jørgensen 2002).

Grounded theory stipulates that collected data, including interview transcripts and memos, be analysed through *coding*. Coding entails attaching descriptions or interpretations to sections of interview transcripts or memos. This analysis process should occur throughout the data collection process, and the codes developed should be compared with each other and new data collected in order to refine code definitions and conceptualise the interdependencies between codes. Such code refinement should be performed before the collection of additional data (Seaman 1999). Through this method of *constant comparison*, codes can be grouped into themes, which help to develop over-arching theories and conceptualisations of the data (Strauss and Corbin 1998: 101-161). Through coding, the data collected drives the analysis results and key concepts identified.

3.1.2 DATA COLLECTION AND ANALYSIS

The empirical studies were conducted using a grounded-theory informed approach across two phases of interviews:

- **Phase 1** consisted of an *exploratory study at the aerospace company* and investigated the prescribed and practised change processes and IA techniques (Section 3.1.3).
- **Phase 2** conducted *empirical studies specifically on IA at the aerospace and telecommunications companies* and determined the influences on performing IA in practice (Section 3.1.4).

While conducting these interviews on-site at the aerospace and telecommunications companies, memos were also used to capture data from informal discussion, observation, and documentation review.

Of the techniques implemented, interviews and informal discussions were the primary means to gain insights into the challenges of change management, while observations and documentation reviews supplemented this information gathered. Since the author was often on-site daily at the companies, numerous discussions occurred and provided perspectives on the research project. In most instances, interviews clarified and elaborated on topics covered in these discussions. However, in order to provide an additional means to capture intangible information, such as the topics in these discussions, the tone of interviews, and insights made during observation and documentation review, memoing accompanied all of these activities.

A semi-structured style of interviewing was implemented in that a list of topics was generated before each interview, but the conversation was allowed to progress into relevant areas as initiated by interviewees. All interviewees were male, except for one working at the telecommunications company. Within both companies, key stakeholders, including (1) system designers, (2) software designers, (3) design process engineers[19], and (4) managers, all participated in the interviews. Based on questioning the individuals sponsoring the research project within these companies, no other category of employee directly influenced the processes or IA performed at the systems-software interface[20].

[19] In the participating companies, teams are assigned to define and support the systems engineering and agile processes implmented. This dissertation describes individuals with this role as *design process engineers*.

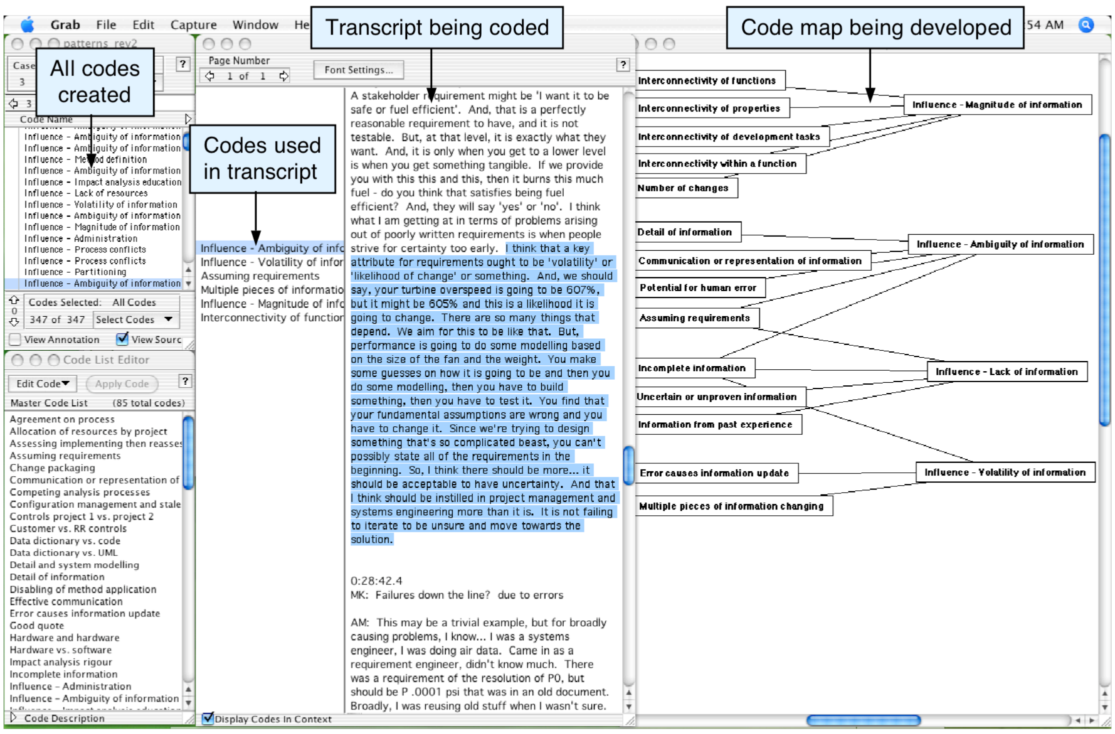
[20] This research project was also discussed with individuals within other areas of the companies. Mechanical and hardware engineers were particularly consulted in the aerospace firm. However, they reported to have little input to the software processes. Similarly, verification and validation teams primarily indicated finding design flaws as opposed to performing IA and root causing the required modifications (Section 3.4.1). Within the telecommunications company, interactions with stakeholders from around the company during a 3-day design workshop confirmed these primary stakeholders, as hardware designers had no input to the IA performed on the software application design (I-40).

Although the partitioning of work between systems and software engineers differ between the two companies and even among design projects within the companies, both companies define systems and software engineering roles. In turn, Appendix A lists the stakeholder role of each interviewee based on his or her assigned job title and allocates each interviewee a unique identifier, which is used as a reference in the remainder of this dissertation. References are given in parenthesis in the form "I-*id#*". Appendix A also includes interviews external to the empirical studies that were conducted at three other companies (a large computer hardware and software development firm, a small company focusing on research and development for computer hardware and software, and a medium-size consultancy working within telecommunications) for general research feedback, advice, and input.

Given that a specific project group within the aerospace company sponsored this research project, full access was available to this group. More information could be readily gathered from these engineers through daily interactions since the author was co-located with this team while on-site. The author mitigated this potential bias through interviewing contacts gained outside of this group, including three different project groups, the software maintenance design team, the team of design process engineers, and senior managers responsible for all software projects. In turn, 49% of the interviews were conducted outside of the sponsoring project design team. Within the telecommunications company, the author did not work closely with a project team. Interviewees across the organisational structure were recruited through personal contacts of the individuals sponsoring the research project within the company. Just as in the aerospace company, a cross-section of project teams, the design process engineering group, and managers provided interviews.

As described by grounded theory, transcripts of interview audio recordings and memos, which captured discussions, observations, and documentation reviews, were coded by identifying and revising overarching concepts and themes. A software tool, HyperRESEARCH[21], was used for all coding activities (Figure 3.2).

[21] More information about HyperRESEARCH™ can be found at http://www.researchware.com.


Grab File Edit Capture Window He
Transcript being coded
Code map being developed
All codes created
Codes used in transcript
patterns_rev2
Page Number
1 of 1
Font Settings...
Code Name
Codes Selected: All Codes
347 of 347
Select Codes
View Annotation
View Sourc
Code List Editor
Edit Code
Apply Code
Master Code List
(85 total codes)
Code Description
Display Codes In Context
Interconnectivity of functions
Interconnectivity of properties
Interconnectivity of development tasks
Interconnectivity within a function
Number of changes
Influence - Magnitude of information
Detail of information
Communication or representation of information
Potential for human error
Assuming requirements
Influence - Ambiguity of information
Incomplete information
Uncertain or unproven information
Information from past experience
Influence - Lack of information
Error causes information update
Multiple pieces of information changing
Influence - Volatility of information

Influence - Ambiguity of information
Influence - Ambiguity of information
Influence - Method definition
Influence - Ambiguity of information
Influence - Impact analysis education
Influence - Lack of resources
Influence - Volatility of information
Influence - Ambiguity of information
Influence - Magnitude of information
Influence - Administration
Influence - Process conflicts
Influence - Process conflicts
Influence - Partitioning
Influence - Ambiguity of information

Agreement on process
Allocation of resources by project
Assessing implementing then reasses
Assuming requirements
Change packaging
Communication or representation of
Competing analysis processes
Configuration management and stale
Controls project 1 vs. project 2
Customer vs. RR controls
Data dictionary vs. code
Data dictionary vs. UML
Detail and system modelling
Detail of information
Disabling of method application
Effective communication
Error causes information update
Good quote
Hardware and hardware
Hardware vs. software
Impact analysis rigour
Incomplete information
Influence – Administration
Influence – Ambiguity of information

Influence - Ambiguity of info
Influence - Volatility of infor
Assuming requirements
Multiple pieces of informatio
Influence - Magnitude of info
Interconnectivity of function

A stakeholder requirement might be 'I want it to be safe or fuel efficient'. And, that is a perfectly reasonable requirement to have, and it is not testable. But, at that level, it is exactly what they want. And, it is only when you get to a lower level is when you get something tangible. If we provide you with this this and this, then it burns this much fuel - do you think that satisfies being fuel efficient? And, they will say 'yes' or 'no'. I think what I am getting at in terms of problems arising out of poorly written requirements is when people strive for certainty too early. I think that a key attribute for requirements ought to be 'volatility' or 'likelihood of change' or something. And, we should say, your turbine overspeed is going to be 607%, but it might be 605% and this is a likelihood it is going to change. There are so many things that depend. We aim for this to be like that. But, performance is going to do some modelling based on the size of the fan and the weight. You make some guesses on how it is going to be and then you do some modelling, then you have to build something, then you have to test it. You find that your fundamental assumptions are wrong and you have to change it. Since we're trying to design something that's so complicated beast, you can't possibly state all of the requirements in the beginning. So, I think there should be more... it should be acceptable to have uncertainty. And that I think should be instilled in project management and systems engineering more than it is. It is not failing to iterate to be unsure and move towards the solution.

0:28:42.4
MK: Failures down the line? due to errors

AM: This may be a trivial example, but for broadly causing problems, I know... I was a systems engineer, I was doing air data. Came in as a requirement engineer, didn't know much. There was a requirement of the resolution of P0, but should be P .0001 psi that was in an old document. Broadly, I was reusing old stuff when I wasn't sure.

Figure 3.2: Screenshot of HyperRESEARCH

This software tool allows for the coding of text files, collates codes and quotes into reports, and provides a means to visualise code relationships and hierarchies in a code map. While other software tools provide similar functionalities (*e.g.* NVivo[22]), HyperRESEARCH was the only tool available for the Macintosh operating system when the empirical studies commenced. Hence, this software tool was used given that the author's computer operated on this platform. Although some literature advocates not using software tools for coding and rather manually coding using pen and paper (Gilbert 2002), manual coding would have been difficult to perform, especially while on-site at the aerospace company. The author did not have access to print transcripts and memos, and the alternative of handwriting transcripts is a laborious task and limits the manipulation of data. Accordingly, computer tools effectively allowed for the transcription and coding during this research project.

During the exploratory empirical study at the aerospace company (Section 3.1.3), transcription, memoing, and coding were performed on-site between interviews and other interactions, and the method of constant comparison was used to refine codes prior to subsequent interviews. The pattern of responses observed allowed for the construction of questions asked in following interviews. Subsequent interviewees pointed out errors and subtleties not initially identified, and, in turn, an

[22] NVivo is a popular qualitative research tool for the Windows platform. More information about this product can be found at http://www.qsrinternational.com.

understanding of the prescribed and practised change processes and range of IA available within these processes emerged. The concepts and themes related to these change processes and spectrum of IA stabilised during this interview process, suggesting that a common understanding had been reached between the author and the interviewees.

The IA empirical studies (Section 3.1.4) built on the concepts and themes identified in the exploratory empirical study and introduced more detailed coding for the issues and difficulties related to implementing IA. Constant comparison was not strictly implemented for the second phase of the aerospace study. Based on the allocated time on-site at the aerospace company, audio recordings of the interviews could not be transcribed and coded prior to the next interview. Interview transcripts and memos were analysed together in order to refine the coding scheme. However, constant comparison was used at the telecommunications company since more time was available between data collection activities.

The coding of the interviews and memos in all empirical study phases provides the findings used in the IA characterisations (Chapter 4). Based on the exploratory empirical study, initial definitions and models for the characterisations were created to describe and clarify the nature of IA. These definitions and models captured the notable differences between the prescribed and practised IA and were refined into the characterisations presented in this dissertation through the IA empirical studies.

Since the IA techniques, tasks, quality, and rigour (Section 4.1, Section 4.2, and Section 4.3) characterisations are descriptive and theoretical in nature, they did not change significantly after the IA empirical studies. More importantly, the interviews during the IA studies illuminated areas for improvement in the influences characterisation (Section 4.4), allowing this characterisation to be more thorough and applicable to a wider range of IA implementations. Specifically, the coding of these interviews helped to refine the meanings of the codes originally produced in the exploratory study. In some cases, codes from the exploratory study with similar connotations were grouped into a single overarching concept. In other cases, codes were divided into sub-codes with more specific definitions. In addition, some of the original codes had been mistakenly used for multiple, dissimilar meanings, and the transcripts and memos were recoded for clarity. All of the transcripts and memos were recoded until all codes had unique meanings and were grouped logically into themes.

3.1.3 PHASE 1 – THE EXPLORATORY EMPIRICAL STUDY

An initial 7-week study on-site at the aerospace company aimed to gain insights into the interface between system and software designers. Given the promotion of systems engineering processes within the company, the empirical study specifically explored the requirements management and change processes implemented, including the IA techniques available during these processes. During this empirical study, IA practices were also initially investigated in terms of the challenges of implementing IA. Thus, the study enquired into the following areas:

1. The change and requirement management processes prescribed by the company,
2. The change and requirement management processes in practice, and
3. The IA performed by the system and software designers.

Twenty-six interviews, lasting between one and two hours each, indicated that variation in the IA implemented existed and that no IA may occur within some change processes, suggesting further inquiry into IA practice. (Appendix A depicts the details of the interviews.)

3.1.4 PHASE 2 – THE IMPACT ANALYSIS EMPIRICAL STUDIES

The second phase included a 1-week follow-up study at the aerospace company within the sponsoring project group, including 13 additional interviews approximately a year after the exploratory study. Five of these 13 interviewees had been interviewed in phase 1, and the remaining interviewees were newly recruited. These interviews typically took about half an hour since the background information on the company processes was collected within the first phase. Six interviewees were systems engineers, and the remaining 7 were software designers. At the time of these interviews, the project only employed 17 systems designers and 20 software engineers. Thus, about 35% of both the system and software design teams participated in this IA study.

Given the disparity between the prescribed and practised forms of IA observed in the first phase, this IA empirical study at the aerospace company focused on understanding the reasons for this difference. In addition, interviewees were asked to respond to a series of high-medium-low ratings of selected parameters to quantify IA practice within the company (detailed in Section 5.2.1) and estimate the *quality* of IA results in order to gauge the effectiveness of the IA performed. Each interviewee was then asked to describe what improvement strategy would allow them to give only high quality ratings to the IA results. Hence, the interviews aimed to:

1. Identify the range of IA implemented in practice, which might not be prescribed by the company
2. Determine the IA perceived as available, which might not be performed,
3. Elicit the influencing factors in selecting the IA performed or barriers to not implementing IA,
4. Quantify the quality of the IA implemented, and
5. Obtain suggestions for IA improvement.

As described in Section 3.1.2, these interviews allowed for the refinement of the IA characterisations (Chapter 4) through the discussion of the first three aims in the list above. As such, the interviews also intended to cover the spectrum of IA typically applied to capture IA practice (Chapter 5) through the parameter rating exercise. Thus, similar to the characterisation development, a structure or model for an elicitation method to capture the range of IA implemented was constructed during the exploratory study. During the IA study, this elicitation method was employed, but also benefited from the data collected and was refined to remove less significant elements to finalise the elicitation method presented (Section 5.1). The development of the method is detailed in Section 5.1.1.

Through this elicitation method, these interviews discussed the above list of topics surrounding IA by walking through a number of specific changes and their associated IA, which recently occurred or were occurring at the time of the interview. To set a basis, each participant was first asked to identify the range of IA techniques perceived to be available, either prescribed by the company or individually developed. Designers were then asked to describe the change and IA performed according to the elicitation method. Finally, participants rated the IA used (detailed in Section 5.2.1), were prompted to discuss the influences for not selecting other available IA techniques, and suggested an IA improvement strategy.

In addition, the second phase of the empirical studies benefited from 9 interviews[23] during a 3-month period (during which the aerospace IA study also occurred) and numerous informal discussions during a 3-day design workshop at the telecommunications company. The interviews at the telecommunications firm were approximately one hour since background questions on the change processes implemented were required. These interviews and discussions centred on the first three elements of the IA empirical study interviews at the aerospace company, as described in the list above. Since no access to a project group could be obtained, discussions on a number of specific changes could not be conducted. As such,

[23] These interviews were not allowed to be audio recorded. As such, notes were taken during the interviews and memos were written afterwards.

interviewees were asked to describe their general opinion on the quality of the results from different IA techniques and to suggest improvement strategies. This empirical study contributes a perspective on IA based on an agile software design process and contextualises the results obtained from the aerospace company.

3.2 EMPIRICAL STUDIES DISCUSSION STRUCTURE

After introducing the products developed by the collaborating companies (Section 3.3), the remainder of this chapter focuses on the change processes, IA techniques, and processes improvement strategies within the aerospace (Section 3.4) and telecommunications (Section 3.5) companies. This discussion provides the foundation for Chapter 4 to present the IA characterisations developed, which generically depict the disparity between prescribed and practised IA. Finally, Chapter 5 describes the elicitation method and makes observations on the IA practised specifically at the aerospace company based on the details of the changes and ratings elicited. These particular observations as well as the elicited strategies for IA improvement are interpreted through the IA characterisations and contribute to addressing the research questions (Section 1.6). Figure 3.3 outlines this discussion of these outcomes with respect to the phases of the empirical studies conducted.

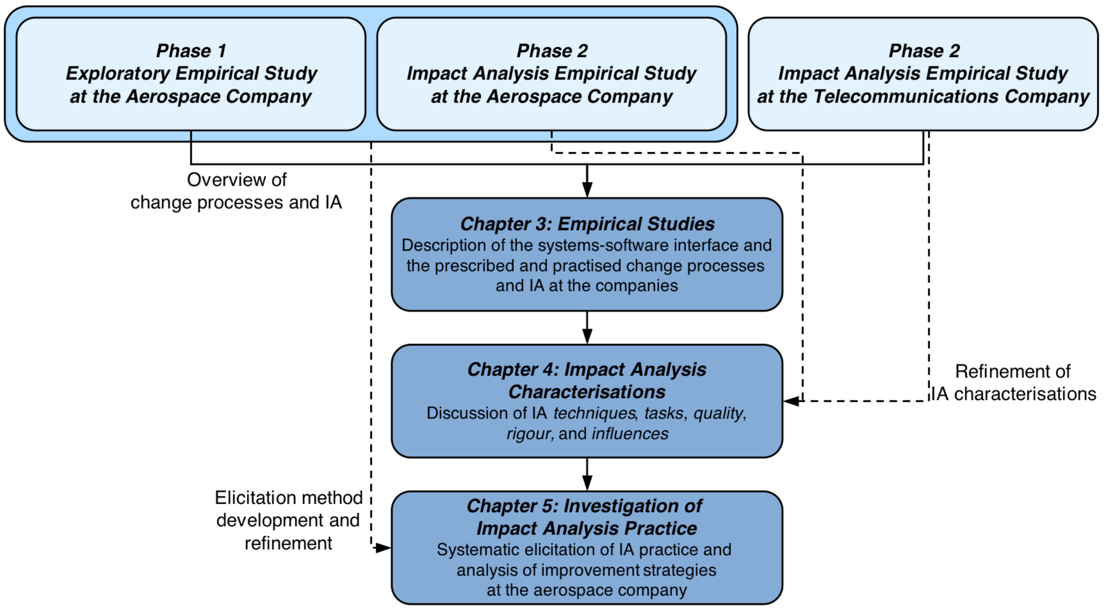

Phase 1
Exploratory Empirical Study
at the Aerospace Company
Phase 2
Impact Analysis Empirical Study
at the Aerospace Company
Phase 2
Impact Analysis Empirical Study
at the Telecommunications Company
Overview of
change processes and IA
Chapter 3: Empirical Studies
Description of the systems-software interface and the prescribed and practised change processes and IA at the companies
Chapter 4: Impact Analysis Characterisations
Discussion of IA techniques, tasks, quality, rigour, and influences
Refinement of
IA characterisations
Elicitation method
development and
refinement
Chapter 5: Investigation of Impact Analysis Practice
Systematic elicitation of IA practice and analysis of improvement strategies at the aerospace company

Figure 3.3: Empirical studies discussion structure

3.3 INTRODUCTION TO THE AEROSPACE AND TELECOMMUNICATIONS FIRMS

The collaborating aerospace and telecommunications companies are both considered large, global firms. The aerospace company has development, manufacturing, and service facilities across 50 countries and employs close to 40,000 people, while the telecommunications firm provides products and services in more than 150 countries and employs over 100,000 individuals. The empirical studies were conducted within the product development departments located within the UK, although interviewees sometimes mentioned their counterparts in other countries. Within this context, system and software design processes only account for a fraction of the product development within a framework of manufacturing, operation, and support. Nevertheless, improving the initial design of the product can have knock-on effects into the operation, maintenance, and decommissioning phases.

Within the aerospace company, the products available have similar software architectures and are derived from similar requirements. The company estimates that at least 70% of the software requirements and design can be reused for new products, while the remaining 30% are application specific. In turn, the product platform of the software design can be essentially customised for specific customers to meet new applications. In some cases, the software of in-service products is also redesigned to provide new functionality or change the existing product behaviour to accommodate the needs of customers. Section 3.4.5 discusses the impact of this product-line approach on the company's strategy for improving software development.

Embedded software implements the algorithms and logic of several electronic controllers within the aerospace company's products. These controllers interact with mechanical actuators and electronic sensors to manipulate the product and ensure safe operation. The customer highly influences the software interface used to operate the product. Each customer can have different interfaces for the information and data supplied as an input to control the product and the operation information outputted from the product, and the aerospace company is required to adapt to different input and output data formats. Since this information is provided through software developed by the customer, the system and software development teams of the aerospace company and customer must work together to ensure the software designs are compatible. Consequently, the customisation of the control system platform as well as the interface with mechanical design and the customer's software design results in a highly iterative development process primarily influenced by the changes performed.

In contrast, software development projects within the telecommunications company widely vary. For example, software is designed to use by representatives in call centres, to manage customer and billing information across many products and services, and to implement the products and services provided by the company. Consequently, the telecommunications firm has setup an independent, in-house software development team to implement the various projects within the company. As such, software implemented to provide new services, as focused on within the empirical study, could be considered one-off development projects. The requirements for these projects often change frequently during the development process, and, based on the company vision and changing business landscape of the company, services may be further developed, combined, or discontinued. Hence, this software must be designed for adaptability and maintenance.

In the past, new software developed to provide such services has not necessarily been synthesised with existing services. Additional stovepipe systems often were developed separately to add new functionality. The nature of this software development practice has caused fragmentation between the services offered. In turn, the telecommunications company is implementing a new all-encompassing and extendable platform to manage the services provided. As opposed to the design process of the aerospace company, which focuses the customisation of products, this strategy places emphasis on change management during the maintenance of the platform delivered.

3.4 FINDINGS FROM THE AEROSPACE COMPANY

The following sections outline software development within the aerospace company at the time of the empirical studies. The systems-software interface is first defined based on the company-defined job roles and the interactions of these designers in handing design changes (Section 3.4.1). The discussion then contrasts the prescribed (Section 3.4.2) and practised change processes (Section 3.4.3) and IA (Section 3.4.4). Finally, the company design process improvement strategies affecting change management are described (Section 3.4.5).

3.4.1 THE SYSTEMS ENGINEERING AND SOFTWARE DESIGN INTERFACE

Within the aerospace company, the control system design group develops the software for the products. Systems engineering, software design, and hardware design roles[24] are explicitly defined by job titles within this development team. *Systems engineers* serve as the primary interface internally between hardware and software designers as well as between the control system group and other *external stakeholders* (*i.e.* not in the control system design group), including mechanical design teams, hardware suppliers, and the customer. Input and negotiation with these external stakeholders allow the systems engineers to develop requirements for the hardware and software. As such, the aerospace company explicitly prescribes using the Vee model for systems engineering (Figure 2.3). Specifically, systems engineers deliver (1) written system requirements to *software designers* and also may provide (2) models of these requirements to clarify this written text in some cases. These models usually depict the intended behaviour of the software design, but not the implementation details.

Typically, a single systems engineer is assigned to develop the requirements and associated models of several *functional areas* of the system design. These areas have distinct purposes within the design and are more or less decoupled from each other. While two or three lead systems engineers with significant experience in designing the product line perform this partitioning of the design for each new product, this division is primarily based on the products previously developed. Depending on the size of a functional area, a single or team of several software designers respond to the system requirement documents and models and develop additional work products for this functional area, including (1) software specifications of the detail design, (2) UML software design models, and (3) code.

The systems engineers are the central point of communication between the hardware and software designers. The majority of the software team only interfaces with systems engineers to develop *application software*, which controls the functionality of the product, based on the system requirements and models. However, a small portion of the software designers may also work directly with hardware engineers to develop *operating system software* for the electronic controller. In addition, software designers tend not to communicate with external stakeholders. Little interaction occurs between hardware suppliers or mechanical engineering and the software engineering team (I-3, I-10, I-11). Some designers consider this communication unnecessary in that systems engineers are updated directly by these stakeholders and the relevant information is passed on to the software designers (I-19). This

[24] The control system development group also defines hardware and software verification and validation roles. However, these individuals do not conduct the primary IA techniques performed during change processes and, thus, are not covered in this dissertation. These roles also were not filled during the exploratory study with the assigned project group.

practice also allows for tight control over the software design in order to meet safety-critical software regulations (Section 3.4.2). However, software designers often may work with the customer to define the details of the inputs and outputs to the software interface (Section 3.3). These discussions usually focus on clarifying the data formats used between the software developed by the aerospace company and by the customer. Even though systems engineers specify this input and output information in a requirement document, this interface usually changes during the product development process, and direct communication between the software development teams often can effectively resolve any issues.

In contrast, electronic hardware engineers often directly contact suppliers and mechanical design teams to develop the suite of actuators and sensors used by the control system. As such, hardware engineers can initiate bottom-up changes to the system requirements and design written by systems engineers based on this communication pattern, while systems engineers primarily spawn top-down functionality changes to the software designers. Software designers may also request changes to the system design, but these modifications tend to focus on the details of the functionality implementation. Hence, systems engineers tightly control the software developed, compared to the relative autonomy of the hardware engineers, and, as such, systems and software engineers are the primary stakeholders to perform the IA on the software design.

While the control system group defines the role of systems engineers, two other teams within the company also perform systems engineering. These teams are responsible for ensuring that the overall customer requirements are met and integrating the control system design with the mechanical design. However, these systems engineers do not directly develop, change, or perform IA on the software requirements and only may have input into negotiations on the general functionality of the software design. Project managers within the control system group and other senior project managers can similarly influence the software functionality. These managers are the primary negotiators in terms of the functionality provided to the customer, but focus on the schedule and budget for the project. Nevertheless, this input can affect the software modifications performed. These project managers convey such changes to the systems engineers for further negotiation to detail the requirements and perform IA.

In summary, systems engineers control the changes to the software design, and only the systems and software engineers perform IA on the software design. As such, the empirical studies focused on the change processes implemented between the systems engineers and software designers within the control system group to analyse the systems-software interface in practice.

3.4.2 THE PRESCRIBED CHANGE PROCESSES AND IMPACT ANALYSIS TECHNIQUES

The control system group implements a software design process that adheres to the safety-critical software regulation, DO-178B, also known as "Software Considerations in Airborne Systems and Equipment Certification". The Radio Technical Commission for Aeronautics (RTCA), the European Organization for Civil Aviation Electronics (EUROCAE), and the Federal Aviation Administration (FAA) use DO-178B for the certification of avionics software. DO-178B requires that all product development processes demonstrate compliance to prescribed planning, development, verification, configuration management, and quality assurance processes through design documentation (Hayhurst and Holloway 2002; Hicks 2006).

Design process engineers synthesise these defined certification processes with "best practices" elicited from systems engineers and software designers within the company to develop useable software development processes. The company-prescribed design processes are made accessible to all designers across design projects within the control system group and are supported through internet-based software tools. Since software development teams in different countries work on different areas of the product line, standardising the design processes used across these teams has been a major undertaking for the design process engineers during the past 8 years.

The following two sections discuss the change processes prescribed by these design process engineers. The control system group implements *formal* change processes (Section 3.4.2.1) from a fixed time during a project when a first baseline for the software requirements and design is set. This point in time varies from project to project based on the scope of the functionality or customisation required by the customer. Prior to this date, changes to any software work products are implemented according to *informal* change processes (Section 3.4.2.2).

3.4.2.1 THE FORMAL CHANGE PROCESS AND IMPACT ANALYSIS

The prescribed formal change process adheres to configuration management literature (Section 2.4.3). A CCB, consisting of the lead systems engineers, lead software engineers, and project managers, organises regular meetings to review change requests initiated by systems or software engineers. Systems engineers tend to submit change requests for the system or software designs on behalf of external stakeholders, given that the company tightly regulates the software change processes in order to follow DO-178B. During the meetings, the CCB makes the decision whether or not to go forward with the proposed design modifications.

In the case of large functionality changes, requiring significant development time and costs, systems engineers submit change requests to the control system CCB and also to the project CCB. The project CCB, which manages the changes for the entire product, but primarily focuses on mechanical design modifications, tends not to make decisions on these change requests and usually agrees with the recommendations of the control system CCB. An engineer reasoned (I-7) that the project CCB does not have a sufficiently detailed understanding of the control system design to make a fit decision. The primary purpose of escalating software changes to the project CCB is to make high-level project managers and other design disciplines aware of potential schedule delays or budget overruns.

Systems and software engineers must describe the impact of a modification on a change request form for either the control system or project CCB. The consequences of the change on the operation safety of the product and an estimation of the time required to implement the change must be indicated through ratings using pre-defined scales and written rationale for the given ratings. In addition, the size of the change in terms of the affected documentation, models, and code must be described. These design change details directly relate to the results of performing IA. Individual change request forms must be submitted separately for system or software design changes.

While change request forms require IA to be performed, no specific form of IA is stipulated by the company's prescribed development process in this context. However, the empirical studies observed that many IA techniques are available to engineers, as depicted by Table 3.1. These available IA techniques also cover the range of IA described in the literature review (Figure 2.15).

The control system CCB[25] reviews the scope of the modification described on the change request form, providing another instance of IA during the change process. However, since this CCB often has many change requests to review during each

[25] The remainder of this section focuses on the control system CCB since they fundamentally determine if change requests are accepted or rejected, as opposed to the project CCB.

meeting, engineering judgement (*i.e.* experiential IA) is often only performed to assess the change request. This CCB rarely, if ever, consults requirements, models, or code through traceability or dependency IA.

If a modification is approved, the change request enters a queue of other accepted change requests. A systems engineer is usually tasked with prioritising these changes and scheduling them for implementation. This *change packaging* can depend on resources available, the similarity of the changes, and the criticality of the modifications. More detailed IA is applied throughout the change implementation.

Table 3.1: IA techniques (from Figure 2.15) within the aerospace company

Impact Analysis Technique	Description
Traceability IA – Requirement traceability (software tool)	Traceability relationships between system requirements and software specifications are captured in a database tool. This tool provides an algorithm to automatically perform traceability IA.
Traceability IA – Requirement documentation (manual)	All versions of system requirements and software specifications are available to engineers in a computer network directory. These documents can be manually searched to determine their interdependencies.
Traceability IA – System and software data dictionaries	A data dictionary captures information about the variables used in the requirement models developed by systems engineers. Another data dictionary is used for the variables used within the software UML models, which are identical to those within the software code.
Dependency IA – Software requirement models	A model is produced to describe and animate the software behaviour given in the system requirements for each functional area. IA can be implemented by manually searching through the model variables for interdependencies or animating the model under different scenarios.
Dependency IA – Integrated software requirement model	Individual software requirement models are integrated into a meta-model describing the functionality and behaviour of the overall software design. IA can be implemented similarly to the individual software requirement models.
Dependency IA – Software UML model	An integrated UML model is developed for the entire software design. IA is primarily performed through manual searches for dependencies across the software design. The UML variable names differ from the software requirement model variables names.
Dependency IA – Software code	Current or previous versions of the software code can be downloaded and manually searched for function and variable usages.
Experiential IA – Formal design review	Design reviews allow for the detection of design flaws in the requirements and models produced by systems engineers and the specifications, models, and code developed by software designers by using systematic meeting procedures.
Experiential IA – Integrated product team meetings	Regularly scheduled integrated product team meetings, in which systems engineers, software designers, and other stakeholders discuss any issues related to a specific functional area, can be used to determine the consequences of design changes.
Experiential IA – Informal discussions	System and software engineers can discuss changes with each other in person since they are co-located. Alternatively, designers contact each other by telephone or email.
Experiential IA – Engineering judgement	Engineering judgement can always be consulted to determined the impact of changes based on individual knowledge and understanding of the design.

In many cases, different engineers perform the IA for the change request than during the change implementation. As such, these engineers can often obtain different IA results, depending on factors, such as the IA techniques applied and the information available at the time of IA. For instance, an engineer may not perform IA on the requirement documentation and assume that no modifications are necessary, while another designer may notice changes required to the requirements when reading through the documentation. Such a disparity in IA results can cause additional, knock-on changes to be noticed during the change implementation or even later during development. As an outcome of this formal change process, all necessary modifications are expected to be made to the system requirements, software requirement models, detail software specifications, UML models of the software detail design, and software code as well as the traceability relationships captured within the traceability database tool and data dictionaries.

3.4.2.2 THE INFORMAL CHANGE PROCESS AND IMPACT ANALYSIS

Due to the compression of project development schedules by the company to stay competitive in their market, software design begins prior to the finalisation and sign-off approval of the system requirements. Systems engineers release requirements and accompanying requirement models to the software designers for implementation after receiving sufficient input from relevant stakeholders. However, given that the needs of stakeholders change, especially during the early phases of the project, design modifications may be necessary after this release of requirements and prior to the first baselining of the requirements and design. As such, these changes occur through informal change processes.

As prescribed by the company, if a system or software designer wishes to make such a design change, lead system or software designers, respectively, which later serve on the CCB, must evaluate the proposed change. The lead engineer grants approval or rejects the change based on the impact of the change and justification for the modification. In some cases, external stakeholders are involved in these decisions for system design changes. These changes are occasionally and inconsistently documented in a computer database, but are more often captured through emails or informal reports.

Prior to the first baselining of the design (Section 2.4.3), not all of the requirement documentation exists, and the traceability relationships have not been inputted into the traceability tool or data dictionaries. Furthermore, not all of the requirement models or UML models have been implemented. Thus, identifying all of the interdependencies between the design partitions is often not possible using traceability and dependency IA. Often the impact of the change must be estimated using experiential IA. In turn, this informal change process tends to occur through discussions. Given that these discussions or the approval or rejection of the design change require no documentation, this process facilitates communication within the team and provides for an agile process of immediately handling and implementing design modifications.

3.4.3 THE CHANGE PROCESSES IN PRACTICE

Although the formal and informal change processes are generally followed as prescribed in practice, the implementation of these change processes is influenced by the compressed development schedules. As a result of reducing the time allotted for projects, more and more of the design work must occur concurrently. Specifically, more of the software design must occur earlier in the design process without signed-off or agreed requirements from stakeholders. During the initial stages of the design process, stakeholders may need to wait for information from suppliers or experiment test results to determine the specifics of the control system requirements. Nevertheless, the software design begins, sometimes without stakeholder input to requirements or with requirements that are known to change. A system designer (I-13) explained:

> *The main challenges come from the time-scale of the project. You can't do it the way you are supposed to - top-down - because of time scales. So, hardware are ordering and acquiring things before they have got the requirement. Software starts coding before we get them their requirements. So, it's all done all at once. Fair enough, that's real life.*

Since system requirements are often similar between existing designs in the product line, previous requirements and software designs are used as a starting point, thus, removing the necessity for specifics from stakeholders to begin design work. These requirements and design are then changed as necessary to meet the specific stakeholder needs. This coping strategy for handling changes often works effectively for much of the product functionality because the control system designs are similar within the product line. However, based on a company investigation of software changes across the product line, approximately 30% of the software requirements are highly volatile and change frequently across the products. As such, reusing requirements between projects does not always prove effective.

Alternatively, in the case of new functionality development, the requirements also may be initially unknown before design commences. Systems engineers may write several requirement documents, each with different functionality scopes, and release these documents to the software engineers for implementation. In turn, the software designers model and code several different implementations of the functional area. Once the final requirements are known, documented, and signed-off, the software designers then make modifications to the selected implementation and delete the unnecessary implementations within the software design and code (I-3, I-4). Finally, in other instances, the functionality required continues to change throughout the development process. Requirement documents may be completely rewritten, leading to significant redesign of the software. The systems and software engineers do not employ definitive coping strategies for changes in this situation (I-30).

Given that software design work commences without fixed requirements, many changes occur in practice during development. System and software design changes to a single functional area can occur simultaneously. As such, even formal change processes can have some informal features in that the scope of changes in original change requests are often expanded during implementation to handle or provide for other concurrent or upcoming modifications. Acceptance of such additional changes can occur through informal discussions without necessarily any documentation or IA (I-3, I-28). Working in parallel causes changes to be handled in parallel as well.

If such internally generated changes to a functional area are captured within a single iteration of the system and software design, then relatively little administrative costs for the changes are necessary. However, if much iteration occurs due to finding and implementing changes piecemeal, the administrative costs for the change processes with respect to the amount of change performed are more significant. These costs are firstly composed of the work involved for writing the change request, holding the CCB, and performing the design modification. More significantly, testing must be redone every time a change is implemented within a software function. Hence, the effectiveness of the change processes in terms of administration costs is influenced by the complete identification and single execution of changes.

Due to the concurrent development of mechanical, hardware, and software designs, externally-initiated system and software design changes also can surface individually then interact with on-going modifications, leading to additional modifications. For mechanical and hardware components, design flaws or integration issues between these physical components may be identified only after manufacturing has begun or orders have been placed. The system and software designs may unexpectedly change to account for these deficiencies. For instance, a manager (I-2) described a mechanical flaw that required the development of additional software safety checks. These changes interacted with other simultaneous mechanical changes, causing ripples of many unanticipated, emergent changes and delaying the product delivery. As another example, a system designer (I-15) discussed a series of software changes made incrementally during the design process to account for hardware design faults. These changes compounded and ultimately caused the area of software design to unexpectedly produce undesirable system behaviour when certain conditions were met. Unfortunately, this problem was only detected after the product was released and product failures occurred. As such, concurrent development can unexpectedly increase the number of system and software design changes and the administrative costs of their implementations, but also largely affect project management and product success.

Change packaging can reduce these administrative costs initiated by compressed design schedules and concurrent design (I-2, I-21, I-22). For example, queuing related changes in documentation, models, and code allows designers to implement multiple changes to a functional area simultaneously. Particularly for code changes, change packaging eliminates the need to re-test multiple times for each singular change implemented. Designers can also identify the interactions of the set of changes during implementation, finding emergent effects initially and reducing the need for additional change processes. In effect, change packaging provides another coping mechanism to handle design changes and implement effective software development.

3.4.4 THE IMPACT ANALYSIS TECHNIQUES IN PRACTICE

Given that the aerospace company does not prescribe the use of IA techniques and many are available for use, designers are allowed to freely choose the IA applied. As observed in the empirical studies, the IA implemented depends on the type of change. For instance, systems engineers may rely on a form of traceability IA to a high-level requirement change, and software engineers may only use dependency IA on a low-level software design change. Alternatively, designers may employ a single favourite IA technique as opposed to implementing multiple different methods. In some cases, designers suggested that IA is not performed at all during a change process. This scenario can occur when a low-level design change, such as a variable value change in the code, is simply executed without determining if there are consequences to other areas of the design. This lack of IA for such a number change actually was a primary cause of a design failure after product release (I-15). Provided that this example may be an extreme case, nevertheless, it emphasises the importance of IA.

During the empirical studies, designers stressed the importance of implementing IA. For instance, a systems engineer (I-21) stated: "Impact analysis is absolutely really important". This designer then described using the requirement traceability tool, data dictionaries, and requirement models. Another systems engineer (I-26) said: "I think there should be up-front analysis of the change packages. I'd like to see impact analysis done there rather than these other techniques that are more reactionary than forecasting". Managers (I-2 and I-23) similarly indicated that IA is crucial within the design process. However, some other systems and software engineers did not understand how to use the IA techniques available. Commonly, designers questioned how data dictionaries could be used to perform IA (I-14, I-25). Many designers (I-15, I-19) also did not know that the requirement traceability tool included a feature that could automatically perform IA. Furthermore, several other systems engineers (I-13, I-19) suggested that the review processes are not focused on finding design flaws and are rather a process formality. In particular, one (I-17) asserted: "Do they (engineers) not have time to do them (design reviews)? You always find time to fix problems in the end, but not time to prevent them in the beginning". These observations suggest the lack of IA prescription and lack of implementation of some of the IA techniques, even though IA is acknowledged as an important element of the design process.

In summary, this description of the aerospace company's change processes and IA practices sets the stage to characterise how IA can differ as prescribed and implemented in Chapter 4. Chapter 5 builds on these descriptions to analyse the specific IA techniques typically applied by designers within the aerospace company and develop strategies for IA improvement.

3.4.5 THE STRATEGIES FOR IMPROVING THE DESIGN PROCESS

Given that this dissertation assesses IA improvement strategies (Section 5.3) and proposes heuristics for IA improvement (Chapter 8), other design process improvement strategies also must be considered. Specifically within the aerospace company, an initiative is being implemented to collate and systematically reuse control system requirements. Standardised requirement documentation, models, and traceability relationships are being developed in order to efficiently extend the product line and customise products. As such, more rigorous forms of traceability and dependency IA are also being investigated for these requirements and models, in turn, providing effective means to manage design changes.

In addition, another initiative within the aerospace company is standardising the design and change processes prescribed, as suggested in Section 3.4.2. Previously, each design project defined new processes and implemented new design tools in an attempt to improve the product development from earlier projects, and, in some cases, these simultaneous changes led to worse design processes. Consequently, the new company strategy tightly controls the changes made to the processes and tools in order to systematically and incrementally improve their product development processes. However, acceptance and support of this initiative by designers and managers has been difficult to obtain. A design process engineer (I-16) stated:

> *My biggest worry is that for every program here people are going to reinvent the wheel… unless they get some standardisation or agree to some standardisation. I see a great reluctance to do that since everyone thinks that they can do it better than the previous project, but I think that is the biggest risk for this organisation. What they perceive as 'we can do this better than the previous one' will turn into 'oh dear, this is not better than the previous one'.*

Designers may also have a preference for familiar, past processes and are reluctant to change their use of design tools. Specifically, some IA tools and techniques used on past projects are no longer supported, and some designers suggested a preference for these forms of IA. A systems engineer (I-30) described his preference for Yourdon methods previously used since he could easily determine the impact of changes. He said:

> *This is going back to the days when we use to do a lot of Yourdon analysis and so on. The whole models were data driven. Because of that, if anyone wanted a piece of data, it had to be declared up front. You had to declare that in the Yourdon structure so you could use it. Here we tend to do it after the event. So we do our designs, we try to fit them together… the data dictionary is always so far behind, as it stands today.*

Given that the standardised processes prescribed do not outline the use of IA, this systems engineer could not perform IA to find out the flow a particular variable in the requirement model because the data dictionary information was incomplete. Arguably, the standardised processes should allow for the necessary IA methods to implement design changes. As such, these processes should continue to evolve and support the implementation of the IA tools available, thus, potentially allowing for acceptance and adaptation of the new methods by designers. Moreover, understanding the influence of IA on the design process can provide for this evolution.

3.5 FINDINGS FROM THE TELECOMMUNICATIONS COMPANY

The following sections highlight the systems-software interface (Section 3.5.1), the context of the agile development methods implemented (Section 3.5.2), and the prescribed (Section 3.5.3) and practised (Section 3.5.4) change processes and IA within the telecommunications company. Their strategies for processes improvement are also discussed (Section 3.5.5). Comparing these empirical findings with the aerospace company suggests the influence agile development processes and the nature of the products designed can have on the change processes and IA implemented. The IA characterisations (Chapter 4) draw from this discussion to describe how IA practice can vary.

3.5.1 THE SYSTEMS ENGINEERING AND SOFTWARE DESIGN INTERFACE

Although agile development processes do not define systems engineering as a role, the telecommunications company includes two different types of systems engineering positions in their team structure. At a design project level, *system architects* are responsible to ensure that the customer needs and requirements are met and perform IA on the software design. At a higher level, *solution designers* synchronise the requirements for multiple projects in order to meet the overall, strategic goals of the company in terms of the product suite developed, while providing input into individual design projects. Although these designers do not necessarily produce requirement documentation, as do the systems engineers in the aerospace company, nevertheless, they elaborate on the system requirements for the software design through other work products and communication means grounded in agile development methods. In some cases, they can even become involved in synchronising detail designs and perform IA across software designs. As such, *software engineers* design and implement software code to meet the requirements produced by the system architects and solution designers, and this systems-software interface is comparable to that within the aerospace company (Section 3.4.1). However, given the different project sizes within the company (Section 3.3), system architects and software engineers may only be responsible for the entire product design, from requirements to code, in smaller projects.

Similarly, stakeholders can vary from project to project. Generally, an in-house customer (*i.e.* another area of the company commissioning the product) is the primary stakeholder. In some cases, in-house hardware engineers can also be stakeholders. However, standard computer hardware is frequently purchased from suppliers. Neither of these stakeholders typically performs any detailed IA on the software design, but may request software design changes.

3.5.2 THE CONTEXT OF IMPLEMENTING AGILE DEVELOPMENT METHODS

The telecommunications company began transitioning to agile development practices 5 years ago to focus on producing working software and align with the agile manifesto (Beedle *et al.* 2001). Design process engineers promote the use of agile techniques within the company by running training programs and assisting in the development of processes on each project. Given this transition to agile development processes, design process engineers work to educate designers on agile methods and displace the waterfall processes long used within the company. In addition, they are aiming to certify their agile processes implemented with CMMI as well as define off-shore outsourcing procedures since software coding is performed less and less within the UK. The integration of agile processes with these two elements has proved challenging for the design process engineers.

Large development projects also pose another challenge for the agile development prescribed. Even though agile development processes tend to be associated with small development teams, the telecommunications company is also applying agile development to large-scale projects. Design process engineers (I-42 and I-43) admit that systems engineering techniques are typically implemented along with a few agile techniques in these cases. In particular, these projects often focus on requirement development and system design partitioning before the majority of the software design commences for each segment of design and code incrementally delivered. A manager (I-44) suggests that a systems-engineering approach has been found to be more appropriate and notes that the company has recognised the limitations of implementing agile processes in these projects. Despite the mandate of agile processes, this manager speculates that the company may soon undergo another change and begin to prescribe systems engineering processes.

3.5.3 THE PRESCRIBED CHANGE PROCESSES AND IMPACT ANALYSIS TECHNIQUES

The telecommunications company prescribes that all projects must deliver work products and software code to the customer at fixed intervals throughout the development process. These intervals are typically between one and two months. However, this company also stipulates that projects individually define the combination of different development techniques to tailor suitable design and change processes. For example, design projects can combine Scrum daily meeting at which all design team members meet to discuss their work output for the day with pair programming as well as with other agile techniques. This practice accounts for the range of software projects sponsored and also greatly varies the development processes implemented in practice.

Some projects implement formal change processes just as in the aerospace company with a CCB (Section 3.4.2.1), while others only implement change processes resembling the informal change processes within the aerospace company (Section 3.4.2.2). In the later case, team meetings or individual discussions may determine if proposed changes should or should not be implemented and the packaging of these changes, and story cards or backlog databases may only capture and document any design modifications.

The design process engineers support a range of software tools to implement IA, including requirement traceability databases and software modelling programs. While these tools are available for use by design projects, the company does not specifically prescribe their use on any project similar to the aerospace company (Section 3.4.2.1). Thus, different projects can implement different IA tools. However, given the agile processes mandated, the company particularly encourages experiential IA. Team communication and design reviews used for IA are recommended and occur frequently during projects. In some projects, experiential IA may be the primary form of IA besides directly using software code to perform dependency IA.

3.5.4 THE CHANGE PROCESSES AND IMPACT ANALYSIS TECHNIQUES IN PRACTICE

Although the interviewees generally indicated that agile development processes were implemented in practice more and more, several suggested their preference for other design processes. For instance, a system architect (I-37) and software designer (I-38) interviewed indicated that they liked the processes used prior to the switch to the agile development methods. They suggested they now had to place too much work effort into developing the design processes they used on projects as opposed to previously when they could focus more on designing software products. While analysis and understanding of the design process implemented can be beneficial, these interviewees maintained that they believe that the processes implemented were not efficient. As such, the company culture may not yet have fully adopted the mandate of agile processes.

Similarly, even though customers are encouraged to actively take part in the software development process and particularly design reviews, the design process engineers (I-39, I-42) interviewed suggested that project economics does not always provide for this practice. Given that the customers within the company also have their jobs to fulfil, these individuals often cannot spare working on-site with the design team. Some of these customers also do not accept the agile development perspective and believe that they are contracting the software design and should not be required to put significant effort into the development process. In turn, this lack of adoption of agile development methods affects the experiential IA relied upon, causing it to be less useful to obtain customer feedback, which agile processes often demand.

The design process workshop observed and interviewees (I-37, I-38, and I-39) further indicated that experiential IA is primarily implemented in practice. Through the adoption of agile processes, other forms of IA have been abandoned in some cases since they require too many resources. For example, traceability IA is now only implemented in large projects. This practice contrasts with the aerospace company in that the telecommunications company projects appear to select a few IA techniques to implement, while the aerospace company has opted to investigate more costly and in-depth means of traceability and dependency IA for all projects. These differences are likely due to the nature of the products developed and change processes implemented. Since the aerospace company essentially performs customisation to extend the software product line, investing in IA techniques can be beneficial. The telecommunications company may prefer experiential IA since the products do not stem from a product platform, and the product requirements often change during development.

3.5.5 THE STRATEGIES FOR IMPROVING THE DESIGN PROCESS

Currently, the design process engineers are educating and promoting the adoption of these practices to shift the company culture. The primary focus is to uproot the legacy of waterfall development and build enthusiasm for agile processes. As envisioned, implementing agile development methods is the design and change process improvement strategy. To a lesser extent, as suggested in Section 3.5.2, the design process engineers are also refining the agile processes supported to comply with CMMI and allow for outsourcing procedures. Although incorporating systems engineering practices may be the step taken to improve the development processes for larger design projects, no definitive measures have been taken to adjust the agile development processes.

3.6 SUMMARY

This chapter first describes the method used to conduct the empirical studies at the aerospace firm and telecommunication company and outlines the structure for discussing these studies in this dissertation. Subsequently, the systems-software interface is then depicted in terms of each company's structure, and the change processes and IA are then summarised as prescribed and practised. Finally, this chapter contrasts the strategies for process improvement in the companies. In turn, Chapter 4 further discusses and characterises this disparity of prescribed and practised IA, and Chapter 5 investigates a variety of IA improvement strategies within the context of these process improvement initiatives.

The empirical studies address the first research question (Figure 1.7) through the observations of the company-prescribed change processes and IA. Both companies support the formal change processes prescribed within literature. However, these firms also do not necessarily strictly enforce their application of these change processes and also use informal change processes. In addition, designers are granted access to the range of IA techniques outlined within literature in both companies, but both company-prescribed processes do not specify the application of IA techniques within either the formal or informal change processes. In fact, designers may not be educated on the range of IA in some cases, leading to the disuse of some of the techniques. Hence, the empirical studies imply that there is a lack of IA prescription within change processes by industry, which the literature review does not explicitly identify. The companies allow for the implementation of the literature-prescribed change processes and IA techniques, but do not enforce their application. Furthermore, unlike the agile process literature disparaging traceability IA, the agile development processes supported in the telecommunications firm provide for traceability IA. Thus, the empirical studies suggest that the disparity identified in the literature review between systems engineering and agile design perspectives does not necessarily occur in practice.

4 :: IMPACT ANALYSIS CHARACTERISATIONS

The observations and coding of the interviews and memos from the empirical studies allow for the conceptualisation of key themes surrounding implementing IA in practice. This chapter discusses these results from the empirical studies in terms of four characterisations of IA. Specifically, the following characterisations define the difference between prescribed and practised IA, outlined in the overview of the aerospace and telecommunications company change processes and IA (Chapter 3), through the terminology of IA *techniques* and *tasks* (Section 4.1) as well as IA *quality* (Section 4.2), describe a scale for this disparity through IA *rigour* (Section 4.3), and classify *influences* affecting the actual implementation of IA (Section 4.4). These characterisations are evaluated through literature on software development practice and by the industry collaborators (Section 4.5). Thereby, the IA characterisations establish the foundation for further inquiry into IA practice and improvement strategies specifically occurring within the aerospace company (Chapter 5).

4.1 IMPACT ANALYSIS TECHNIQUES AND TASKS

The empirical studies suggest that IA may not be applied as prescribed or idealised. Some interviewees discussed the lack of IA implementation in practice, while others suggested the difficulties in applying IA (Section 3.4.4 and Section 3.5.4). This dissertation distinguishes this disparity by taking IA *techniques* to include the possible methods or approaches to estimating the scope a change (*e.g.* traceability IA) and IA *tasks* as the actual implementation of IA techniques within a particular context (*e.g.* without complete traceability relationships) (Figure 4.1).

IA Technique:
The theoretical method or approach
to estimating the scope of a change

IA Task:
The actual implementation of an IA technique
within a particular context

Figure 4.1: Definitions of IA techniques and tasks

4.2 IMPACT ANALYSIS QUALITY

In turn, this research proposes that the *quality* of IA[26] depends on the actual results of the IA tasks performed as opposed to the theoretical results of the IA techniques (Figure 4.2). Going through a systematic process of applying an IA technique that produces incorrect or useless results is not necessarily better than performing an *ad hoc* examination yielding correct and helpful results. Under this guise, the quality of the IA results can be characterised, for example, by *completeness*, *correctness*, and *clarity*[27]. Complete IA results indicate all possible system and software design artefacts that are affected by an initiating change; correct IA results do not erroneously denote work products that are affected by the change; and clear IA results communicate potential change impacts unambiguously. In turn, increasing the quality of IA results reduces the likelihood of emergent, knock-on modifications by providing an understanding of actual change impacts.

IA Quality:
Completeness, correctness, and clarity
of IA results produced from IA tasks

Figure 4.2: Definition of IA quality

4.3 RIGOUR OF IMPACT ANALYSIS TECHNIQUES AND TASKS

Given that IA quality characterises the results of IA tasks, this research introduces the concept of IA *rigour* to describe IA results with respect to IA techniques. This research takes IA rigour to correspond to the risk of unexpected change propagation, similarly to IA quality. More rigorous IA is expected to reduce the tendency of emergent changes, while less rigorous IA may lead to knock-on modifications. However, as opposed to IA quality, IA rigour (Figure 4.3) correlates to the size of the search space for possible change impacts, which depends on the IA technique used.

IA Rigour:
Thoroughness of the search for potential
impacts of a design change

Figure 4.3: Definition of IA rigour

[26] Note that the cumulative influence of applying multiple IA techniques can affect this characteristic.
[27] These three attributes mimic adjectives used to describe well-defined, high quality requirements. In requirements engineering literature, other adjectives, such as *consistency*, *unambiguity*, *usability*, or *reliability*, have also been used with similar and overlapping definitions by published standards and authors. Thus, there are other possible adjectives that could be used to describe IA results (Kececi and Abran 2001). However, the terms, completeness, correctness, and clarity, suggest the essence of IA result quality to anticipate knock-on changes.

As such, more rigorous IA is expected to systematically search all possible means of change propagation, decreasing the potential for unanticipated, knock-on changes, while less rigorous IA may perform a more *ad hoc* or unsystematic search process and has a risk of requiring unexpected modifications. As a consequence, IA rigour can be used to compare the different approaches of IA techniques. Figure 4.4 suggests a scale of rigour for instances of IA techniques. This characterisation does not aim to classify all IA techniques possible and only suggests a relative rating of typical IA techniques, also classified the literature review (Figure 2.15). In turn, more rigorous IA techniques (*i.e.* traceability IA) do not necessarily translate to being *good* IA techniques; likewise, less rigorous IA techniques (*i.e.* experiential IA) are not always *bad* IA techniques. The quality of IA results ultimately indicates if the IA techniques applied were sufficient or *good* enough.

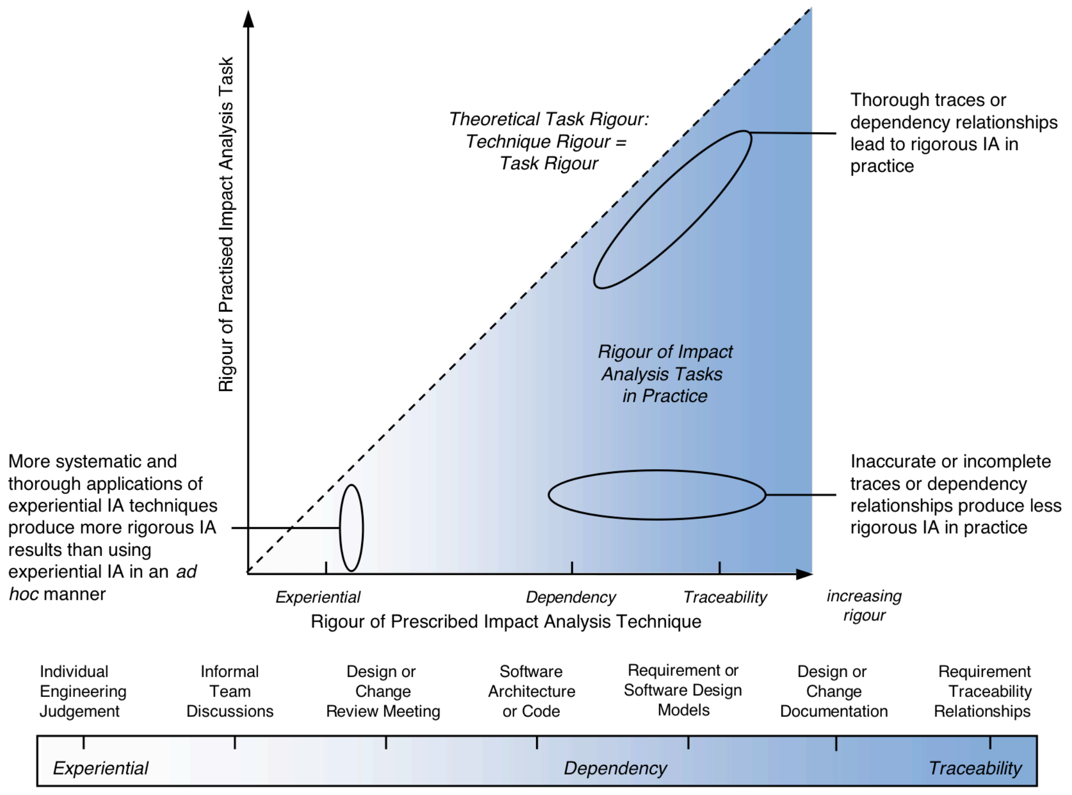

Rigour of Practised Impact Analysis Task
Theoretical Task Rigour: Technique Rigour = Task Rigour
Thorough traces or dependency relationships lead to rigorous IA in practice
Rigour of Impact Analysis Tasks in Practice
More systematic and thorough applications of experiential IA techniques produce more rigorous IA results than using experiential IA in an ad hoc manner
Inaccurate or incomplete traces or dependency relationships produce less rigorous IA in practice
Experiential
Dependency
Traceability
increasing rigour
Rigour of Prescribed Impact Analysis Technique
Individual Engineering Judgement
Informal Team Discussions
Design or Change Review Meeting
Software Architecture or Code
Requirement or Software Design Models
Design or Change Documentation
Requirement Traceability Relationships
Experiential
Dependency
Traceability

Figure 4.4: Rigour of IA techniques and tasks (from Figure 2.15)

Figure 4.4 shows IA through requirement traceability relationships as the most rigorous technique. This method enables systematic (and often automated) searches for the effects of changes from high-level product functionalities to software detail designs. In theory, similar results can be obtained through manual searches through documentation. However, requirement traceability relationships may capture additional, secondary design dependencies than referenced explicitly in documentation. Thus, manual methods may not include such comprehensive searches.

Dependency IA may provide less thorough searches for possible change impacts and, hence, is less rigorous than traceability IA. Requirement and software design models or software architecture and code dependencies (between variables, logic, modules, etc.) may not necessarily connect the detail design with overarching system functionalities, resulting in a limited search space. Specifically, requirement and software design models abstract away from high-level requirements of product designs, and interdependencies due to mechanical or physical constraints may be lost in the resulting decomposition. Software code may contain even less information to determine the implications of changes on the high-level system design than these models. Thus, Figure 4.4 places dependency techniques involving models at a higher level of rigour than dependency analyses using software architecture and code.

Experiential IA is classified at the lower-end of the rigour scale. Although expert judgement can provide the best form of IA in a situation through tacit knowledge of dependencies not captured by documentation, models, or code, this method for searching possible change impacts is often more *ad hoc*. Consequently, emergent changes can occur using this IA technique, and the rigour of experiential IA is lower than traceability and dependency IA. However, by increasing the number of experts or individuals performing the IA, IA rigour may increase given that the thoroughness of the search can theoretically improve. The systematic discussion of a design with a variety of relevant stakeholders, such as through review meetings, may further increase IA rigour. Similarly, reviewing captured design rationale to support performing more systematic analyses (Section 2.5.3) can enhance the rigour of these experiential IA techniques. Hence, the scale of experiential IA in Figure 4.4 corresponds to this order.

Figure 4.4 also suggests the relationship between IA techniques and tasks in terms of rigour. In theory, an IA task should be performed exactly as prescribed by the IA technique. However, influences can shape the application of these techniques in practice, as described in the next section. In effect, the relative rating scale of IA techniques does not imply that these techniques are more or less rigorous in practice. For example, design reviews may provide a more thorough search for potential changes than incomplete requirement or software design models.

4.4 INFLUENCES ON IMPACT ANALYSIS TECHNIQUES AND TASKS

None of the literature found related to IA thoroughly discusses the reasons or rationale for the difference between the prescribed and practised IA (Section 2.7). However, the empirical studies suggest that the disparity between prescribed IA techniques and practised IA tasks can depend upon many influences. The responses from interviewees have been categorised into *technique* and *task influences*. Technique influences affect IA rigour or prohibit the actual application of an IA technique through design or change procedures or methods, while task influences inhibit the IA result quality across IA techniques through the context of the technique application. In other words, technique influences can systematically and repeatedly affect the performance of IA techniques, and task influences affect specific incidences of applying IA techniques, independent of the processes implemented. For instance, the lack of procedures to update traceability relationships can lead to incompleteness of these captured traces and, in turn, reduce the search space for potential change impacts and IA rigour, exemplifying a technique influence. Designers even may not perform such traceability IA given the incompleteness of results, potentially leading to disuse of the technique. In contrast, as an example of a task influence, a lack of information, resources, or time to perform traceability IA in a certain instance can also yield poor quality IA results. These influences can also simultaneously inhibit forms of dependency and experiential IA as well.

The factors affecting IA cited in the empirical studies were grouped into the high-level themes of technique and task influences by analysing and coding interview transcripts and memos. In turn, these high-level themes were constructed from a hierarchy of key sub-themes. Figure 4.5 and Figure 4.6 give an overview of the coding analysis results in terms of these key sub-themes through the code map developed using the HyperRESEARCH software application (Section 3.1.2). The key sub-themes presented correspond with the specific factors described in the IA influences characterisation, summarised in Figure 4.7. For instance, partitioning and synchronisation are two key sub-themes identified in the coding process and, hence, are two IA technique influences discussed in the IA influences characterisation.

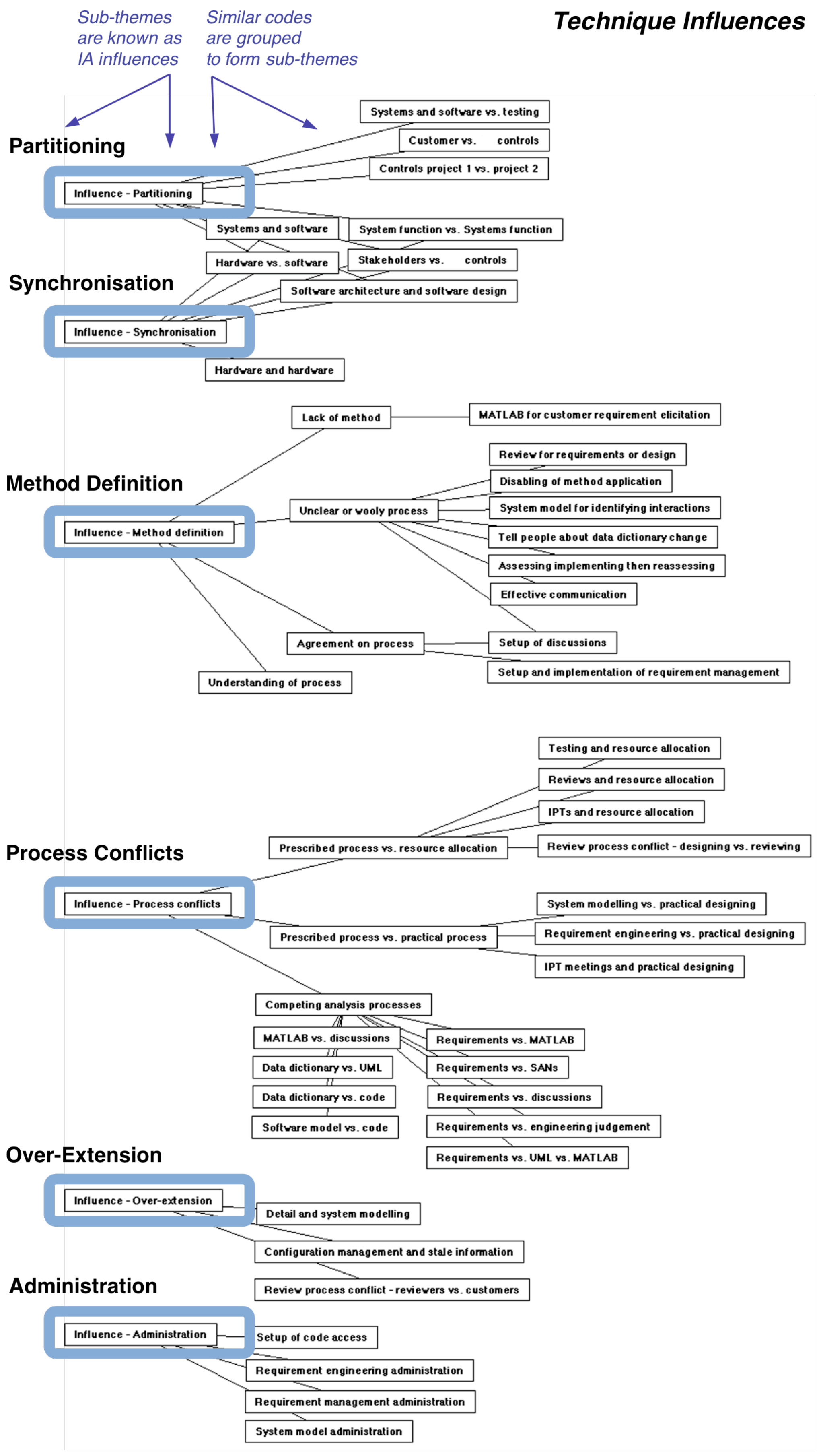

Sub-themes are known as IA influences
Similar codes are grouped to form sub-themes
Technique Influences
Partitioning
Influence - Partitioning
Systems and software vs. testing
Customer vs. controls
Controls project 1 vs. project 2
Systems and software
System function vs. Systems function
Hardware vs. software
Stakeholders vs. controls
Software architecture and software design
Synchronisation
Influence - Synchronisation
Hardware and hardware
Lack of method
MATLAB for customer requirement elicitation
Review for requirements or design
Disabling of method application
Method Definition
Influence - Method definition
Unclear or wooly process
System model for identifying interactions
Tell people about data dictionary change
Assessing implementing then reassessing
Effective communication
Agreement on process
Setup of discussions
Setup and implementation of requirement management
Understanding of process
Testing and resource allocation
Reviews and resource allocation
IPTs and resource allocation
Prescribed process vs. resource allocation
Review process conflict - designing vs. reviewing
Process Conflicts
Influence - Process conflicts
System modelling vs. practical designing
Prescribed process vs. practical process
Requirement engineering vs. practical designing
IPT meetings and practical designing
Competing analysis processes
MATLAB vs. discussions
Requirements vs. MATLAB
Data dictionary vs. UML
Requirements vs. SANs
Data dictionary vs. code
Requirements vs. discussions
Software model vs. code
Requirements vs. engineering judgement
Requirements vs. UML vs. MATLAB
Over-Extension
Influence - Over-extension
Detail and system modelling
Configuration management and stale information
Review process conflict - reviewers vs. customers
Administration
Influence - Administration
Setup of code access
Requirement engineering administration
Requirement management administration
System model administration

Figure 4.5: Code map of IA technique influences

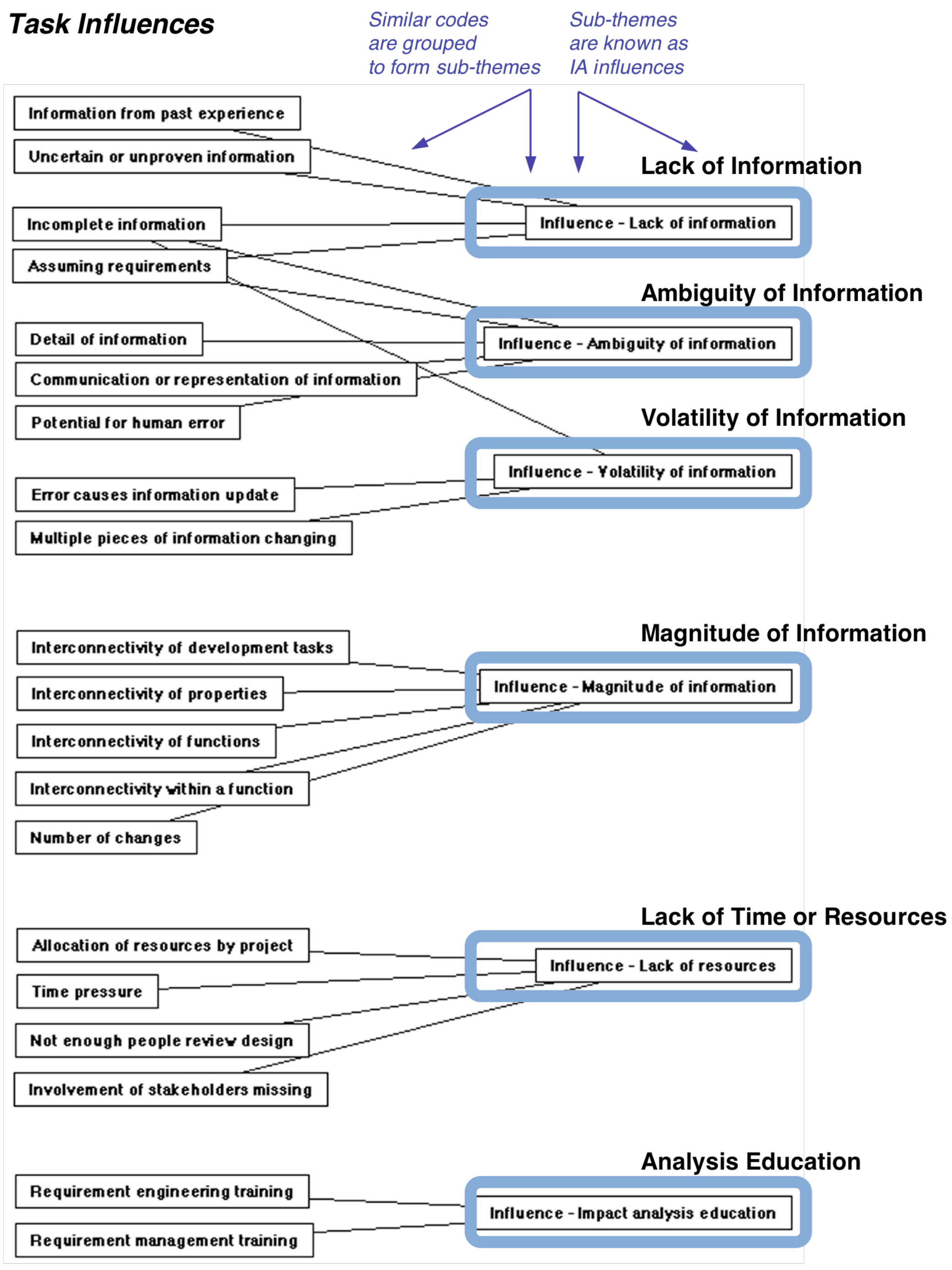
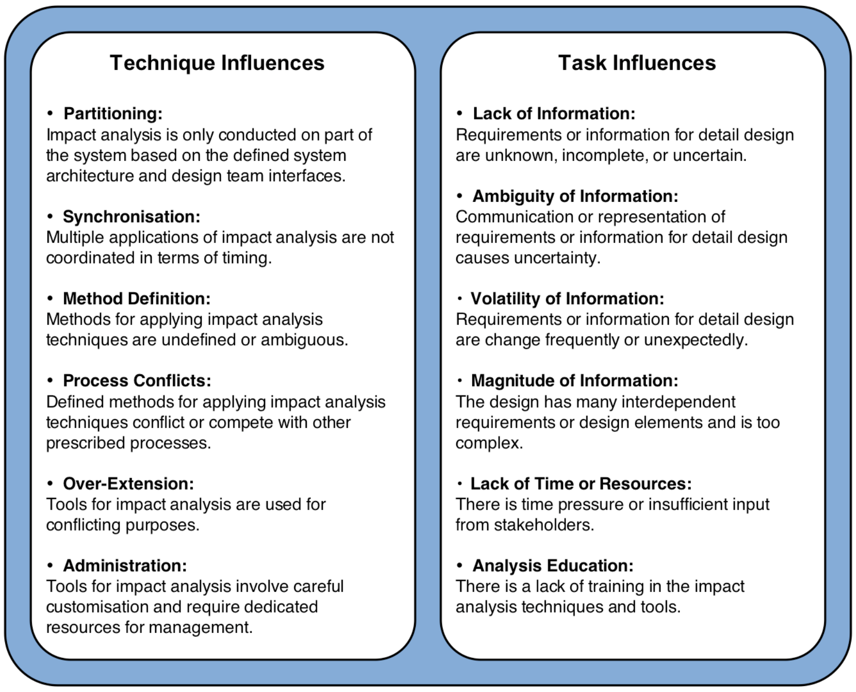

Task Influences
Similar codes are grouped to form sub-themes
Sub-themes are known as IA influences
Information from past experience
Uncertain or unproven information
Incomplete information
Assuming requirements
Lack of Information
Influence - Lack of information
Ambiguity of Information
Detail of information
Communication or representation of information
Potential for human error
Influence - Ambiguity of information
Volatility of Information
Influence - Volatility of information
Error causes information update
Multiple pieces of information changing
Magnitude of Information
Interconnectivity of development tasks
Interconnectivity of properties
Interconnectivity of functions
Interconnectivity within a function
Number of changes
Influence - Magnitude of information
Lack of Time or Resources
Allocation of resources by project
Time pressure
Not enough people review design
Involvement of stakeholders missing
Influence - Lack of resources
Analysis Education
Requirement engineering training
Requirement management training
Influence - Impact analysis education

Figure 4.6: Code map of IA task influences


Technique Influences

- **Partitioning:**
Impact analysis is only conducted on part of the system based on the defined system architecture and design team interfaces.
- **Synchronisation:**
Multiple applications of impact analysis are not coordinated in terms of timing.
- **Method Definition:**
Methods for applying impact analysis techniques are undefined or ambiguous.
- **Process Conflicts:**
Defined methods for applying impact analysis techniques conflict or compete with other prescribed processes.
- **Over-Extension:**
Tools for impact analysis are used for conflicting purposes.
- **Administration:**
Tools for impact analysis involve careful customisation and require dedicated resources for management.

Task Influences

- **Lack of Information:**
Requirements or information for detail design are unknown, incomplete, or uncertain.
- **Ambiguity of Information:**
Communication or representation of requirements or information for detail design causes uncertainty.
- **Volatility of Information:**
Requirements or information for detail design are change frequently or unexpectedly.
- **Magnitude of Information:**
The design has many interdependent requirements or design elements and is too complex.
- **Lack of Time or Resources:**
There is time pressure or insufficient input from stakeholders.
- **Analysis Education:**
There is a lack of training in the impact analysis techniques and tools.

Figure 4.7: Influences on IA techniques and tasks (from Figure 4.5 and Figure 4.6)

4.4.1 TECHNIQUE INFLUENCES

The following section describes the technique influences summarised in Figure 4.7 through examples from the empirical studies.

4.4.1.1 PARTITIONING

Partitioning can affect IA rigour if system or software designers cannot analyse the implications of changes on both the system and software designs; system designers may be limited in their search for potential change impacts on the software design, and vice versa. Systems and software engineers particularly may not have processes in place to effectively share information with which to perform IA on both domains (Section 2.6.1). A single designer may make decisions on design changes affecting both areas using incomplete IA results due to such constraints.

Such partitioning of a design between system and software designers can influence IA rigour if software designers begin development prior to obtaining complete requirements from systems engineers. In turn, IA may only be applied to part of the design. A software engineer (I-3) stated:

> *As we were coding, we would spot changes or have questions on the requirements and how exactly (a system designer) wanted it implemented. A lot of time (the system designer) had to do some modelling or speak to (a stakeholder). In the meantime, we went off on a path and made a decision to continue on. But, (the stakeholder) would come back, and we would have to revise. There were revisions going on all the time.*

As such, some of the coping strategies for designing under compressed development schedules can account for this partitioning (Section 3.4.3). Specifically, IA can be performed on complete, but reused designs (I-13). Such methods may mitigate the effect of design partitioning and improve IA rigour.

Within the aerospace empirical studies, some interviewees also recognised the lack of IA across the functional areas of the system design (I-15). A systems engineer (I-30) said:

> *I've got little visibility of that (effect on other system functions). Now if that information was (in the integrated software requirement model), we'd be able to pull out the links immediately of the (system) functions that we would need to consider the impact of any changes of those (system) functions, but those links don't exist at the moment.*

This designer suggests that information-sharing and updating processes restrict the rigour of the IA technique. This effect can also occur similarly between design areas in software models, causing emergent software changes (I-28, I-40). Furthermore, this interviewee implies that company procedures largely do not support IA techniques examining changes across multiple functional areas. Therefore, if an area of the system design contains elements decoupled from other functions, then performing localised IA may be sufficient. However, if the decomposition of the system design causes coupling between areas, then partitioning may have more of an influence on IA rigour.

Furthermore, partitions among system and software design teams can limit the IA search space. For instance, designers working on an instance of a product platform may not necessarily have coordinating processes to share and update information with the team enhancing and maintaining the platform (I-21). Changes may not be systematically investigated across designs and, in turn, knock-on effects across these designs may not be recognised, potentially decreasing IA rigour.

4.4.1.2 SYNCHRONISATION

S*ynchronisation* of modifications also can be difficult due to the distribution of design work among systems and software engineers. Multiple changes from different stakeholders can affect a single functional area or design work product, and, in some cases, these modifications can compound or contradict each other (Section 2.3). When such changes affect a design area, the synchronisation of these modifications determines the search thoroughness for potential knock-on effects, influencing IA rigour and the potential for rework. In turn, procedures and processes supporting the timely sharing of information, such as change packaging, can mitigate this synchronisation influence, particularly for the timing of implementing software modifications initiated by system design (I-22).

As an example of the synchronisation influence, a system designer (I-30) discussed the coordination of functional area designs, each requiring the input of the others:

> *There are too many stakeholders (system designers). There are so many people talking and so many different links. As soon as you turn your back, somebody has changed something, and you're back to square one… The only way we are going to come together is through an iterative process… There is no way we are going to get it right the first time, but there is certainly room for improvement.*

This designer indicates that many changes from different system designers require synthesis for the design to progress. Given that IA tasks performed by system designers may not account for the upcoming changes being simultaneously analysed by other system designers with dependent designs, improving information-sharing processes in terms of timing could reduce the iterations necessary and enhance IA rigour.

Similarly, the synchronisation of IA across software functionality designs and the software architecture or code structure design could also reduce rework and unexpected, emergent changes. A software engineer (I-28) explained:

> *Generally, we find that the software architecture and software design are done in parallel, and we generally find that once the (software) review has taken place that there are architecture changes. And, then there is a knock-on effect on the (software) design. We try and cut time scales… Of course, there are associated risks with that.*

As such, this particular influence of concurrent development can be addressed through procedures to improve participation in design reviews to share critical information. Design reviews can find design faults and incongruence early during the design process and provide means to increase the thoroughness of the search for potential change propagation. Furthermore, capturing such information during reviews through documenting design rationale can improve later applications of IA (Section 2.5.3).

4.4.1.3 METHOD DEFINITION

Method definitions, describing procedures for implementing IA techniques within a design process, may not exist or be well defined in some cases. Fundamentally, if designers do not delineate or agree on a process for performing IA, IA techniques may not be used in practice. Providing procedures to implement IA techniques increases the likelihood of their application and, in turn, improves the search thoroughness for potential change impacts and increases IA rigour.

In the aerospace empirical studies, defining methods to allow for the implementation of IA techniques was elicited as an area for improvement. For instance, in the context of describing IA through requirement traceability, a systems designer (I-22) said:

> *The other thing that we are generally bad at or poor at is defining processes early on in the project and being strong and forceful in enforcing those processes. We generally just have very woolly processes, and we don't think about those things like traceability… We just let the mass of people kind of build and tease their own way to get there.*

In order to promote the application of such IA by designers, procedures for using IA techniques and tools must be defined and integrated into the design process. Another engineer (I-19) confirmed the lack of a defined process for implementing requirement traceability:

> *I found out about the (requirement traceability IA) tool yesterday… If I don't know what the capability of the tool is; if I am not educated on part of the tool; if nobody makes it part of the process; if I run the tool and it throws up lots of errors (when performing IA), then what do I do? Ideally we would have all the time in the world to fix it, but the reality is we don't. So, we don't run the tool.*

As such, this designer indicates that the lack of a defined method to perform this traceability IA has lead to the disuse of the technique. In contrast, on another project within the company, an interviewee (I-16) reported that the same traceability tool was used successfully because they defined an implementation procedure and stated:

> *I know that the experience (on a specific project) has been very good because they know what they want to do and have set what the process is going to be for managing requirements before implementing the tool.*

Thus, the definition of a method to apply IA within a design process influences the use of IA techniques in practice.

Similarly, the definition of a process to implement IA should also be unambiguous in order to promote application. For instance, an engineer (I-15) reported on different methods for updating a data dictionary with design information. Some designers revised the information in real-time, while others updated stale information only after major design iterations. The ambiguity in this update process affects the consistency of the information within this tool, and, consequently, the search thoroughness for potential change impacts and IA rigour across its application.

4.4.1.4 PROCESS CONFLICTS

Process conflicts occur when defined methods for implementing IA contradict other prescribed procedures or compete with applying other IA techniques. In a couple of instances in the empirical studies, company prescribed procedures or policies influenced IA rigour. For example, configuration management procedures inhibited traceability IA rigour, as described by a manager (I-23) and confirmed by another interviewee (I-39). Prescribed configuration management processes required that only formally accepted changes could be entered into a traceability database. Hence, the traceability relationships captured were not always up-to-date since change authorization can require a significant amount of time and the design may have progressed in the meantime. The actual traceability relationships may differ from the captured relationships in the database, potentially causing low IA rigour when using the database. Similarly, delays in updating models or code used in dependency IA can affect IA rigour. As such, designers may lack confidence in such IA techniques and their results, leading to the disuse of these techniques.

In another instance, the company policy of resource allocation for design projects inhibits effective design reviews and potentially other forms of experiential IA. A systems engineer (I-19) explained:

> *There is an inherent problem. The problem is that, if you look at a V-diagram, there is a systems job passed to hardware and software, which breaks down into a smaller V-cycle. Then, they have their integration and testing, and we have our systems test up here. If you want to flatten that out into a line and if you were (a manager), you wouldn't resource these people (testing and integration) earlier, which leaves the problem – who does the review?*

If testing, verification, and validation team members are not hired or allocated to a project and, in effect, do not review the requirements and design, system and software design work products may need to be revised unexpectedly upon later IA. Without testing, verification, and validation stakeholders present to perform experiential IA, the thoroughness of the search for change impacts is compromised. Thus, the policy of only allocating these resources later in the design process limits the rigour of IA performed early in development.

Different IA techniques may also compete with each other. Designers may have multiple means or tools to perform IA, causing a conflict in when to apply the techniques available. A systems designer (I-13) described the future plans for integrating the information among IA tools:

> *The dependencies between requirements will be handled in (a dependency tool) models… Whilst this helps in some way, that can arguably be another level of complication because you have your requirements stated in (the dependency tool) models; You have them stated in English in (a traceability tool); Then, in some sense, you have them represented in (another dependency tool). You could arguably have three things. Now how you ensure consistency between those things is a big issue.*

Storing traceability information in several tools can allow for a more complete analysis by converging different representations and providing an increase in traceability and dependency IA rigour. However, multiple methods of implementing traceability and dependency IA can also obfuscate the appropriate or best means of IA. Designers often may choose their favourite IA technique, leading to disuse of the alternative IA techniques. Their preferred method of implementing IA, therefore, limits their application of other IA techniques. In addition, if the information is not consistent across IA techniques, only using their preferred IA can limit IA rigour.

Furthermore, short-cut procedures in the design process to speed-up product development also influence IA rigour. These procedures can conflict with idealised processes for implementing IA. For instance, tacit acceptance of procedures or even management strategies pressuring designers only to update traceability relationships and requirements after "hard" work products, such as models and code, have been completed affect IA rigour (I-14, I-29, and I-40). Without well-maintained information available for IA techniques, IA rigour decreases.

4.4.1.5 OVER-EXTENSION

Over-extension of tools, or using tools for conflicting purposes, also can systematically affect IA rigour. For example, the influence of configuration management on performing traceability IA (mentioned in the *process conflicts* influence, Section 4.4.1.4) is also affected by the tool implementation for managing traceability relationships. The procedure for capturing and managing the configuration of design traceability within one database tool, in effect, influences IA rigour. Alternative processes to maintain these dependencies in a separate tool or database in real-time and without configuration management could potentially increase IA rigour.

In another example from the aerospace empirical studies, the tool implementation of the integrated software requirement model (Table 3.1) required design updates from the software requirement models of functional areas. A systems designer (I-4) explained:

> *The problem is that the (integrated software requirement) model is only as good as the rest of the system. Your (integrated software requirement) model is built up from all of the models you have of the whole system… the problem is at the moment that the (integrated software requirement) model is always out-of-date. So, you actually want something that you have today and it takes a finite period of time that it is caught up with reality.*

Given that the integrated model worked with stale information from the functional area models, the IA may not completely search for potential change impacts, leading to its disuse by designers. As opposed to another tool implementation of such an integrated model, which worked well in a different project (I-16), the extension of the functional-area modelling tool from its original purpose limits IA rigour. Thus, defining multiple purposes for tools used for IA can influence IA rigour.

4.4.1.6 ADMINISTRATION

Administration of some tools to implement IA techniques may require substantial effort. Tools may involve customisation or dedicated resources for constant maintenance in order for use. In these instances, the overhead cost of the tools may outweigh the usefulness of the IA, leading to disuse of the tools and IA techniques (I-39). For example, a system designer (I-17) said:

> *We would have to do a lot of rework to use the tool. It's just not cost-effective. So, we just left it behind. (The requirement traceability IA tool) does not have much benefit unless you have big teams and are doing a lot of change. We don't have a big team, and we don't do a lot of change. You can just use (a spreadsheet), and it's a lot cheaper.*

In this case, the IA tool was abandoned because of the high maintenance cost and replaced with a more cost-effective means of performing a similar analysis. The administration procedures of this tool limited its value and, thus, implementation. System and software designers (I-3 and I-15) also suggested that administration procedures of tools can limit the access to IA techniques. For instance, a lack of software tool licenses, due to their high cost, may reduce the application of the associated IA techniques.

In summary, technique influences can systematically affect IA rigour (*e.g.* partitioning, synchronisation, process conflicts, and over-extension) and preclude the implementations of IA techniques (*e.g.* method definition and administration). In turn, the quality of IA results can also be influenced.

4.4.2 TASK INFLUENCES

The following section details the task influences summarised in Figure 4.7 through examples from the empirical studies.

4.4.2.1 LACK OF INFORMATION

Lack of information, regarding changing work products, was cited by designers to complicate implementing traceability, dependency, and experiential IA. Without sufficient requirements or detail design information, many different change impacts are possible since the design modification is essentially undefined. In turn, this uncertainty of the scope of the change can lead to poor quality IA results.

Designers (I-4, I-13, and I-40) suggest that lack of information about requirements can be due to customers or stakeholders not knowing exactly what they want. A lack of information can also occur from within the design team in that system and software designers may not share detail design information with each other either unknowingly or not in a timely manner. A system designer (I-13) stated: “So, you can effectively end up fire fighting the entire time because the wrong thing is there or you are working with incomplete data. Everything is in a state of flux”. This engineer suggests that a lack of information can cause IA tasks to produce low-quality results, leading to emergent changes.

The response to this lack of information in terms of IA can vary. For example, designers may wait for more information about the requirements before beginning any design work (1-17) and only perform IA when the relevant information becomes available in order to ensure sufficient IA result quality. Alternatively, if work begins before the requirements are finalised due to scheduling constraints, the designers may reuse old design work products and develop multiple potential design solutions (Section 3.4.3). These strategies attempt to compensate for the lack of information and effectively manage of changes once information becomes available. In this case, IA results may be more thorough given that they are grounded in requirements, models, and code already developed.

4.4.2.2 AMBIGUITY OF INFORMATION

Ambiguity of information in requirements or detail design work products can cause similar difficulties in performing IA since the areas affected by the modification are difficult to pinpoint based on the information represented. If IA is implemented, the results may be incomplete or unclear. For instance, ambiguous requirements can cause traceability IA to yield meaningless or unhelpful results. A systems engineer (I-17) stated:

> *They (requirements) are not detailed enough… There are extensions to requirements at lower levels. You can’t tell until you look at the code… As a systems engineer, you tend to get a feeling of what you can believe and where you have to go to the code… What we call ‘woolly’ requirements.*

If requirements are written at a very high level, which may be appropriate since requirements should not necessarily contain all design information, they may not clearly and unambiguously reflect the detail design dependencies. As such, manually performing traceability IA can miss indirect design dependencies, affecting the IA results obtained.

The empirical studies also indicate that ambiguous information represented in detail design work products can influence IA result quality. A software engineer (I-12) cited that software design models including ambiguous variable names often can mislead designers and cause inaccurate IA results; alternatively, multiple, different representations of information within a single model or across several models of a single design can be deceptive. In addition, designers (I-2 and I-3) indicated that some ambiguities were often caused by human error. Typos in models or code, oversight during design reviews, or forgetting to update information in databases can all lead to degradation in IA quality.

Moreover, ambiguous information communicated between designers during meetings has caused inaccurate experiential IA (I-3). In several cases, designers have implemented a course of work perceived as agreed-on during an integrated product team meeting. However, no meeting minutes or further communication ensued, and, once completed, the detail design work did not exactly meet the other's expectations. Thus, IA results can be highly dependent on the clarity of information available.

4.4.2.3 VOLATILITY OF INFORMATION

Volatility of information, due to frequent or unanticipated changes to requirements or detail design information, also can affect the quality of IA results. A system designer (I-13) stated:

> *I think a key attribute for requirements ought to be 'volatility' or 'likelihood of change' or something. There are so many things that are dependent… Since we're designing something that's such a complicated beast… it should be acceptable to have uncertainty.*

Similarly, another systems engineer (I-12) noted that frequent changes to information used as a input to the software detail design most often occurred in highly coupled functional areas and described these areas as "volatile". As such, the current state of system or software designs, given the uncertainty of information used as a basis for requirements and detail design, may be difficult to track by designers. In turn, designers must perform traceability, dependency, and experiential IA by considering the compound impact of many changes with the anticipation of further, unexpected modifications, and, given this context of information volatility, these IA techniques can produce low quality of results.

Volatility due to sudden, unexpected changes also can produce inaccurate IA results. As opposed to many modifications in highly coupled design areas, a relatively independent design can be subject to a single, unanticipated change that requires significant rework. A designer (I-5) remarked on the high number of "late changes" requiring such rework. In effect, IA tasks may not identify the full scope of the modifications necessary due to the uncertainty of the information provided and produce low-quality IA results.

4.4.2.4 MAGNITUDE OF INFORMATION

Magnitude of information, in contrast to the *lack of information* influence discussed (Section 4.4.2.1), was pointed to by designers as a factor decreasing the IA result quality. Given many interdependent work products or design functional areas, the search space for possible change impacts can be large for a single initiating change. Even with the help of automated IA tools, a designer may not be able to rigorously search through all of the dependencies within a reasonable amount of time (I-3, I-40). The knock-on effects of making a change can be difficult to identify in this case, and, consequently, the IA results may not be complete.

4.4.2.5 LACK OF TIME OR RESOURCES

Lack of time or resources, in terms of a lack of time to do a design change or people to perform or have input into a design change, also can affect the quality of IA results. The time pressure of completing a design can cause a rush to simply implement changes without performing any IA or only quickly using engineering judgement and intuition. A systems designer (I-15) said:

> *We have a bit of a tendency to run and go fix it and then go screw something else up. We don't understand the way that a high-level requirement fans out or how it horizontally impacts other components. So, it's probably fair to say that we go - 'blink' - I've got a fix.*

Without rigorously implementing IA, knock-on effects can be missed. Thus, the need to quickly produce design work under time pressure can lead to poor IA quality.

Similarly, the number of changes occurring can also influence the quality of IA results. A system designer (I-17) stated: "If you've got 100 (changes) to work with, things get through. They are bound to get through". A large number of changes can cause designers to be under time pressure to perform IA. In such a situation, IA may not be as rigorously applied and not account for all of the knock-on changes, producing inaccurate results.

Other designers (I-14, I-29, and I-40) discussed the time pressure experienced during the design process and indicated having to choose between producing models and code as opposed to other "soft" design artefacts, such as documentation (as cited within the *process conflicts* influence in Section 4.4.1.4). This focus on models and code can cause a scarcity of other design artefacts used for IA. Thus, traceability IA, specifically, may be of low quality due to these artefacts being incomplete or out-of-date.

Similarly, designers indicated that a lack access to stakeholders limits the IA results. Without understanding the perspectives of all stakeholders on a design change, any IA performed may be incomplete. In turn, documenting the links between arguments in design rationale and the concerns of different stakeholders can improve these IA results. Nevertheless, if stakeholders are not frequently included in design change decisions or their design rationale is not captured (I-13 and I-19), then knock-on effects can be easily overlooked. Another designer (I-30) stated:

> *One of the biggest failings, I believe, is that we have lost a lot of person-to-person communication with the customers. For me, it is absolutely paramount to understand what the customer wants. You don't get that through communicating via email or the web and so on. When you sit down with a real person on a table, you come to an understanding. You tend to improve your understanding and gain agreement on what you're doing.*

Gaining a mutual understanding of design changes can help the stakeholders individually assess the knock-on changes to their design areas, potentially producing higher quality IA results.

4.4.2.6 ANALYSIS EDUCATION

Analysis education, or training on IA techniques or tools, is a potential source for poor IA result quality. Training may not be delivered at all or not in a timely manner. In the aerospace empirical studies, a designer (I-19) did not know that the traceability IA tool existed and generally commented on the lack of training on requirements engineering, and other designers did not understand IA using data dictionaries (Section 3.4.4). Without educating designers on the IA techniques available, these techniques cannot be implemented in practice. In turn, IA may not be performed across some design areas as rigorously as possible, affecting the IA results obtained.

Similarly, a lack of education on processes related to implementing IA techniques can cause a reduction in IA quality. For instance, if designers do not understand traceability management procedures, then they may not update the traceability information in a timely manner. In turn, other designers may perform IA using stale traceability links, producing incomplete IA results. In one case at the aerospace company, designers had to stop all design work for a couple of weeks in order to input all traceability information into a database because not all of the designers had been updating the traceability database while designing (I-22). In effect, the lack of education on implementing appropriate processes for IA can potentially lead to incomplete and inaccurate IA results.

In summary, despite these factors that can decrease the quality of IA results, task influences do not always preclude the implementation of some type of IA. IA can still be performed even with imprecise or some unavailable information. For instance, experiential IA does not require significant resources or training. The perception that these influences halt the implementation of some IA techniques is not always the case; there may only be degradation of result quality.

4.5 EVALUATION OF THE IMPACT ANALYSIS CHARACTERISATIONS

Theories developed from empirical studies, such as the IA characterisations presented, are difficult to prove applicable in other contexts. In order to substantiate the IA characterisations developed from the empirical studies, the characterisations are compared with literature published on software development practices (Section 4.5.1), and the responses from 3 interviewees (I-2, I-16, I-39) asked to comment on the characterisations are presented (Section 4.5.2). This later approach, known as *member validation*, can be applied within the grounded theory research methodology used (Section 3.1) and relies on the experience of interviewees to judge the "adequacy" of the characterisations (Bloor 1997).

4.5.1 COMPARISON OF THE CHARACTERISATIONS WITH LITERATURE

The IA characterisations developed correspond with literature published on IA and software practice. The characterisation differentiating IA techniques and tasks (Figure 4.1) and the IA quality (Figure 4.2) and rigour (Figure 4.3 and Figure 4.4) characterisations depict the variation in prescribed and practised IA denoted in the literature review (Section 2.5 and Section 2.6). Although literature focusing on IA does not describe the breadth of reasons for the difference between prescribed and practised IA, as done by the IA influences characterisation (Figure 4.7), literature highlighting factors affecting software development in general corresponds with this classification. Table 4.1 and Table 4.2 introduce and compare such literature with the IA technique and task influences, respectively. As such, software literature supports the IA characterisations and suggests that the IA influences characterisation embodies general properties typically affecting software development practice.

Table 4.1: Mapping of literature to the IA technique influences characterisation

Technique Influence	References	Mapping to Characterisation
Partitioning	System and software architecture literature highlights the decomposition or partitioning of designs. Developing modular designs essentially provides for maintenance as changes can be isolated to design areas and propagation can be reduced (Maier and Rechtin 2000).	This literature suggests design partitioning affects IA in that appropriately decomposed designs can reduce latent design dependencies, which IA can miss. Furthermore, the partitioning of teams influences the information available for design work as well as IA.
	de Micheli and Gupta (1997) note that system design partitioning affects design team operation and cite that communication of information between designers highly affects the design process.	
	Strens and Sugden (1996) discuss improving impact analysis though the effective sharing of information between designers.	
	Beecham *et al.* (2003) characterise the partitioning of design teams as a barrier to process improvement since information is not effectively communicated.	
Synchronisation	de Micheli and Gupta (1997) highlight that the scheduling of tasks performed by designers, given the design information available for input to these tasks, must be taken into account during design partitioning.	Synchronisation, just as partitioning, influences IA given that IA is dependent on information provided by others.
	Sengupta and Abdel-Hamid (1996) assess the dynamics of sharing information in software projects. They indicate that having enough information available to make decisions can allow for process improvement.	
Method Definition	Ramesh (1998) cites *ad hoc* processes as an impeding characteristic for implementing requirement traceability.	This literature indicates that method definition can allow for prescribed procedures to be implemented in practice. Thus, a lack of process definition to implement IA can affect IA in practice.
	Rainer and Hall (2003) indicate procedure definition as factor that provides for process improvement.	
	Olson (2006) emphasises that defining useable processes and allowing access to process documentation enables prescribed processes to be implemented.	
Process Conflicts	Ramesh (1998) discusses conflicts in performing requirement traceability when also used for performance appraisal in that the traces captured can have a degradation in quality.	Process conflicts can take a variety of forms, but similarly affect the processes used in practice. The processes applied to implement IA are no exception.
	Bush (2004) indicates that organisational processes can conflict with problem or error detection through peer reviews. Organisational processes specifically can blame designers, which leads to a conflict when performing reviews.	
Over-Extension	Ramesh (1998) characterises the compatibility of tools as a means to improve requirement traceability practices.	This literature suggests additional examples of over-extension and its influence on design practice. IA tools are another such instance.
	Olson (2006) discusses how process documentation and tools for using such documentation can be over-extended for too many purposes, leading to disuse.	
Administration	Ramesh (1998) cites treating requirement traceability as overhead costs rather than necessary costs as a barrier to implementing requirement traceability.	Administration of tools used to implement process improvement strategies similarly can be a barrier to IA. Not supporting process initiatives through the necessary administration can dissolve the potential for improvement.
	Beecham *et al.* (2003) characterise the implementation of tools and technology as a barrier to software process improvement initiatives, and discuss how keeping tools used in process improvement strategies up-to-date requires resources.	

Table 4.2: Mapping of literature to the IA task influences characterisation

Task Influence	References	Mapping to Characterisation
Lack of Information	George and Bohner (2004) indicate that dependencies between design elements may not be captured and, in turn, can affect IA results.	Lack of information can influence the ability to manage change and, more specifically, affect IA results through latent, unrecognised dependencies.
	Stephenson and McDermid (2005) characterise no information as well as incomplete and inconsistent information as factors affecting the ability to cope with requirement change.	
	Kettunen (2003) highlights the importance of capturing software design information across project teams and describes its impact on managing change.	
Ambiguity of Information	George and Bohner (2004) suggest that dependencies may be initially ambiguous and become clearer as the design develops, influencing IA results.	This literature implies that ambiguous information can also affect change management and IA using unclear dependencies.
	Weinberg (2003) notes that an information storage infrastructure can improve the understanding of design information and develop the foundation for change management.	
	Stephenson and McDermid (2005) characterise ambiguous information as an influence on the ability to cope with requirement change.	
Volatility of Information	Takahashi and Kamayachi (1989) conclude from empirical studies that the frequency of changes suggests the quantity of software errors made.	This literature suggests that volatile information can affect IA results in that more errors (*i.e.* missed changes) may occur. In turn, volatile information and its influence on IA can lead to higher costs and longer process durations.
	Curtis *et al.* (1988) discuss frequent requirement changes influencing design process cost and duration.	
Magnitude of Information	Weinberg (2003) indicates that information storage infrastructures allows designers to cope with large quantities of information used for change management.	Providing means to manage large quantities of information allows for effective change management. In turn, IA can be improved through managing and interpreting large magnitudes of information to determine relevant dependencies.
	Stephenson and McDermid (2005) suggest that too much information, as a factor, can affect the ability to cope with requirement change.	
Lack of Time or Resources	Baddoo and Hall (2003) characterise time pressure and the lack of resources as factors detrimental to process improvement.	Lack of time or resources can affect prescribed strategies for IA to mitigate unexpected changes similar to how other process improvement initiatives can be affected by these factors.
	Humphrey *et al.* (2007) indicate that providing the time and resources for process improvement is essential to enable such strategies. Designers must participate to enact strategies in order for process improvement to occur.	
Analysis Education	Rainer and Hall (2003) classify training as a factor affecting process improvement initiatives.	Training or providing education to designers allows for the realisation of process improvement strategies. As such, educating designers on IA can influence the implementation of IA in practice.
	Humphrey *et al.* (2007) stress that implementing process improvement strategies requires staff training.	
	Beecham *et al.* (2003) characterise the training of designers as a barrier to process improvement initiatives.	

4.5.2 EVALUATION OF THE CHARACTERISATIONS BY INTERVIEWEES

As discussed by Bloor (1997), member validation is not without faults. The perspective of participants, which can change over time, can highly affect their response to the theories presented (*i.e.* the IA characterisations). Bloor suggests viewing responses, which are produced by different methods than the data collection process used to develop the theories (Section 3.1), as additional data points that can support the theories constructed. As such, member validation is a form of evaluation rather than a rigorous approach to validation.

Nonetheless, as discussed by Bacharach (1989) and Patton (2002: 577-581), eliciting interviewees' responses to theories can indicate their *falsehood* and *utility*. Patton (2002: 581-584) also suggests that interviewees can suggest their *transferability*, extrapolating from theories presented to support their generality. In turn, the interviewees were prompted to comment on the IA characterisations using these criteria. Specifically, interviewees were asked to respond verbally to the following questions about the characterisations (Figure 4.1, Figure 4.2, Figure 4.3, Figure 4.4, and Figure 4.7):

- **Falsehood:** Do the characterisations reflect the state of IA? If so, why do you agree? If not, what other contexts or attributes are not captured?
- **Utility:** Do the characterisations explain the state of IA? Do the characterisations provide reasonable predictions for the state of IA in other contexts? If so, why do you agree? If not, how should they be modified?[28]
- **Transferability:** Do the characterisations apply to other contexts or attributes of IA not mentioned? If so, describe these other contexts or attributes. If not, why are they specific to the empirical studies conducted?

Three evaluators (I-2, I-16, I-39), a manager and two process engineers, were selected from the aerospace and telecommunications companies to reflect on the characterisations. These interviewees were chosen because of their overview of the system and software design processes implemented in their respective companies as well as experience managing process improvement strategies. The manager holds a senior-level role in the aerospace company's control group, while the process engineers interviewed drive process improvement initiatives at each of the companies. This selection of interviewees also allowed the results of the empirical studies to be delivered to the appropriate individuals in the companies with the capabilities to potentially induce changes in these organisations. The following sections discuss the responses of these evaluators to the characterisations.

[28] The interpretation of utility in terms of explaining and predicting practice is derived from Bacharach (1989).

4.5.2.1 FALSEHOOD

All interviewees (I-2, I-16, I-39) agreed with the notion that IA influences (Figure 4.7) account for the difference between IA techniques and tasks (Figure 4.1), affecting the IA quality of results (Figure 4.2) and rigour (Figure 4.3 and Figure 4.4), and focused on scrutinising the IA influences characterisation. At the aerospace company, the evaluators (I-2, I-16) indicated that the IA influences characterisation is a good summary of issues affecting IA. For instance, the process engineer (I-16) gave several examples from his work supporting the IA influences characterisation. He stated: "I'm surprised at how many of these (IA influences) line up with what we've been doing for (an internal process improvement project)". This interviewee also commented that he believed that "lack of experience and wrong attitude" were the primary barriers in getting project teams to implement IA techniques. The project teams cannot envision the potential benefits of implementing IA techniques and viewed them as burdens on resources. Based on the conversation, these attributes can also be interpreted through the IA influences characterisation as *analysis education* and *administration*, respectively.

At the telecommunications company, the process engineer evaluator (I-39) stated: "All of these things (IA influences) resonate with (the telecommunications company)". This interviewee described the recognition by the telecommunication company the need to constantly and quickly implement design changes; finding and scoping necessary changes are central to their design projects. As such, he indicated that several of the IA influences characterised (Figure 4.7) have directly driven the shift to implementing agile development processes. Specifically, their agile processes target the *partitioning*, *lack of information*, *ambiguity of information*, and *lack of time and resources* influences.

In turn, the evaluation of the characterisations by the interviewees against the falsehood criteria suggests that the characterisations reflect IA practice and correspond with their experience on implementing process improvement initiatives.

4.5.2.2 UTILITY

For the IA influences characterisation (Figure 4.7), the interviewees from the aerospace company (I-2, I-16) agreed that these issues were useful in highlighting and predicting barriers to implementing IA in practice. The manager (I-2) noted his interest in the *over-extension* of tools influence. He discussed the over-extension of the requirements traceability tool (Table 3.1) in the past and reminded himself during the interview to discuss the use of IA tools with a colleague. As such, the presentation of the IA influences characterisation provided a useful reminder of potential difficulties in applying IA. The manager also discussed the *administration* influence and commented: "The appreciation of project managers for impact analysis is an optional thing rather than part of the project". In turn, he indicated that he believed that administration was the most significant barrier to improving IA practice in the aerospace company and suggested that it was useful to have the breadth of IA influences co-located.

At the telecommunications company, the process engineer interviewee (I-39) indicated the predictive ability of the IA influences characterisation and said: "I think they all fit perfectly and resonate with what I have found in the past". This evaluator also discussed the utility of viewing process improvement through an IA perspective. He stated: "(The characterisations are) interesting stuff – a different way of digging into the things we're trying to resolve and change the way we are working in the company at the moment. It is nice to see it from a different angle".

Although the interviewees (I-2, I-16, I-39) agreed with the notion that different IA techniques can be implemented more or less rigorously depending on context and can deliver IA results of varying quality levels (Figure 4.2 and Figure 4.3), they all discussed usefulness of determining the optimal set of IA techniques to product high quality IA results based on problem type or change characteristics. For instance, certain types of experiential IA may be more suitable for some changes, while traceability IA through requirement dependencies may produce higher quality results for others. Variations of the scale in Figure 4.4 could be developed for these different contexts. The aerospace company interviewees (I-2, I-16) suggested that further investigation of this point would require in-depth analysis of the IA used across each functional design area of their product platform. Different functional areas tend to experience different types of changes and volatility levels. As such, guidelines could be developed for designers of specific functional areas. Section 9.3 discusses this extension of the characterisation as future work.

Consequently, the interviewees suggest that the characterisations are a useful perspective on implementing process improvement and can serve as a reminder for or predict potential limitations for implementing IA. Moreover, the rigour characterisation (Figure 4.4) provides a framework to map IA implementation policies given a problem type or change characteristics.

4.5.2.3 TRANSFERABILITY

The evaluators proposed that the categories of the IA influences characterisation also encompass additional factors not specifically discussed during the empirical studies. The manager from the aerospace company (I-2) suggested that "over-optimism" as a cause of the *administration* influence. Project managers often do not acknowledge the extent of rework necessary during development and, thus, may not allocate effort into implementing and managing IA tools. Similarly, the evaluator from the telecommunications company (I-39) indicated that the difference between cultures as a factor within *partitioning*. From his experience in outsourcing software development overseas, he noted that designers from other cultures do not always openly share information about upcoming design changes or problems they discover and only discuss such issues when they are later acknowledged by others. As such, early IA often misses the knock-on effects to these unannounced problems. In turn, the IA influences characterisation appears transferable to other contexts and can accommodate other specific factors to IA within software development.

In addition, the interviewees (I-2, I-16, I-39) implied that the IA characterisations also apply beyond the systems engineering and software design domains. In particular, the evaluators from the aerospace company (I-2, I-16) suggested that the disparity between IA techniques and tasks also occurs within the mechanical and hardware design processes for their product line and that the influences characterisation is similarly appropriate. However, they noted that these other design areas do not catalogue or analyse change information as rigorously as done within the software design. The design process engineer (I-16) described beginning work to share means to capture change information with these other design teams. This interviewee also proposed that he did not think the characterisations are limited to the products produced by the aerospace company. The evaluator from the telecommunications company (I-39) likewise stated: "I would imagine these would apply to manufacturing or production". Hence, this evaluation suggests that the IA characterisations encompass generic properties of IA applied across a variety of design processes.

4.6 SUMMARY

The IA characterisations describe the disparity between prescribed and practised IA. The introduction of the characterisation nomenclatures of IA techniques, tasks, and quality emphasises this difference. In turn, the IA rigour characterisation builds on these definitions and classifies traceability, dependency, and experiential IA techniques in terms of their search space for identifying potential change impacts.

Finally, the IA influences describe reasons for why prescribed and practised IA does not yield the same rigour and result quality. Although these characterisations are fundamentally based on the empirical studies, they correspond with literature on software development practice and conform to the perspectives of experts interviewed. Chapter 5 uses these characterisations as a basis to further investigate IA practice and improvement strategies specifically within the aerospace company.

The IA characterisations also address research questions 2 (Figure 1.8) and 3 (Figure 1.9). In practice, IA is recognised as a means to manage design changes within the companies participating in the empirical studies (Section 3.4.4), and a range of IA techniques and tools are supported within these firms. However, the application of IA can vary in practice and, in turn, determine the risk of unanticipated, emergent changes. The definitions of IA techniques and tasks highlight these effects, and, more specifically, the IA quality and rigour characterisations delineate the control of emergent changes through IA in practice. The search space for potential change impacts and the inputs to the IA (*e.g.* information, resources, and time) affect the IA result quality, based on the IA technique and task influences. As such, this decomposition of terminology addresses how IA affects managing emergent changes in research question 2.

In turn, the IA influences characterisation accounts for the disparity between IA techniques and tasks. As observed in the empirical studies, these influences challenge the theoretical application of IA and can cause the degradation of IA rigour and result quality. Hence, the classification of the IA influences responds to research question 3.

5 :: THE APPLICATION OF IMPACT ANALYSIS IN PRACTICE

A detailed examination of IA practice at the aerospace company was also conducted in the second phase of the empirical studies. This study aimed to investigate the range of IA techniques typically applied by the system and software designers within a project group and the quality of the associated results. As opposed to the IA characterisations developed (Chapter 4), focusing on the frequency of design changes handled using particular IA techniques depicts IA practice specifically within the aerospace company. The IA characterisations alone yield insight into how prescribed IA techniques can differ from practised IA in general. This analysis of IA practise was only conducted at the aerospace company since no access to a specific project team at the telecommunications company could be obtained.

In order to ensure that the spectrum of IA practised by the aerospace project team was covered in the empirical study, an elicitation method was developed and employed (Section 3.1.4). The IA implemented by interviewees was systematically captured by discussing specific changes handled by designers and the associated IA tasks. As such, this chapter first describes the development of this elicitation method and outlines its implementation (Section 5.1). The data collected on IA practice is then analysed, and key observations from this investigation are made and correlated with the IA characterisations (Section 5.2). Elicited IA improvement strategies are also assessed based on the IA characterisations and these observations of IA practice at the aerospace company (Section 5.3), and practical IA improvement strategies from this firm are extracted (Section 5.4). This chapter concludes the discussion of the empirical studies by revisiting the research questions (Section 5.5) and refining the fourth research question (Section 5.6) to pursue further investigation of process improvement through IA. Specifically, IA improvement strategies are evaluated against other process improvement strategies to analyse their efficacy (Chapter 7 and Chapter 8).

5.1 AN ELICITATION METHOD FOR IMPACT ANALYSIS PRACTICE

Two primary observations (Figure 5.1) during the exploratory empirical study formed the basic structure of the elicitation method. Firstly, the exploratory study indicated that some interviewees had difficulty pinpointing the IA techniques they used when prompted. Occasionally, a designer would give conflicting responses by reporting that he never used a particular technique and then later discussing his application of the very same technique. However, more typically, designers could easily recall their application of IA for specific changes. As a result, the IA study at the aerospace company employed an elicitation method to capture IA practice by discussing particular modifications and the associated IA tasks with designers.

Secondly, another observation from the exploratory study was that the IA techniques applied depend on the type of change (Section 3.4.4), which can be characterised by the roles of the stakeholder(s) and designer(s) involved in requesting and implementing the change. For instance, system designers may deal with high-level changes, while software designers may be more concerned with detail design modifications. In turn, software designers may perform different IA techniques on changes requested by system designers than other modifications from within software design. High-level changes may require software designers to perform traceability and dependency IA across multiple work products, while low-level changes may only require dependency IA within software code.

Observation 1:
Designers can easily describe their application of IA techniques for specific changes.

Observation 2:
The IA techniques applied depend on the type of change, which can be characterised by the stakeholders or designers requesting and implementing the modification.

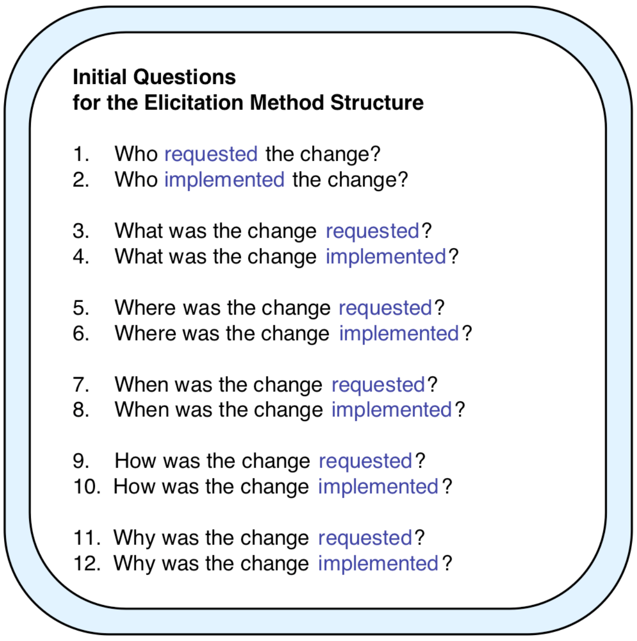
Figure 5.1: Observations forming the elicitation method structure

Consequently, the elicitation method works on the premise that eliciting a spectrum of different types of design changes and discussing the related IA techniques implemented systematically delineates the range of IA technique(s) applied in practice. As such, the elicitation method is employed by discussing many instances of changes in interview sessions with different designers. If only certain types of modifications are discussed, then the full range of IA used may not be covered.

In turn, Section 5.1.1 details the development of the elicitation method, and Section 5.1.2 discusses the classification of changes types used to cover the spectrum of IA practised. The application of the elicitation method is then described in Section 5.1.3.

5.1.1 DEVELOPMENT OF THE ELICITATION METHOD STRUCTURE

The initial structure or model for the elicitation method was formed through combining the second observation from the exploratory empirical study (Figure 5.1), acknowledging the relationship between the request and implementation of changes, with the general questions: Who? What? Where? When? How? Why? As such, answering the questions in Figure 5.2 can characterise the essential aspects of a design change, which can be correlated with the IA techniques implemented.



Initial Questions
for the Elicitation Method Structure

1. Who requested the change?
2. Who implemented the change?

3. What was the change requested?
4. What was the change implemented?

5. Where was the change requested?
6. Where was the change implemented?

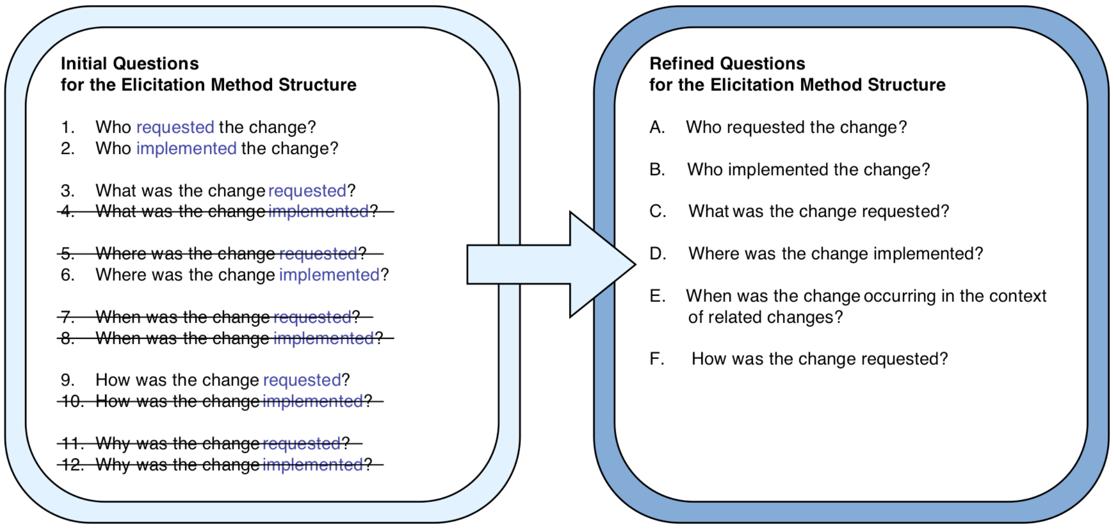
7. When was the change requested?
8. When was the change implemented?

9. How was the change requested?
10. How was the change implemented?

11. Why was the change requested?
12. Why was the change implemented?

Figure 5.2: Initial questions for the elicitation method structure

These questions were further refined or eliminated based on observations made during the exploratory empirical study, simplifying the structure for the elicitation method (Figure 5.3).



Initial Questions
for the Elicitation Method Structure

1. Who requested the change?
2. Who implemented the change?
3. What was the change requested?
4. ~~What was the change implemented?~~
5. ~~Where was the change requested?~~
6. Where was the change implemented?
7. ~~When was the change requested?~~
8. ~~When was the change implemented?~~
9. How was the change requested?
10. ~~How was the change implemented?~~
11. ~~Why was the change requested?~~
12. ~~Why was the change implemented?~~

Refined Questions
for the Elicitation Method Structure

A. Who requested the change?
B. Who implemented the change?
C. What was the change requested?
D. Where was the change implemented?
E. When was the change occurring in the context of related changes?
F. How was the change requested?

Figure 5.3: Refined questions for the elicitation method structure

The first question (*Who requested the change?*) and the second question (*Who implemented the change?*) clearly reflect the second observation from the exploratory empirical study, stated in Figure 5.1, in that system and software designers request and implement different types of changes than each other, influencing the IA performed. As such, these two questions remain in the final structure of the elicitation method.

The level at which changes are implemented (*i.e.* the work products requiring modification) also affects the IA selected. For instance, software designers use different IA techniques to modify high-level specification documentation as opposed to design models and code. In effect, the language of question 6 (*Where was the change implemented?*), as opposed to question 5 (*Where was the change requested?*), more appropriately characterises the IA applied in practice. More in-depth IA tends to occur during change implementation and uses a variety of techniques, while prior to the implementation of design changes only superficial IA may be performed to gauge the scope of the modification. Thus, question 5 is eliminated because it does not reflect the range of IA techniques applied to identify specific artefacts needing modification.

Question 3 (*What was the change requested?*) and question 6 (*Where was the change implemented?*) may be synonymous when the work products modified in a particular design area correspond to the initially expected changes (*i.e.* the work products indicated on the change request). The IA used during implementation, encompassed by question 6, should cover the range of IA applied in this case. In contrast, if a change request impacts many design areas (*i.e.* high-level product or system behaviour changes), then these two questions can differ in that question 3 can reflect the IA employed to determine the broad, knock-on effects to various detail designs (*e.g.* using integrated product team meetings to determine the consequences of changes). Hence, question 3 remains in the model structure along with question 6. However, question 4 (*What was the change implemented?*) is eliminated since it essentially reiterates question 6. The work products changed should align with the modification actually made to the product functionality or behaviour. This distinction in language does not definitively characterise IA practice.

For question 7 (*When was the change requested?*) and question 8 (*When was the change implemented?*), the timing of the request and implementation of changes are both relevant. The range of IA techniques applied can vary at different points in time based on the information available regarding other changes. If high-level changes and low-level changes are related, different types of IA may be performed depending on when the relationships are identified. Thus, these questions are re-written as: *"When was the change occurring in the context of related changes?"*

Question 9 (*How was the change requested?*) suggests that the process of requesting a change can influence the IA technique selected. Documented change requests can be implemented through formal change processes, which include review processes to ensure knock-on modifications are identified and, thereby, require the use a variety of IA techniques. In contrast, modifications occurring through informal processes may be implemented without review and may only be analysed using minimal IA (Section 3.4.2.2). Thus, the formality of change processes affects the range of IA applied. However, question 10 (*How was the change implemented?*) does not distinctly relate the nature of the design change with the IA implemented. Retaining this question creates a tautology by implying that the change implementation process affects the IA performed during implementation. Thus, question 10 is eliminated.

Finally, question 11 (*Why was the change requested?*) and question 12 (*Why was the change implemented?*) are not included in the elicitation method structure since the answers can vary greatly and are difficult to deterministically classify. Nevertheless, they should be asked during interviews in that they can provide the context of the IA implemented.

5.1.2 CHANGE ATTRIBUTES FOR IMPACT ANALYSIS TECHNIQUE ELICITATION

Questions A – F in Figure 5.3 of the elicitation method structure each indicate an *attribute* of a change, and the elicitation method includes ranges or scales for the potential answers to these questions. Accordingly, different types of changes are characterised by different permutations of answers for each attribute. Eliciting a variety of combinations of attribute ratings on these scales (*i.e.* different types of changes) can cover the range of situations in which IA techniques are practised. An initial scale for each attribute was developed after the exploratory study, and the collection of many examples of changes during the aerospace IA study supported the refinement of the scale descriptions, eliminating unessential middle categories. As such, given that the scales defined are based on observations, they are intended only to conceptualise the range of attributes, as opposed to universally define the entire scale.

Two diagrams (Figure 5.4 and Figure 5.5) visually represent the elicitation method through the defined scales of the change attributes. The elicitation method does not categorise the IA associated with certain permutations of attributes or classify IA according to change types since the IA performed depends on the IA techniques available for use; different companies have different IA methods and tools.

The first diagram for the elicitation method (Figure 5.4) responds to questions A, B, C, and D (Figure 5.3) through the attributes of change *source*, *direction*, and *level*. Questions A and B are reflected by the source and direction attributes, respectively, while questions C and D are both represented by the level attribute. The following discussion describes the scale used in the diagram.

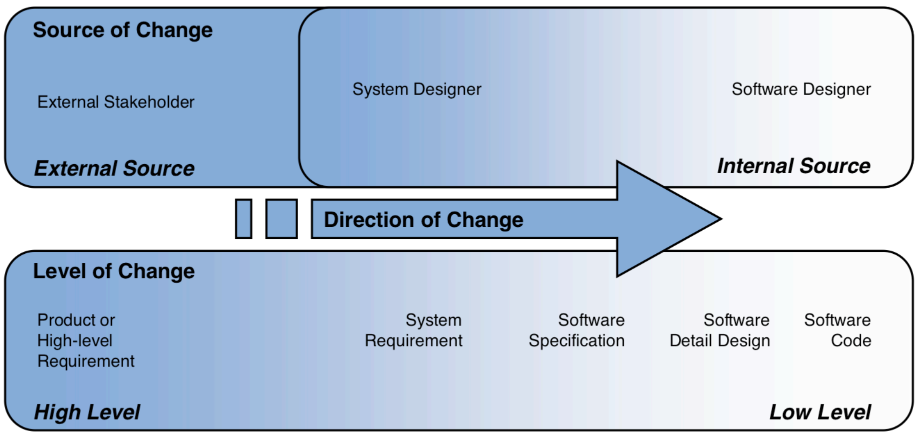


Figure 5.4: Scale for source, direction, and level attributes

Source of Change: Who requested (i.e. was the source of) the change? External and internal stakeholders were observed in the empirical studies to request changes in the system and software designs. External stakeholders may be customers, suppliers, other subsystem designers, or even business and marketing managers, while internal stakeholders primarily consisted of system and software designers. External stakeholders tended not to perform IA based on software design change requests. Instead, systems and software designers were the primary stakeholders to implement a variety of IA techniques to determine the effects of design changes. As discussed in Section 3.4.1, verification and validation teams can be considered external stakeholders from this perspective on the source attribute since they were observed to request software design changes and did not determine the causes of errors found or investigate the necessary corrective design changes.

Given that systems and software engineers tended to reflect the IA techniques applied based on their role (Section 5.1.1), the scale of the source attribute (*i.e. external stakeholder*, *system designer*, and *software designer* in Figure 5.4) highlights these stakeholders. In turn, the scale of the source attribute does not distinguish between the potential external stakeholders because they had little influence on the IA performed within the aerospace company. Furthermore, defining a generic scale can be difficult since these stakeholders can vary greatly between projects and companies. Nevertheless, interviews using the elicitation method should cover a variety changes from different external stakeholders. External stakeholders can request both high and low-level changes, and system and software designers can respond with different types of IA.

Direction of Change: Who implemented the change? Designers only identified a couple of instances in which external stakeholders performed modifications in their requirements or designs in response to a software design change. In these rare cases, the external stakeholders effectively determined the IA techniques applied for change requests. As a result, given that system and software designers select the IA techniques performed for most changes, the scale of the source attribute also applies to the change implementer and, thus, should be used for the direction attribute. However, instead of reiterating the scale of the source attribute, Figure 5.4 captures the relationship between these roles through the arrow illustrated in that changes typically are requested by external stakeholders and are implemented by system and software designers.

Level of Change: What was the change requested? Where (i.e. on what level) was the change implemented? The level attribute scale applies to both questions C and D since they both can be answered in terms of the artefacts modified in change processes. The scale selected (*i.e. product or high-level requirement*, *system requirement*, *software specification*, *software detail design*, and *software code* in Figure 5.4) is based on the range of changes observed in the empirical studies. Specifically, external stakeholders often initiated product or high-level requirement changes modifying the system behaviour, while system and software designers primarily generated lower-level changes. These lower-level changes typically involved design artefacts, such as system requirement and software specification documentation, software detail design work products (*e.g.* design models or other documentation), and software code. As such, these observations from the empirical studies also suggest a pattern between the source and level attributes. The corresponding shading in Figure 5.4 of the source and level attributes visually depicts this relationship. Furthermore, the overlap of the arrow denotes that product-level changes are often absorbed by the software design.

The second diagram for the elicitation method (Figure 5.5) addresses questions E and F (Figure 5.3) through the attributes of change *timing* and *formality*, respectively.

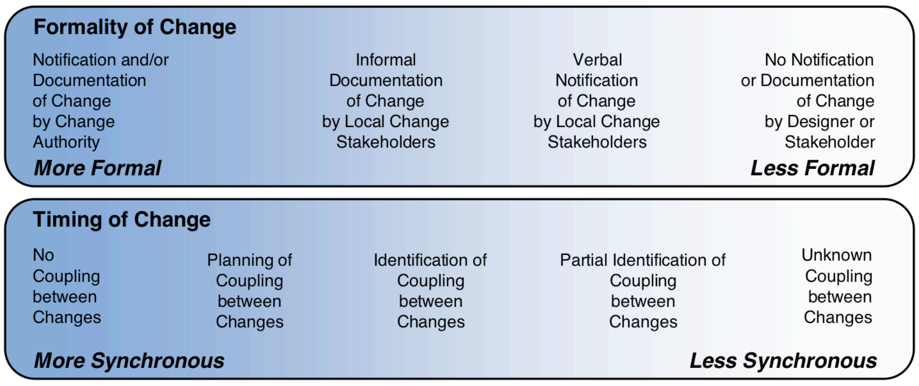

Formality of Change
Notification and/or Documentation of Change by Change Authority
Informal Documentation of Change by Local Change Stakeholders
Verbal Notification of Change by Local Change Stakeholders
No Notification or Documentation of Change by Designer or Stakeholder
More Formal
Less Formal
Timing of Change
No Coupling between Changes
Planning of Coupling between Changes
Identification of Coupling between Changes
Partial Identification of Coupling between Changes
Unknown Coupling between Changes
More Synchronous
Less Synchronous

Figure 5.5: Scale for formality and timing attributes

Formality of Change: How was the change requested? Change formality essentially refers to the process by which modifications are performed. The formality of changes acts independently from the pattern highlighted between the source, direction, and level attributes in that product-level changes as well as software detail design changes can be implemented through either formal or informal processes.

The stakeholders involved, communication, and information available in change processes reflect the formality of these process. Formal or official change requests go through administrative procedures involving selected stakeholders as means to document and communicate the changes necessary, while less formal, verbal directions from external or internal stakeholders may initiate immediate action to implement changes. In this later case, not all stakeholders may be involved and little or no documentation may initially exist, only following after implementation and reflecting the IA previously applied (Section 2.4.2).

As such, the scale in Figure 5.5 is based on the form of documentation and communication between stakeholders as observed in the empirical studies. At one extreme, accepted change requests by a CCB may have pre-defined procedures, timelines, and documentation to follow. However, informal change processes may not go through such a change authority and involve more informal documentation and communication (Section 3.4.2.2). In turn, verbal communication may be solely relied on during change processes (Section 3.4.3 and Section 3.5.3). Finally, completely *ad hoc* change process can also occur at the other extreme. In this scenario, designers or stakeholders may make changes without any documentation and may not even inform others about the modification. These later change processes not based on configuration management can influence the IA techniques used. For example, if communication is only relied upon, then only experiential IA may occur. Alternatively, a lack of documentation, in some cases, can limit the usability of traceability and dependency IA results. Thus, the formality attribute can reflect influences or barriers to certain IA techniques (Section 4.4).

Timing of Change: When was the change occurring in the context of related changes? Especially within large project teams, multiple changes often affect a single functional area, and the potential coupling or dependencies between changes must be initially identified in order for the range of relevant IA techniques to be applied. Thus, an artefact of synchronising changes and IA implementation is the identification of the coupling between changes. At one extreme, if no coupling between changes exists, they are synchronous, and the IA techniques applied are sufficient to determine the scope of the changes. If interdependencies between changes do exist, then, at best, they can be planned for and synchronised in their implementation, and the range of IA used supports identifying any knock-on effects and reduces the risk of rework. The identification of all or partial couplings without planning does not necessarily provide for implementing changes as synchronously. Implementing interdependent changes singularly only promotes performing IA specific to a change, potentially missing emergent changes and resulting in the later use of additional IA techniques. Finally, unknown dependencies between changes can also occur at another extreme with the risk of requiring additional rework and implementing further IA. Hence, the timing attribute can estimate the comprehensiveness of the spectrum of IA techniques applied to interdependent changes and also suggests the number of further design iterations and associated applications of IA required.

In theory and as indicated by the shading in Figure 5.5, more formal change processes allow for the planning of the implementation of interdependent changes. In contrast, less formal processes can lead to unknown coupling between changes. However, "no coupling between changes" is not affected by the formality of the change process and primarily depends on the partitioning of the system and software designs.

5.1.3 APPLICATION OF THE ELICITATION METHOD

Elicitation of IA practice in the aerospace IA study occurred through discussing specific instances of changes with system and software designers. These designers, as opposed to external stakeholders, were targeted since they implemented the IA tasks. The interviewees discussed changes they worked on implementing, and, thus, the interviewees can be considered experts on the details of these design modifications. During the interviews, all of the questions within the elicitation method were covered for each change discussed, and the associated IA techniques applied were noted. Designers were asked to walk through the steps to making a change to systematically cover the IA applied. The interviewer classified the responses regarding the type of change alone according to the scales in the diagrams (Figure 5.4 and Figure 5.5)[29] in order to reduce the time required of the interviewee to explain each of the scales and then perform the ratings. Given that the scales presented are intended to be notional, detailing of the scales in terms of specific external stakeholders, relevant artefacts, and change formality levels is envisioned to occur for different change processes. However, broader application of this method in additional companies has not occurred to date.

After performing several interviews, the types of changes collected with a variety of attribute permutations were analysed in terms of the range of attribute combinations possible, suggesting the breadth of the IA covered in the interviews. Given that certain attribute combinations did not occur, they were inquired about in subsequent the interviews. Specific questioning revealed many of the interviewees considered

[29] Note that the direction of change attributes should be rated according to the source of change scale, as described in Section 5.1.2.

some permutations non-existent given the structure of the aerospace company (*e.g.* software designers typically do not directly interact with or receive changes from external stakeholders), eliminating the total number of possible attribute combinations. Extreme attribute ratings for formality and timing also did not occur during the discussions and were specifically prompted for by the interviewer. However, these design changes with such extreme attributes could not be found. Not many changes occurred without any or with perfect formality or synchronisation. As such, this elicitation method presented does not necessarily ensure capturing these extreme cases, but systematically encompasses the range of IA techniques typically applied for certain change types. Without such a method, interviews may unintentionally focus on certain IA techniques rather than covering the breadth of change situations and the associated IA applied.

5.2 IMPACT ANALYSIS IN PRACTICE AT THE AEROSPACE COMPANY

The elicitation method (Section 5.1) was applied at the aerospace company during the IA empirical study, allowing for the discussion of many changes and IA tasks. Section 5.2.1 describes this information collected and also a data collection exercise performed in conjunction for rating other factors dealing with the IA applied. Based on an analysis of this data in Section 5.2.2, key observations on typical IA practice are identified and are compared with the IA characterisations (Chapter 4), leading to insights when reflecting on elicited IA improvement strategies from the aerospace company (Section 5.3).

5.2.1 DATA COLLECTION

In total, 42 change cases[30] were volunteered by interviewees as examples and discussed in detail (50% from systems engineers and 50% from software engineers). The attributes of these changes were often classified between the intervals in the scales of the elicitation diagrams (Figure 5.4 and Figure 5.5) because the changes included elements of adjacent attribute scale descriptions. In turn, each attribute was given a rating of high, medium, or low to account for this variation. The breakdown of the high-medium-low scores corresponds to the elements in the elicitation method diagrams as depicted in Figure 5.6. The division in the source, direction, and level attributes into high, medium, and low ratings is based on the roles of the system and software designers in the aerospace company and their responsibility for work

[30] These changes were discussed for a current, on-going project at the aerospace company, while the design modifications extracted from the change database (Section 6.4 and Appendix D) were from a completed project. The 42 modifications discussed occurred through informal change processes (Section 3.4.2.2) and were not yet captured within such a database. Future work (Section 9.3) could correlate the discussions of the 42 changes with the actual modifications implemented through this upcoming database.

products. The timing and formality partitioning is created based on the extremes of these attributes as elicited in the interviews.

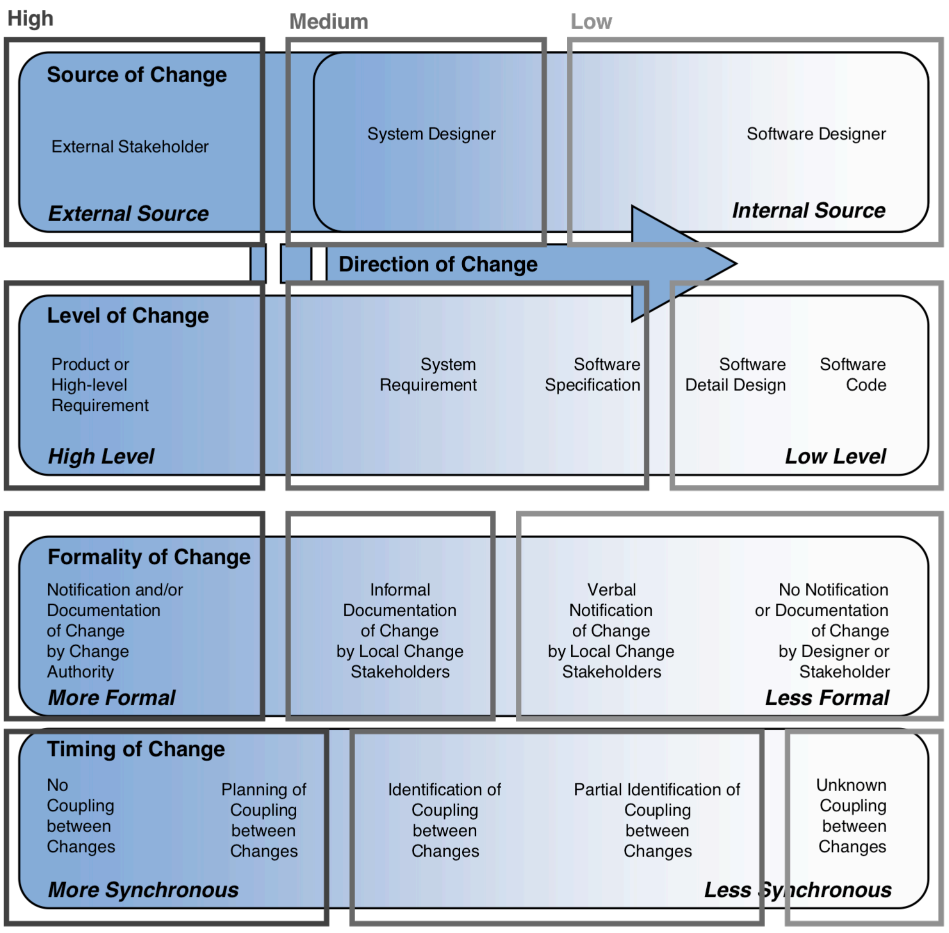

High
Medium
Low
Source of Change
External Stakeholder
System Designer
Software Designer
External Source
Internal Source
Direction of Change
Level of Change
Product or High-level Requirement
System Requirement
Software Specification
Software Detail Design
Software Code
High Level
Low Level
Formality of Change
Notification and/or Documentation of Change by Change Authority
Informal Documentation of Change by Local Change Stakeholders
Verbal Notification of Change by Local Change Stakeholders
No Notification or Documentation of Change by Designer or Stakeholder
More Formal
Less Formal
Timing of Change
No Coupling between Changes
Planning of Coupling between Changes
Identification of Coupling between Changes
Partial Identification of Coupling between Changes
Unknown Coupling between Changes
More Synchronous
Less Synchronous

Figure 5.6: High-medium-low rating scale for elicitation method as applied

As suggested in Section 5.1.3, not all of the high-medium-low combinations exist given the organisational structure between systems and software engineering. Specifically, only 4 combinations for the source, direction, and level attributes typically occurred as follows:

1. *An external stakeholder requested a systems engineer to make a system design change*
2. *A systems engineer requested a software engineer to make a software design change*
3. *A software engineer requested another software engineer to make a software design change*
4. *A software engineer requested a systems engineer to make a system design change*

Interestingly, systems engineers seldom asked other systems engineers to implement changes. This behaviour can be attributed to the functional area partitioning of the system design (Section 3.4.1). Systems engineers primarily interact only when their functional areas are coupled. Thus, this lack of interdependency can be beneficial to divide the workload. In addition, a specific type of change was found as an exception to these combinations. For variable value changes within the software code, external stakeholders directly email new values to the software engineers. While the systems engineers are also included on this email, the software designers can independently determine when these changes are implemented, virtually circumventing IA performed by systems engineers on these changes (Section 3.4.4). Interviewees also noted that the change process for variable values in the software is an anomaly (I-15, I-19).

Nine possible combinations for the formality and timing attributes exist given the high-medium-low scale. However, as discussed in Section 5.1.2, the formality and timing attributes are dependent to some extent. More formal processes tend to allow for the improved synchronisation of changes. As such, combinations of high-low ratings for these attributes were not elicited, leaving 7 permutations remaining. Yet, each of the 4 combinations for the source, direction, and level attributes elicited were not associated with all of these 7 permutations for the formality and timing attributes. Specifically, the extreme attribute ratings (*i.e.* high-high or low-low combinations) for formality and timing did not always occur, even though interviewees were questioned for these extreme combinations (Section 5.1.3). However, either extreme or near extreme ratings (*i.e.* high-medium or medium-low combinations) were given for the formality and timing attributes for all of the 4 combinations of the source, direction, and level attributes.

Figure A in Appendix B displays in detail the attribute rating results for the 42 changes described by interviewees as examples and the associated IA techniques applied. Each change is given a unique id number in the form "C-*id#*", and Figure A categorises these techniques according to the traceability, dependency, and experiential classification (Section 2.5 and Table 3.1). These findings are summarised here in Table 5.1.

Table 5.1: Summary of change attribute and IA data collected (detailed in Appendix B)

Information Elicited	Frequency of Occurrence
IA Techniques Used	The 42 changes elicited were analysed: 57% of the time through traceability IA 69% of the time through dependency IA 79% of the time through experiential IA
Specific IA Techniques Used	Traceability IA was performed: 83% of the time by manually analysing software requirement documentation 17% of the time by manually analysing software design documentation
	Dependency IA was performed: 65% of the time using software requirement models 28% of the time using software UML models 7% of the time using software code
	Experiential IA was performed: 60% of the time through informal discussions 14% of the time through formal design reviews 14% of the time through integrated product team meetings 12% of the time through individual engineering judgement
Source-Direction-Level Attribute Combinations	45% of changes were from an external stakeholder requesting a system design change 25% of changes were from a systems engineer requesting a software design change 25% of changes were from a software engineer requesting a software design change 5% of changes were from a software engineer requesting a system design change

During this empirical study, each interviewee was also asked to rate the quality of results for the IA technique or suite of IA techniques applied for a couple of specific changes, as mentioned in Section 3.1.4. This estimation consisted of first scoring the result quality of the IA as high, medium, or low (*i.e.* no emergent changes expected, some emergent changes expected, a significant number of emergent changes expected, respectively) and, secondly, giving the information, resource, and time availability used for the IA techniques implemented a rating of high, medium, or low (*e.g.* no missing information, some missing information, or little information available, respectively). The information, resource, and time parameters were chosen since they were the primary elements of the IA task influences cited by designers during the exploratory study (Section 4.4). The interviewees were asked to give a rationale for each of the ratings, and, in turn, the IA technique influences invariably also were mentioned, even though the interviewer did not explicitly question for them. The empirical studies focused on quantifying the task influences, as opposed to the technique influences, since asking designers to identify and rate information, resources, and time were deemed straightforward and easy to understand. Unlike simply coding the interview transcripts to develop the IA characterisations (Chapter 4), this rating exercise provides for estimating the typical IA techniques and task influences occurring in practice over a range of change types.

The interviewees were then prompted to provide ratings for several hypothetical scenarios. The information, resources, and time parameters were each individually changed from their original scores to high ratings (if they were not already scored as high), inducing additional ratings of the IA result quality by the designers. Finally, all of the parameters were then set to high, and the interviewees provided a rating for the expected IA result quality and their rationale in this optimal scenario. These hypothetical scenarios also gave designers a chance to further respond regarding technique and task influences and, in some cases, propose mitigation strategies. Figure E in Appendix B displays this rating data for the changes discussed.

In total, 23 changes were rated for IA quality during the IA empirical study. Not all of the initial 42 changes were covered because this number of ratings would take a considerable amount of time for each interview. Changes were selected that included the broadest range of IA techniques reportedly implemented by the interviewee. Occasionally, interviewees suggested replacement changes to cover different IA techniques or contexts. The information, resources, and time parameters were rated, and then the change type attributes and the IA performed of these substitute changes were elicited. These additional changes were not included in the original set of 42 changes since they were not always discussed as thoroughly as this initial set.

5.2.2 KEY OBSERVATIONS OF IMPACT ANALYSIS PRACTICE

Using the data collected, insights were made into IA practice at the aerospace company. The four categories of information collected, including (1) the IA techniques provided for by the company, (2) the IA techniques applied, (3) the change attribute ratings, and (4) the IA quality and information-resource-time availability ratings, were compared against each other and trends within the data were found to make observations. Four permutations of these categories led to four key observations:

1. *The IA techniques provided for by the company vs. the IA techniques applied (Section 5.2.2.1)*
2. *The IA techniques applied vs. the change attribute ratings (Section 5.2.2.2)*
3. *The IA techniques applied vs. the IA quality ratings (Section 5.2.2.3)*
4. *The change attribute ratings vs. the IA quality ratings (Section 5.2.2.4)*

The other potential combinations do not yield helpful insights. More specifically, the IA techniques prescribed can only be usefully compared to the IA applied in practice. In turn, the four combinations listed above cover the range of possible data permutations.

5.2.2.1 OBSERVATION 1: SEVERAL IA TECHNIQUES ARE NOT IMPLEMENTED IN PRACTICE

As suggested previously (Section 3.4.2.2 and Section 3.4.4), designers reported not to implement some of the IA techniques available within the aerospace company. The IA influences characterisation (Section 4.4) also highlights this lack of use of some IA techniques. However, analysing the IA techniques implemented as elicited from the full range of change types, as opposed to focusing on specific interviewees describing or rationalising the lack of use, distinguishes the number of designers potentially not implementing or rarely applying such IA techniques. Based on the elicitation exercise data, the engineers frequently did not apply IA through the integrated software requirement model, requirement traceability tool, or data dictionaries (Table 3.1). Even though each designer was questioned if he used these IA techniques during the aerospace IA study (Section 3.1.4), no designer had used the integrated software requirement model; only one systems engineer said that he had used the requirement traceability tool for IA; and, only one other systems engineer stated that he used the system data dictionary. As such, this data indicates that these IA techniques are widely not implemented in practice. Designers may not use these techniques because they do not know they exist or because they favour other techniques. Some designers also noted that these tools required dedicated administrators. Consequently, defining methods for making these techniques part of the change process, providing resources to administer these forms of IA, and educating designers on these techniques and their implementation processes can support the application of these techniques, contributing to IA rigour and result quality improvement.

Furthermore, the changes elicited show that systems engineers did not consult the software specifications, UML models, or code, developed by the software engineers, on changes to the functionality of the system design. The partitioning of the design work between system and software designers influences this application of IA, as highlighted in the next section.

5.2.2.2 OBSERVATION 2: SYSTEM AND SOFTWARE DESIGNERS RELY ON DIFFERENT IA TECHNIQUES

By categorising the IA elicited according to the changes attributes, patterns of the IA typically applied by systems and software engineers can be identified, as shown in Figure 5.7.

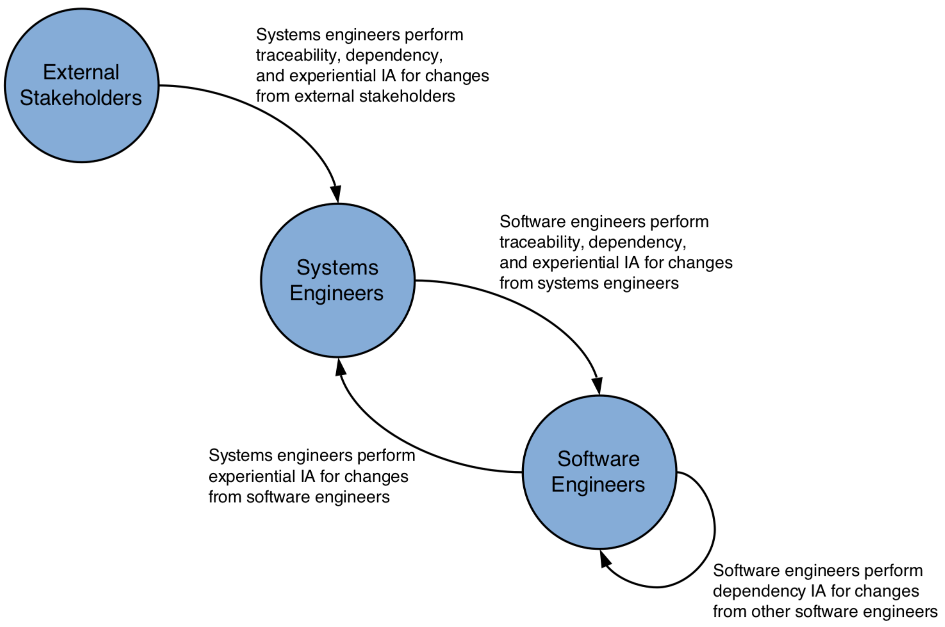

External Stakeholders
Systems engineers perform traceability, dependency, and experiential IA for changes from external stakeholders
Systems Engineers
Software engineers perform traceability, dependency, and experiential IA for changes from systems engineers
Systems engineers perform experiential IA for changes from software engineers
Software Engineers
Software engineers perform dependency IA for changes from other software engineers

Figure 5.7: IA performed by systems and software engineers

Firstly, the changes discussed with systems engineers were split into two groups according to their source, direction, and level attribute rating (*i.e.* system design changes requested by external stakeholders or software designers), covering 2 of the 4 possible combinations for these attributes (Section 5.2.1). These modifications were then grouped according to their formality and timing ratings. One group consisted of changes with low-low or low-medium rating combinations for formality and timing, and the remaining cases were placed into another group with higher formality and synchronicity. The medium-medium ratings were placed into the higher category since these changes are often well managed, even though they may not be perfectly documented or synchronised. Figure 5.8 illustrates the general results of this grouping, and Figure C in Appendix B displays the detail results for the changes elicited.

Changes from External Stakeholders	Changes from Software Designers
Traceability Dependency Experiential	Experiential

More Formal and More Synchronous Changes	Less Formal and Less Synchronous Changes
Traceability Dependency Experiential	Experiential

Figure 5.8: IA practised by systems engineers according to attribute ratings

By cross-referencing the changes across this grouping structure, observations can be made on IA practice. For instance, systems engineers primarily only used experiential IA for informal, asynchronous design modifications originating from software designers. In contrast, systems engineers performed traceability IA by manually tracing requirement documentation, dependency IA through software requirement models, and all forms of experiential IA for formal, synchronous changes requested by external stakeholders (Table 3.1). For the other two combinations, the changes elicited do not clearly indicate what IA technique is preferred. However, experiential IA may be relied upon when traceability and dependency IA cannot be performed. Figure 5.9 displays these patterns of the IA techniques typically applied by systems engineers.

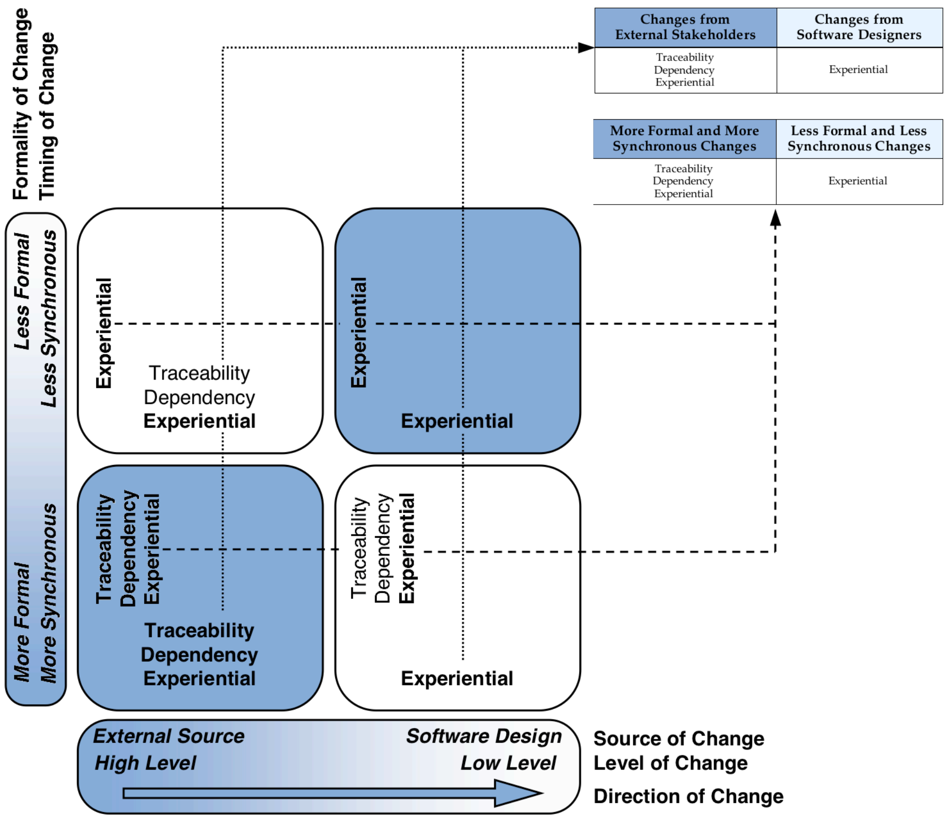


Figure 5.9: IA techniques practised by systems engineers (from Figure 5.8)

The changes elicited from software engineers were similarly analysed. The changes were first grouped according to the two other possible sources (*i.e.* software design changes requested by systems engineers or other software designers). The same criteria for dividing the changes by formality and timing attributes were used. Figure 5.10 depicts the general results from this division, and Figure D in Appendix B shows the detail results from the changes elicited.

Changes from System Designers	Changes from Software Designers
Traceability Dependency Experiential	Dependency

More Formal and More Synchronous Changes	Less Formal and Less Synchronous Changes
Dependency	Experiential

Figure 5.10: IA practised by software engineers according to attribute ratings

Although software designers implemented traceability, dependency, and experiential IA to some extent within each of the attribute groupings, Figure 5.10 depicts the dominant forms of IA used. For instance, for changes initiated by software designers that were more formal and synchronous, designers performed dependency IA using the software requirement and UML models as well as software code in all but two changes; traceability and experiential IA were used less frequently. In general, the changes elicited indicate that dependency IA may be relied upon more than the other forms of IA. Figure 5.11 shows the patterns for how software engineers applied IA.

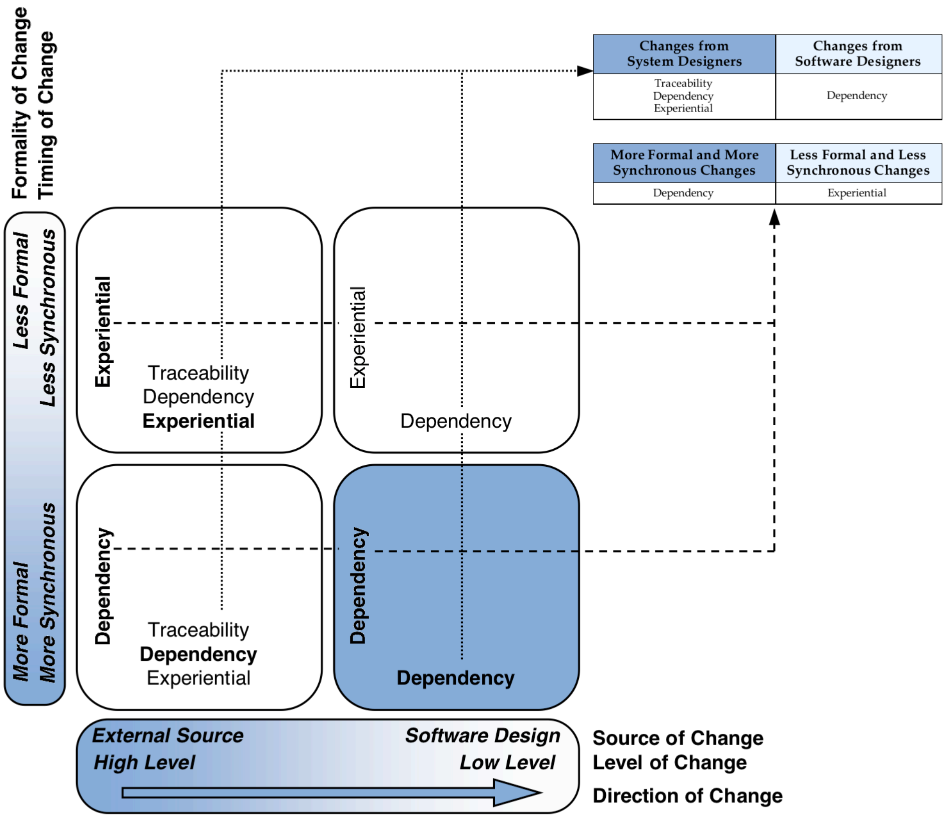


Figure 5.11: IA techniques practised by software engineers (from Figure 5.10)

Software designers most frequently performed dependency IA for formal, synchronous changes requested either by system or software designers. Even though traceability and experiential IA were applied for changes initiated by system designers, dependency IA may be applied more regularly since this technique provides the detail information required for software design and code changes. Interestingly, few changes could be elicited that occurred informally or asynchronously within software design. The changes that were elicited in this category included variable value changes (C-19), modifications to the real-time scheduling for all areas of the software design (C-28), and changes to an area of software code that is dependent on most other areas of the software design (C-42). The later two design areas notoriously change frequently and require significant stakeholder input. In turn, experiential IA may be relied upon to perform informal, asynchronous modifications initiated by system designers. However, this is only substantiated by two changes. More significantly, given that few changes under these conditions could be found, few also may exist in light of the aim to decouple the software design areas.

Notably, software engineers performed traceability IA through system requirements and software specifications as well as implemented dependency IA through software requirement and UML models (Table 3.1). As such, the IA elicited suggests that software engineers tend to perform a wider variety of the IA techniques available than system designers (Section 5.2.2.1). However, software designers also focus on the detail design and tend to rely on dependency IA (Figure 5.11), and the IA practised by system designers may cover more of the high-level design dependencies through traceability, dependency, and experiential IA (Figure 5.9). Given the combination of the work between a team of system and software designers working within a functional area, the culmination of the IA techniques can be very rigorous in that the IA performed can effectively search through the high-level and low-level dependencies of the software design (Section 4.3). Nevertheless, the sharing of IA results between these engineers must be implemented in a timely manner in order for this rigour to be achieved (*i.e.* addressing the partitioning and synchronisation influences) since knock-on effects and upcoming changes must be accounted for in the subsequent IA applied.

5.2.2.3 OBSERVATION 3: IA TECHNIQUE AND TASK INFLUENCES MAY EQUALLY AFFECT IA RESULTS

Both IA technique and task influences can theoretically affect IA result quality (Section 4.4). The information, resource, and time availability ratings support this characterisation and further indicate that these two general types of IA influences may equally affect the IA results obtained within the aerospace company. All of the IA influences were mentioned in this rating exercise as affecting the IA results produced, except for the process conflicts technique influence and the analysis education task influence. Moreover, most of the changes discussed did not produce perfect-quality IA results (*i.e.* not in the hypothetical scenarios elicited) and further iterations were expected, suggesting that addressing these influences could potentially lead to less rework.

Out of the 23 changes rated, designers designated that techniques influences primarily affected the IA results of 10 changes (Figure D in Appendix B). In most of these instances, high ratings for information, resources, and time produced mediocre IA results and were frequently affected by partitioning and synchronisation technique influences. However, in a few cases, high quality results were still obtained, and the designers discussed how they overcame or mitigated these IA influences. The remaining 13 changes discussed were primarily affected by task influences, including insufficient information, resources, or time available to perform IA. Designers most frequently cited task influences dealing with information as limiting factors. Given the concurrent nature of the design process at the aerospace company (Section 3.4.3), the change ratings fit this expectation of information availability. Even though a relatively small number of changes were rated, the outcome of this rating exercise suggests that both IA technique and task influences can equally play crucial roles in IA result quality.

5.2.2.4 OBSERVATION 4: LESS THAN IDEAL IA RESULTS TEND TO OCCUR IN CHANGES REQUESTED ACROSS INTERFACES

As illustrated by Figure D in Appendix B, the changes discussed during the rating exercise for IA quality primarily occurred across interfaces (*i.e.* the source and level attribute ratings differ). Given that most of these changes produced less than ideal results based on the IA influences (*i.e.* not in the hypothetical scenarios elicited), changes requested by external stakeholders tended to cause system designers difficulties in performing IA, and modifications between system and software designers tended to challenge these engineers. As indicated in Observation 3 (Section 5.2.2.3), these changes were often linked to the partitioning and synchronisation technique influences and task influences dealing with information. Only one change (C-33) across an interface did not include these specific technique and task influences. Consequently, improving the practices of obtaining, sharing, and updating information across these interfaces can address these primary technique and task influences to support effective IA (Section 4.4). Although other changes were occasionally discussed that could be implemented smoothly (not shown in Appendix B), the data collected indicates that the changes initiated across interfaces are prone to IA difficulties. This change data does not suggest a correlation between the formality and timing attributes and the quality of the IA results.

5.3 IMPACT ANALYSIS IMPROVEMENT STRATEGIES

As described in Section 3.1.4, interviewees in the aerospace IA empirical study were asked to describe strategies to improve IA results. Improvement strategies were also generally discussed with designers at the telecommunications company. Some of the designers could not propose specific means to make such improvements on-spot during the interviews, and others felt as if the IA techniques available were sufficient. However, designers suggested IA improvement strategies, which fall into three categories:

1. *Including more IA within change processes (Section 5.3.1),*
2. *Improving the input quality to IA (i.e. information, resource, and time) (Section 5.3.2), and*
3. *Advancing the IA techniques performed (Section 5.3.3).*

The following discussion analyses these improvement strategies through the IA characterisations (Chapter 4) and the four observations of IA practice within the aerospace company (Section 5.2.2). These IA improvements are also contextualised within the more general process improvement strategies of the aerospace and telecommunications companies (Section 3.4.5 and Section 3.5.5) and through the costs and benefits of implementation. This analysis leads to the proposal of practical IA improvement strategies (Section 5.4).

5.3.1 INCLUDING MORE IMPACT ANALYSIS WITHIN CHANGE PROCESSES

When queried on how to improve the quality of IA task results, interviewees (I-2, I-4, I-13, I-15, I-16, I-17, I-19, I-21, I-26) recommended performing "more" IA. These suggestions occurred in the context of performing traceability or dependency IA more frequently or in addition to relying on other IA techniques. Other strategies elicited include improving the design review process, entailing more frequent and less formal reviews, focusing on finding potential design faults, or having more stakeholders or experts review the design earlier in the development process.

These suggestions essentially imply the application of more rigorous IA techniques. Using traceability or dependency IA techniques above and beyond current IA practice and mandating more design reviews with participation from a broader range of stakeholders increases the search thoroughness for potential change impacts, and, thus, IA rigour. As suggested by the first observation (Section 5.2.2.1), including more IA can demand defining processes to implement additional IA techniques, providing resources to administer the associated tools, and educating designers on these processes.

In addition, the observations of IA practice at the aerospace company indicate that addressing other IA influences may be necessary. In particular, the second observation (Section 5.2.2.2) suggests that system and software designers must effectively share information from traceability and dependency IA to obtain rigorous IA results, while the fourth observation (Section 5.2.2.4) points out that system and software designers may face challenges in transmitting such information used for IA across this interface in a timely manner. As such, the methods for sharing IA results in time (*i.e.* addressing the partitioning and synchronisation influences), allowing for appropriate changes to be made synchronously, may require improvement in order to realise the benefits of including more IA.

Within the aerospace company's process improvement strategy to provide for requirement reuse (Section 3.4.5), a standardised, reusable requirement traceability matrix and integrated requirement model are being developed. Upon the launch of this strategy, the product development process prescribed in terms of work products, schedules, and resources will be modified to focus on the customisation of the platform for the creation of another product. In turn, the change processes and the application IA techniques can also change. This shift can create the framework for changing the typical IA performed by designers to include more IA with defined implementation processes. Consequently, this IA improvement strategy elicited may be poised for implementation within the aerospace company. In contrast, specifically implementing more IA within the telecommunications company across all projects may be more difficult since design processes are not intended to be controlled or standardised across projects, and the current company strategy for design process improvement does not entail addressing the application of IA techniques (Section 3.5.5).

In either company, implementing this IA improvement strategy inevitably can incur costs for defining and administering the processes to use the IA techniques available and training the engineers in these methods. However, more rigorous IA can help to reduce the risk of emergent changes late in the design process. Since identifying and implementing design modifications early is less costly than finding and making changes due to design errors late, this IA improvement strategy can potentially reduce development costs. Nevertheless, while more rigorous IA can make the search process for potential change impacts more systematic and reduce such costs, this improvement strategy alone may not entirely reduce the risk of emergent changes. All other IA influences within a particular change process may also need to be addressed in conjunction with including more IA to benefit from this potential increase in IA rigour.

5.3.2 IMPROVING THE INPUT QUALITY TO IMPACT ANALYSIS

Alternatively, designers interviewed in the IA empirical studies proposed that the quality of information used for IA primarily affects the quality of results. They specifically suggested that timely communication between stakeholders could improve the information available for IA (I-14, I-29, I-30, I-42, I-43). For example, if systems and software engineers work asynchronously, details of changes can be forgotten if not documented, and the IA performed later may not be of high quality. Alternatively, IA performed and documented at the time of a change request can become out-of-date as the design progresses, leading to unaccounted knock-on effects if the change request is later implemented without additional IA. Implementing just-in-time IA using current design information can capture such unanticipated consequences. Furthermore, the communication of IA results and upcoming changes to other relevant stakeholders can eliminate knock-on effects for other IA tasks performed. In addition, designers also suggested that allowing more time and resources for IA tasks can improve IA results (I-13, I-17, I-19). For instance, rushing designs for a software release in order to meet planning milestones can lead to a known degradation of design quality. IA may not be applied thoroughly and software errors may not be fixed due to the lack of time and resources in this case.

This IA improvement strategy primarily addresses the IA task influences and also suggests confronting process conflicts in terms of resource allocation. However, other technique influences can still arise in implementations of systems engineering or agile software development design processes, causing emergent modification to occur. As such, the third observation of IA practice within the aerospace company (Section 5.2.2.3) indicates that this strategy may not completely improve the IA results in practice, given that approximately half of changes can primarily be affected by IA technique influences. Nevertheless, this improvement only through task influences may be significant. The telecommunications company may be inclined solely focus on such a tactical strategy of improving IA through agile development methods. For instance, direct and regular communication between stakeholders, as supported by agile processes, can improve the quality of information and resources available for IA. Such a focus on IA improvement may only incur costs related to the time put in by stakeholders. Alternatively, the aerospace company could use such a strategy in conjunction with other process improvement methods given its outlook on process improvement through requirement reuse.

5.3.3 ADVANCING THE IMPACT ANALYSIS TECHNIQUES PERFORMED

Finally, some interviewees suggested advancing specific techniques and tools used for traceability and dependency IA (I-15, I-16, I-21, I-22). These improvements primarily included building larger and more detailed databases or models of the system and software designs. More data on traceability relationships could be captured (*e.g.* design rationale[31]), or more information could be included in the models of the high-level system functionality. Alternatively, different modelling styles were proposed.

These proposals essentially focus on increasing IA rigour and eliminating some of the task and technique influences. Larger and more detailed databases and models can specifically address partitioning. For instance, an all-encompassing database or model could describe an entire product at a high and low-level of granularity for both systems and software engineers. Improving the ability of tools to capture and share information can also take into account the information perceived to be unavailable, ambiguous, or not required and manage the magnitude of information (*i.e.* addressing the synchronisation technique and information-related task influences). However, such advanced IA techniques can still have limitations due to other IA influences, and these influences also must be addressed.

In addition, the cost of developing supporting software as well as any other data population and infrastructure to implement such innovative IA can be costly, and the benefits should be weighed against the existing IA techniques. Addressing the fourth research question (Figure 1.10) suggests the relative benefit of improving IA through such methods. Nonetheless, this improvement strategy can primarily help to address the difficulties of applying IA across interfaces, as depicted by the fourth observation in the aerospace company (Section 5.2.2.4). By developing advanced traceability and dependency IA techniques, communication between external stakeholders, systems engineers, and software designers may improve by means of common information and models. Given that most of the changes across such

[31] Capturing design rationale could also be used in and to improve experiential IA, as discussed in Section 2.5.3.

interfaces elicited encountered imperfect IA, the benefits of advanced IA techniques can be significant.

Since such advanced IA techniques can require significant costs due populating and maintaining large databases and models, the telecommunications company may be less inclined to implement such a strategy through agile development (Section 2.5.1). In turn, the aerospace company could incorporate such IA improvement strategies as a long-term vision alongside its goal of developing processes and tools for requirement reuse.

5.4 EXTRACTION OF PRACTICAL IMPACT ANALYSIS IMPROVEMENT STRATEGIES

The discussion of the observations at the aerospace study indicates specific technique and task influences to focus on for IA improvement (Section 5.2.2). Although all IA influences ideally should be addressed, the IA improvement strategies elicited imply that addressing all influences may not be simple and require multiple tactics. As such, practically improving IA can constitute focusing on the technique and task influences identified as problematic in practice. Given that the strategies elicited address the influences pinpointed in the aerospace IA study, they may be sufficient from this perspective. However, other strategies handling these same influences can also allow for IA improvement. In turn, based on the observations and elicited strategies from the aerospace IA study, practical improvement of IA can occur through three primary means:

- Delineating the use of existing or new IA techniques, providing resources to administer these techniques, and educating designers on these techniques can increase IA rigour and quality, addressing the method definition, administration, and education influences.
- Defining processes for sharing and updating information used in IA techniques and tools can improve IA rigour and quality, handling the partitioning and synchronisation technique influences as well as the information-related task influences.
- Providing sufficient resources and time for IA can improve IA quality and address these task-related influences. More importantly, IA must be viewed as a worthy and crucial part of change processes to confront the process conflicts caused by resource allocation, in turn, allowing for increases in IA rigour.

Strategies that specifically address these elements may more simply increase IA quality than other potential strategies within the aerospace company, which uniformly attempt to mitigate all of the IA influences through many different tactics. By targeting these primary barriers to obtaining high-quality IA results, IA techniques can be applied more rigorously and frequently. The fourth research question (Figure 1.10) examines the effectiveness of implementing these strategies as opposed to other process improvement initiatives, and the refinement of this question specifies the means of investigation (Section 5.6).

5.5 REVISITING THE RESEARCH QUESTIONS

The empirical studies provide insights into the first three research questions. For the first research question (*How are change processes and impact analysis prescribed at the system and software engineering interface within industry?*), the empirical studies indicate that both formal and informal change processes can occur within systems engineering and agile development processes. In both of these contexts, the companies do not specify or prescribe the application of IA techniques, but make a range of techniques available. No literature suggests this lack of process definition affecting IA implementation (Section 3.6). Consequently, the elicited IA improvement strategies to define methods for implementing IA and processes for sharing information during change processes to use in IA techniques and tools fills a gap in literature and industry practice (Section 5.4).

The second research question (*Does impact analysis influence the management of emergent changes in practice? If so, how?*) is addressed through the definitions of IA techniques and tasks and the IA quality and rigour characterisations (Section 4.6). These IA definitions and characterisations distinguish between prescribed and practised IA and delineate how, depending on the IA techniques applied and the IA influences active, the search thoroughness for potential change impacts and inputs available for IA can vary, thereby, affecting the risk of unidentified, emergent changes. Although the empirical studies indicate that IA is recognised as a means to manage design changes (Section 3.4.4), the observations of practice at the aerospace company suggest that commitment to performing IA may not be consistent based on the observed the lack of resources and time available for IA. As such, addressing the resource-related process conflicts and viewing IA as a crucial part of change processes can improve IA quality and the emergence of unexpected modifications (Section 5.3.2).

The classification of the IA influences contributes to the third research question (*What are the challenges in using impact analysis to manage emergent changes at the system and software engineering interface?*). As particularly observed in changes analysed (Section 5.2.2.3), IA technique and task influences can equally affect the IA rigour and result quality. However, within these general categories, particular influences can dominate. Given that IA improvement strategies cannot simply eliminate all of these influences, pinpointing certain influences to address can lead to practical IA improvement (Section 5.4).

Although IA improvement was found to have support within the companies participating in the empirical studies, the relative consequences of this improvement on the design process compared to other improvement strategies cannot be intuitively estimated based on the two empirical studies conducted. The fourth research question (Figure 5.12) confronts this need to determine the context of IA improvement and is refined through additional research questions in the following section.

5.6 RESEARCH QUESTION REFINEMENT

In order to address the fourth research question (Figure 1.10 and restated in Figure 5.12), two approaches could be taken. Firstly, additional empirical studies could be conducted on the relative consequences of IA and other design process improvement strategies (*e.g.* improving design task scheduling or requirement management). Given the timescale of this research project, at most, only a few additional empirical studies could be conducted. Based on these few, potential data points, the effects of IA improvement may be difficult to generalise and cannot provide complete insight into this research question. Secondly, a theoretical analysis of IA improvement could be conducted. Modelling and simulation provides a means to perform such an investigation and meets the intent of the fourth research question to explore a variety of scenarios for process improvement[32]. Through this approach, IA improvement strategies could be investigated in more detail, and even the consequences of IA on specific design processes could be predicted. However, the simulation implemented only demonstrates or indicates the potential effects of IA improvement in that some of the parameter values are not calibrated to specific empirical data. Nevertheless, given the potential thoroughness of such analysis, this research project focuses on examining IA improvement through modelling and simulation. Figure 5.13 displays the refinement of the fourth research question to focus on this aim.

Can the application of impact analysis improve the design process? If so, how?

Figure 5.12: Research question 4a (from Figure 1.10)

[32] Kellner *et al.* (1999) detail the benefits of modelling and simulating as a means to investigate process improvement, strategic management, planning, and control and operation management.

Can modelling and simulation demonstrate the effects of impact analysis improvement on the design process? If so, how?

Figure 5.13: Research question 4b, refinement of research question 4a

As discussed in Section 3.4.3, change packaging is associated with IA improvement and the necessary rework within design processes. Given the interest in pursuing further investigation of change packaging by the aerospace industrial partner, the modelling and simulation work also investigates this element within the context of IA improvement. Figure 5.14 depicts this extension of the fourth research question.

Does change packaging affect impact analysis improvement? If so, how?

Figure 5.14: Research question 4c, extension of research question 4a

The remainder of this dissertation focuses on modelling and simulating IA improvement and change packaging within the aerospace company to address the fourth research question (Figure 5.12) and its extension (Figure 5.14). This investigation provides a means to specifically evaluate and compare the effectiveness of the practical IA improvement strategies elicited (Section 5.4) and change packaging policies (Section 7.5) for this firm. Based on this analysis, high-level heuristics for when these strategies and other process improvement strategies should be employed are derived (Section 8.2).

5.7 SUMMARY

This chapter outlines an elicitation method for IA practice in terms of the attributes of changes and examines the results of applying this method within the IA empirical study at the aerospace company. Interviewees were queried for specific instances of changes actually occurring within a design project and quantitative ratings of the IA inputs and outputs. This data collected is analysed, producing four key observations into IA practice, which are correlated with the IA characterisations (Chapter 4). Specific IA improvement strategies elicited from aerospace designers are then discussed in terms of the IA characterisations, these key observations, the process improvement strategies from the aerospace and telecommunications companies, and their potential costs and benefits. By bringing together these insights, several practical means for IA improvement are identified. Finally, this chapter concludes by revisiting the research questions through the empirical studies and refining the fourth research question on the effects of IA improvement strategies on the design process (Figure 5.12). The remainder of this dissertation focuses on addressing this research question through modelling and simulation (Figure 5.13) and extending the investigation to determine the effects of change packaging (Figure 5.14).

6 :: INVESTIGATING THE IMPLICATIONS OF IMPACT ANALYSIS IMPROVEMENT

In order to understand the benefits of implementing IA improvement strategies and, in turn, address the fourth research question (Figure 5.12), the subsequent modelling and simulation work aims to compare IA improvement with other possible process improvement initiatives. As suggested by interviewees (I-2, I-16, I-23) in the empirical studies, IA improvement strategies may be viewed as secondary to process improvement to increase work productivity, including optimising design task scheduling or developing better requirement management practices. Investigating the potential consequences of IA improvement through modelling and simulation allows managers to systematically analyse the trade-offs in implementing strategies to improve the design process and demonstrates the significant effects IA improvement can have on product development.

Literature provides insights into the modelling and simulation of design processes, and research in *system dynamics* and *software process dynamics* specifically describes previous modelling efforts and simulation results, suggesting methods for executing the analysis of IA improvement strategies. In turn, the adaptation of *the rework cycle* model developed by Cooper (1993a; 1993b; 1993c) is shown to meet the requirement to investigate IA improvement against other process improvement strategies (Section 6.1). Using this model, a system dynamics method for modelling and simulation (Section 6.2) is implemented, as described in the remainder of Chapter 6 as well as Chapter 7. Given that the adapted rework cycle model describes design processes at a high-level, Chapter 8 derives high-level heuristics for implementing process improvement strategies using the simulation results.

6.1 LITERATURE ON ANALYSING DESIGN PROCESS IMPROVEMENT

As indicated by Browning *et al.* (2006) in an extensive review of process modelling practices across a variety of disciplines, *system dynamics* (Section 6.1.1) provides a means to understand the behaviour of product development processes at a high-level and analyse policy changes, unlike other modelling methods. Task details and interdependencies within design processes are not typically captured in system dynamics models, and, in some cases, entire design processes may only be represented by a few key elements. However, in order to investigate the overall effects of IA improvement strategies on design processes (Figure 5.12), the granularity of a detailed, task-based design process model is not required if the key factors are still included (as discussed in Section 6.1.1). As such, system dynamics literature provides an appropriate basis for the modelling and simulation of IA in this research project, and searches for relevant references to the dynamics of change processes were performed in textbooks and the System Dynamics Review journal. Pertinent citations within these primary sources were also reviewed, supplementing the literature search.

Other disciplines, not mentioned by Browning *et al.*, are related to system dynamics, including *software process dynamics* and *operations research*. The field of software process dynamics (Section 6.1.2) stems from system dynamics and focuses specifically on software development processes. As such, this research project also logically draws from literature in this field. In turn, operations research (OR)[33] focuses on analysing the coordination of activities, including business and design processes (Hillier and Lieberman 1995: 3). This field often concentrates on applying mathematics to analyse management decisions and, as argued by Ackoff (2001), takes a narrower perspective to analyse process behaviour than system dynamics in that emphasis is placed on the type of systems investigated and the associated mathematics used. Although the mathematics behind models is relevant, system dynamics does not focus on these details. In turn, the construction of the model used in this research project to depict IA within a design process draws from system dynamics and software process dynamics literature. However, OR literature was also reviewed through references in these two disciplines for relevant mathematics to model design processes.

6.1.1 SYSTEM DYNAMICS

The fundamental concepts of system dynamics originate from Jay Forrester's work at MIT in the 1960s. Forrester applied feedback control theory to his research in *industrial dynamics*, later termed *system dynamics*, in order to understand the management of industrial processes and then social systems, such as urban population growth (Ford 1999; Lane 1994). With this foundation, the System

[33] OR is also referred to as *management science* in some texts (Hillier and Lieberman 1995), and the field of *industrial engineering*, also known as *operations management*, promotes the application of operations research in practice (Bailey and Barley 2005).

Dynamics Society (2007), the principle organisation for the system dynamics community, defines "system dynamics" as:

> *a methodology for studying and managing complex feedback systems, such as one finds in business and other social systems. In fact it has been used to address practically every sort of feedback system.*

As such, product design and change processes feature in the systems investigated within this literature discipline. The System Dynamics Society (2007) also suggests the examination of system behaviour through iterating between six analysis phases; this methodology:

- *Identifies the problem,*
- *Develops a dynamic hypothesis explaining the cause of the problem,*
- *Builds a computer simulation model of the system at the root of the problem,*
- *Tests the model to be certain that it reproduces the behaviour seen in the real world,*
- *Devises and tests in the model alternative policies that alleviate the problem, and*
- *Implements this solution.*

After defining the scope and elements of the problem to be modelled, literature in system dynamics typically represents the relationships between model parameters through two types of diagrams, *causal loop* and *stock and flow* diagrams (*i.e.* depicting the "dynamic hypothesis" stated in the second bullet point above). Appendix C discusses the notation of these diagrams and their interpretation. Accompanying these diagrams is an associated set of time-dependent equations and even numerical data look-up tables from empirical information, forming the system model. Such models can be simulated with given initial conditions through taking time steps (Sterman 2000). In turn, perturbing the model parameters can indicate the effect of management policy changes. Sterman (2000: 86) and Ford (1999: 171-179) both propose similar methodologies, examining each phase in detail, and Section 6.2 discusses the application of these investigation techniques for this research project.

System dynamics literature reveals a key model that highlights the dynamics of change processes. Cooper (1993a; 1993b; 1993c) developed *the rework cycle* model to describe the impact of additional, unknown design effort required on product development processes. Figure 6.1 depicts the rework cycle through a stock and flow diagram (detailed in Appendix C) in which work products are produced, revised, and completed.

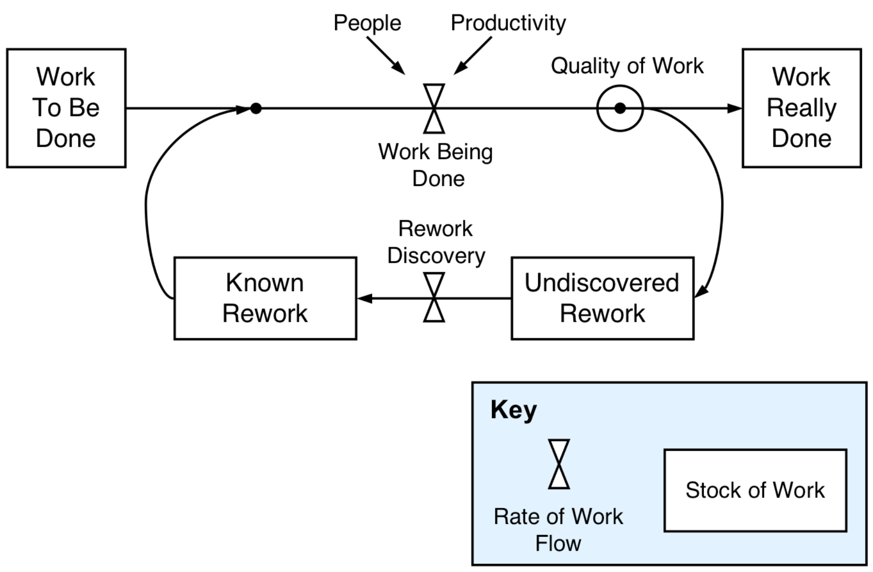


Figure 6.1: The rework cycle

As represented by Figure 6.1, at the start of a project or design process, all work required to complete the project resides in the stock of *work to be done*. As the design process begins, quantities of work are shifted from this stock or reservoir of work to do, and work products are developed at a rate based on the *work being done* flow, which is affected by the *people* available and *productivity* of this staff. Such work is executed at less than perfect quality in that the work products produced contain unknown errors. This quality, which is represented by a fractional value, determines the portion of work that enters the stock of *work really done* and the stock of work requiring rework. Thus, the *quality of work* variable does not affect the rate of work done, but rather distinguishes that some work products implemented have no errors and others require rework. In turn, a circle, instead of a valve marker (Appendix C), denotes the quality of work in Figure 6.1.

Quantities of rework (*i.e.* initially unknown or unidentified errors contained in work products) are first held in the stock of *undiscovered rework*. The *rework discovery* flow indicates the rate of identification of such errors in work products and determines the shift of undiscovered rework into the stock of *known rework*. This known rework is then incorporated into the flow of work being done, and work products are revised to eliminate errors.

Arguably, the rework cycle lends itself to conceptualising change processes. Cooper (1993b) discusses undiscovered rework in terms of "errors", and such errors in work products require associated changes. Consequently, the feedback loop in the rework cycle generally depicts change processes in the context of a generic design process, represented by the feed-forward part of the loop. In addition, the rework discovery flow parameter allows for the conceptualisation of IA in that the scoping of changes is distinguished within the model[34].

Cooper and his colleagues also extend the rework cycle[35] to model additional influences on design processes and specifically include factors affecting the people, productivity, and quality of work variables (Lyneis *et al.* 2001). Figure 6.2 shows the effects[36] of key variables, including the *work quality to date, availability of prerequisites, out-of-sequence work, schedule pressure, morale, skill and experience, organisational size changes*, and *overtime,* on design processes through a causal loop diagram (detailed in Appendix C).

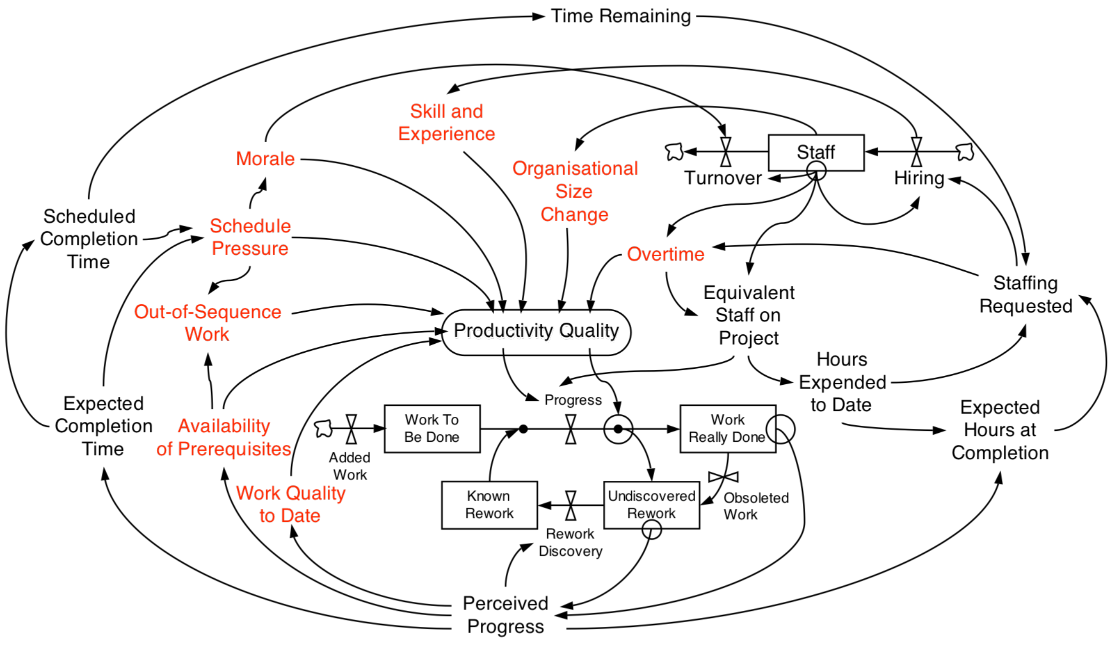


Figure 6.2: The extended rework cycle model (Lyneis *et al.* 2001)

[34] Section 6.3 further interprets IA within the rework cycle model.
[35] The obsolescence of work modelled in Figure 6.2 is further discussed with respect to Figure 6.14.
[36] Section 8.2.1 discusses the polarity of these relationships represented in the extended rework cycle model.

Lyneis *et al.* (2001) further indicate additional feedback loops exist beyond those depicted in Figure 6.2, including other parameters, such as:

- *Baseline design quality (clarity of initial specifications and pre-requisite design),*
- *Availability and quality of procured design and/or products,*
- *Availability of customer-furnished information and/or equipment,*
- *Adequacy of supervision,*
- *Space constraints,*
- *Management concern for quality, and*
- *Managerial continuity and experience.*

In turn, including such additional variables provides a broader context for the basic rework cycle model (Figure 6.1) and allows for the investigation of the effects of a range of design process improvement strategies. For instance, optimising the scheduling of design tasks, as focused on by interviewees in the empirical studies and cited in the introduction to this chapter, corresponds with the *out-of-sequence work* variable in the extended rework cycle model (Figure 6.2) in that effective planning can improve productivity and the quality of work. As such, this process improvement strategy can be examined by estimating the magnitude of the increases in productivity and the quality of work variables and determining the consequences on the basic rework cycle model (Figure 6.1). Given that the other variables included in the extended rework cycle model (Figure 6.2) can be used to similarly represent other process improvement strategies, these strategies also can be compared by establishing the magnitudes of their improvements to the people, productivity, and quality of work variables and analysing their relative effects on the basic rework cycle. Analysing perturbations in the basic rework cycle model variables can illuminate the consequences of a range of process improvement strategies.

In turn, the rework cycle provides a framework to compare IA improvement strategies with other design process improvement initiatives, meeting the requirement to address the fourth research question (Figure 5.12) in that:

- The rework cycle allows for the conceptualisation of IA through the rework discovery variable, and
- The relative magnitudes of the key variables in the basic rework cycle, including people, productivity, quality of work, and rework discovery, can estimate the consequences of implementing IA and other process improvement strategies.

As such, this research project adapts the rework cycle to include IA (Section 6.3) just as other models have been developed on the premise of the rework cycle (Ford and Sterman 1998). Two other key change process models found in system dynamics literature include additional elements to the rework cycle. Ford and Sterman (1998) focus on the quality of work depicted within the rework cycle model and model additional factors influencing this variable. Similarly, Park and Peña-Mora (2003) extend the rework cycle to model elements specific to construction project changes. In both of these instances, additional insights through simulation were gained by extending the rework cycle model (Ford and Sterman 2003a; Ford and Sterman 2003b; Taylor and Ford 2006). Instead of developing another, new model, adapting the rework cycle model to include IA can build on its successful application to many projects in a variety of industries, including software development (Cooper 1993b).

Furthermore, the simplicity of the rework cycle enhances the communication of the model analysis and simulation results to industry. Using a detailed model, such as those developed within software process dynamics research (Section 6.1.2), may not lead to results that can be easily understood in that the complexity of the model can obfuscate how the results arise and are justified. Although high-level models, such as the rework cycle, do not capture the task details of design and change processes, the results are to some extent measurable and can demonstrate the effects of process improvement strategies.

Moreover, other models specifically delineate and investigate software inspection tasks, a type of experiential IA (Section 2.5.3), within design processes. However, these models also confine such IA within a section of large design process models (Madachy 2008: 275-289; Raffo and Kellner 1999b; Thelin *et al.* 2004). In turn, the influence of other IA techniques is lost. As opposed to the adapted rework cycle model (Section 6.3), modelling a variety of individual IA techniques within a detailed design process is an alternative approach. However, this approach is impractical since IA techniques can often be applied by designers at will and do not necessarily occur according to defined processes in practice (Section 4.4). In turn, the high-level rework cycle model is fit-for-purpose and can contribute to advising the industry collaborators on the relative effects of IA and other process improvement strategies.

6.1.2 SOFTWARE PROCESS DYNAMICS

Abdel-Hamid pioneered the application of system dynamics to analyse software processes (Abdel-Hamid and Madnick 1991). His model of software development projects uses a continuous, time-based simulation with over 138 variables and 237 equations to investigate management policies to improve project performance (Ruiz *et al.* 2001). Further work has employed this model framework to suit specific purposes, including planning, operational management, process improvement, and technology adoption (Raffo and Kellner 1999a). As Abdel-Hamid developed his model based on system dynamics, Kellner began creating the first state-based simulation of software development. Kellner's work was followed by additional research into such models and simulation, notably by Raffo and Madachy (Kellner *et al.* 1999; Raffo and Kellner 1999a). This initial research into software process simulation has given rise to the field of *software process dynamics* (Madachy 2008). Software process dynamics draws from the methodology of system dynamics, but includes (1) state-based, (2) discrete, and (3) hybrid models of software processes (Madachy 2007; Wakeland *et al.* 2004).

- **State-based models** use events to trigger transitions in the model state. For instance, the beginning of software coding can spawn the development of test plans. This representation provides a means to easily model parallel events.
- **Discrete models** can depict sequences of activities across a time period. However, unlike continuous models in system dynamics, time is only updated in simulations when particular events occur. As such, discrete models are typically used to represent queues and delays associated with events.
- **Hybrid models** combine state-based, discrete, and continuous system dynamics models to denote relevant aspects of software development processes.

Discrete models fit to investigating change packaging (*i.e.* a prioritised queue) and, thus, meet the need to address research question 4c (Figure 5.14). In turn, the rework cycle model is adapted to model change packaging and implemented through a discrete simulation (Section 6.3). However, other model structures to represent change packaging were also investigated.

Although no research directly related to change packaging was found in the literature search, two discrete models to date, as noted by Pfahl *et al.* (2007), simulate software release packaging (*i.e.* the packaging of features for upcoming software versions). No other varietals of packaging have been found to date in literature. For the first model, Höst *et al.* (2001) analyse the incorporation of new requirements into a series of software releases. In their discrete model, requirements are prioritised and packaged for a software release, and the appropriate extensions are made to the software design accordingly. As such, this framework assumes that all requirements are completely known and that no rework is required. This requirement-packaging model fundamentally differs from the adapted rework cycle (Section 6.3), which assumes not all rework is known and depicts the discovery of unanticipated errors and the associated emergent changes. Furthermore, in the model by Höst *et al.*, new requirements are prioritised and implemented in order based on their arrival time (*i.e.* late requirements are incorporated into the design late). The adapted rework cycle model does not prioritise changes as such and packages early and late changes together (Section 7.5). Figure 6.3 depicts the structure of the model by Höst *et al.*, showing the rearrangement of requirement priority between releases. This model does not show rework, iteration, or IA tasks between the *elicit*, *select*, or *construct* development phases or between releases.

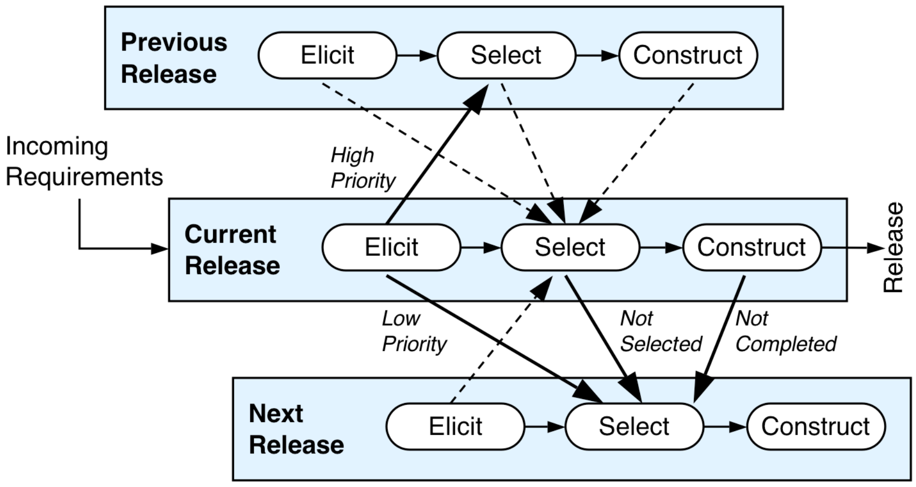


Figure 6.3: Model structure of requirement prioritisation (Höst *et al.* 2001)

Höst *et al.* note that this prioritisation model can be improved, but specify that such improvements are out of the scope of their research. They focus on applying *queuing theory*[37] from operations research (Hillier and Lieberman 1995: 661-714) to model the allocation of resources to implement new requirements. Given the purpose of modelling staffing allocation, this research project does not draw from this model structure and adapts the rework cycle to include change packaging instead (Section 6.3), allowing for the comparison of change packaging strategies with IA and other design process improvement strategies and addressing the refined fourth research question (Figure 5.14).

Pfahl *et al.* (2007) present a second model on release packaging, extending the concepts developed by Höst *et al.* to a more detailed level. They specifically

[37] Given that the simulation of the adapted rework cycle does not explicitly model resources, queuing theory is not used (Section 6.3).

incorporate modelling of resource allocation to specific design tasks (*e.g.* design, implementation, test). However, unlike Höst *et al.* and the adapted rework cycle (Section 7.5), this model requires actual stakeholders to input prioritisation levels in order to perform simulations (Saliu and Ruhe 2005). Although a variety of prioritisation scenarios can be evaluated through this method, the modelling framework does not allow for the representation of rework or IA improvement strategies.

6.2 MODELLING AND SIMULATION METHOD AND DISCUSSION STRUCTURE

As suggested in Section 6.1.1, system dynamics entails performing modelling and simulation through a defined method. The methodology proposed by Sterman (2000: 86) details the standard steps of such investigations, which also apply to software process dynamics research (Kellner *et al.* 1999). Figure 6.4 accordingly articulates these steps as applied to the modelling and simulation of the adapted rework cycle (Section 6.3):

Step 1 **(Section 6.3 and Section 6.4)**	**Problem Articulation and Model Formulation** • Delineate the problem and model boundaries/limitations • Define the key parameters • Map the parameter relationships (*e.g.* through a *stock and flow* diagram) • Evaluate model formulation • Determine the *reference modes* (*i.e.* the intended model behaviour over time)
Step 2 **(Section 7.1 and Section 7.2)**	**Simulation Development** • Specify the simulation structure • Estimate the parameter values and set initial conditions
Step 3 **(Section 7.3)**	**Simulation Testing** • Compare the simulation results against the *reference modes* • Determine the simulation behaviour under extreme conditions • Analyse the sensitivity of the simulation results to parameter estimations
Step 4 **(Section 7.4 and Section 7.5)**	**Policy Design** • Define policy designs to assess (*i.e.* IA quality improvement and change packaging) • Represent the policy designs through simulation scenarios • Analyse the simulation results in terms of the policy designs

Figure 6.4: Steps of the modelling and simulation method applied

Given that data from the aerospace company (Appendix D) is used to develop the reference modes (Section 6.4), parameter values, and initial conditions in the modelling and simulation performed, the simulation results obtained can be used to compare and evaluate the practical IA improvement (Section 5.4) and change packaging strategies derived (Section 7.5) for this firm. As suggested in Section 6.1.1, this examination entails perturbing the basic rework cycle parameter values (Figure 6.1) based on the improvement strategy investigated and comparing the simulation results. However, analysing these results in terms of the extended rework cycle (Figure 6.2) also leads to the suggestion of more general IA improvement and change packaging heuristics (Section 8.2), which highlight the trade-offs between different process improvement strategies.

Consequently, the discussion structure for the remainder this dissertation develops the adapted rework cycle model and defines its behaviour for the aerospace company's design process (Chapter 6), simulates and compares the aerospace company's IA and change packaging strategies using this model (Chapter 7), and derives heuristics for process improvement strategies based on these simulation results (Chapter 8). Figure 6.5 outlines these aims for the remaining chapters.

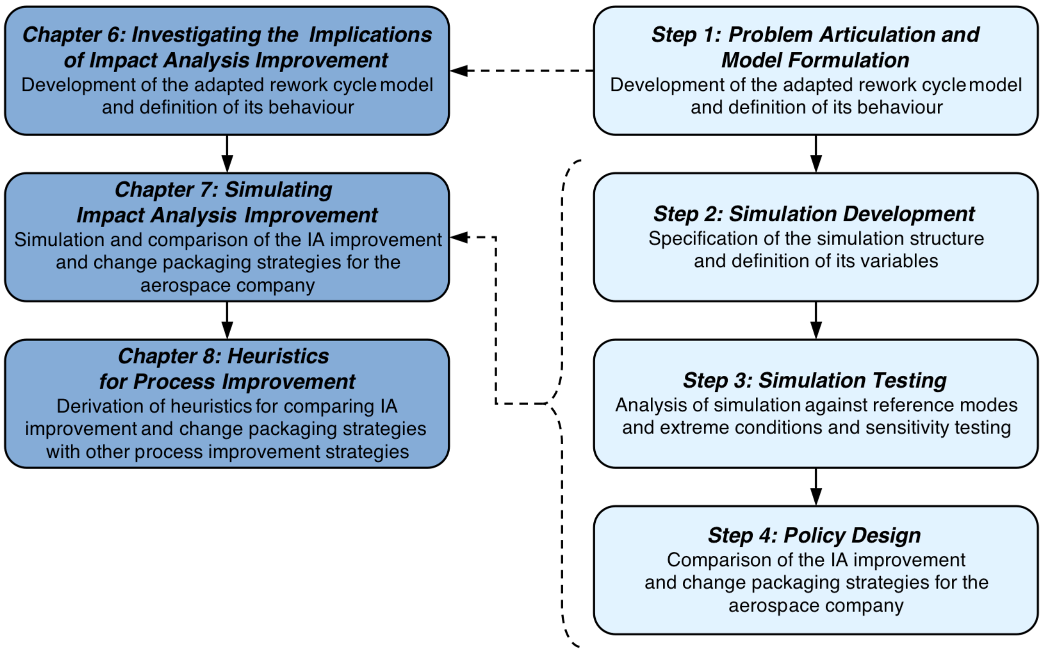

Chapter 6: Investigating the Implications of Impact Analysis Improvement
Development of the adapted rework cycle model and definition of its behaviour
Chapter 7: Simulating Impact Analysis Improvement
Simulation and comparison of the IA improvement and change packaging strategies for the aerospace company
Chapter 8: Heuristics for Process Improvement
Derivation of heuristics for comparing IA improvement and change packaging strategies with other process improvement strategies
Step 1: Problem Articulation and Model Formulation
Development of the adapted rework cycle model and definition of its behaviour
Step 2: Simulation Development
Specification of the simulation structure and definition of its variables
Step 3: Simulation Testing
Analysis of simulation against reference modes and extreme conditions and sensitivity testing
Step 4: Policy Design
Comparison of the IA improvement and change packaging strategies for the aerospace company

Figure 6.5: Modelling and simulation discussion structure

6.3 MODELLING THE DESIGN PROCESS VIA THE ADAPTED REWORK CYCLE

As discussed in Section 6.1, the rework cycle provides the basis to model IA improvement and change packaging, and Figure 6.6 illustrates the *adapted rework cycle*, highlighting these modifications made. The following sections discuss each of these adaptations in detail to define the model used for the simulation, as outlined in Step 1 of the modelling and simulation method applied (Figure 6.4).

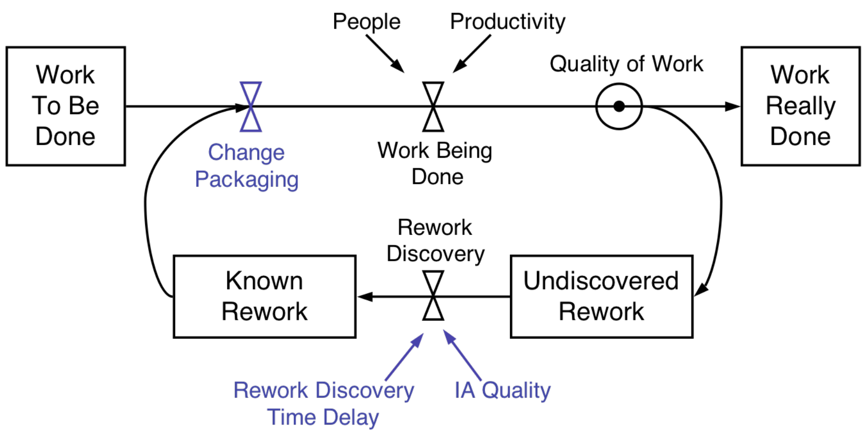


Figure 6.6: The adapted rework cycle

6.3.1 REWORK DISCOVERY TIME DELAY

As originally modelled, the *rework discovery* flow rate is parameterized as a single value. Cooper (1993b) connects increasing this flow of work through discovering errors in work products or rework earlier in the design process (*e.g.* before system testing commences). In this case, the time delay at which this flow of rework begins is reduced. Cooper also discusses improving the rate of rework discovery through decreasing the time delay between discovering errors in work products or rework. Figure 6.7 illustrates these two forms of rework discovery improvement.

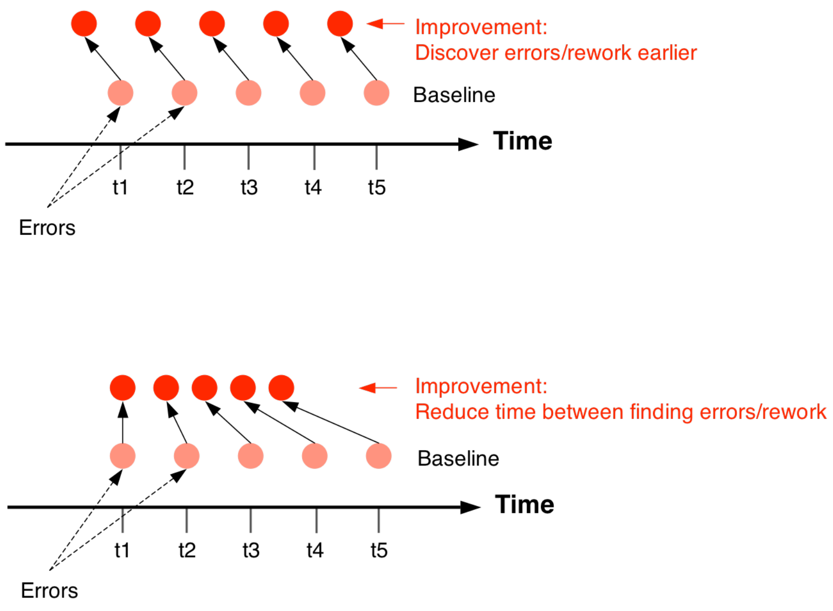


Figure 6.7: Rework discovery time delay improvement

However, identifying more rework at a time can also improve the rework discovery flow rate in that errors are interdependent and rework can lead to additional, knock-on rework. Figure 6.8 shows this type of improvement of the rework discovery flow.

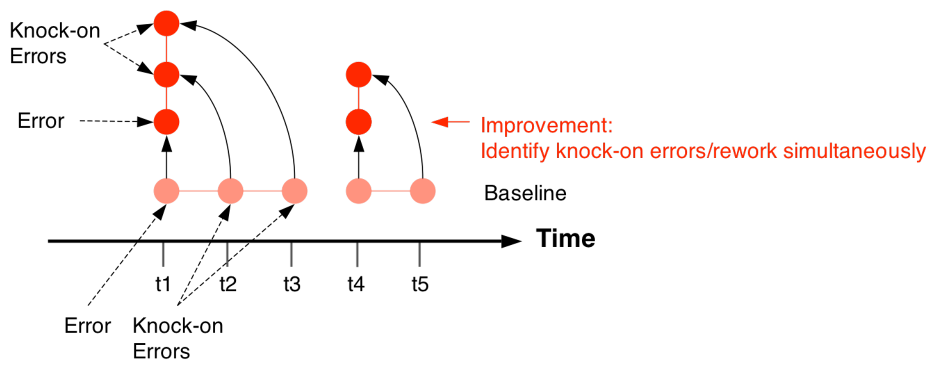


Figure 6.8: Rework discovery improvement through identifying knock-on rework

In turn, if treating this model as a discrete system, two parameters can influence the *rework discovery* flow rate. The *rework discovery time delay* specifies the points in time at which the valve opens to allow the stock of work to flow and errors are identified, while *IA quality* determines how many interdependent, knock-on errors are found while the valve is open. This model interpretation indicates that rework is identified at distinct time intervals. As such, the discrete *rework discovery time delay* parameter essentially corresponds with the times of IA application to discover rework. Notably, more than one IA technique (Section 2.5) can be used in combination for these applications of IA. Consequently, the rework discovery flow accounts for the result of using IA techniques of varying rigour, but does not distinguish IA rigour (Section 4.3) within the model. This conceptualisation of IA suggests that the stock of rework can be modelled as discrete *changes*[38].

For the adapted rework cycle model, reducing the rework discovery time delay shifts the applications of IA uniformly earlier in time, corresponding with one of Cooper's intentions for improving the rework discovery flow (Figure 6.7). However, the rework discovery time delay does not reduce the time intervals between performing IA. This definition would couple the rework discovery time delay to the IA quality parameter, as suggested in Section 6.4.1[39]. Nevertheless, uniformly shifting the applications of IA is representative of a policy shift to defining when and how IA should be used during design processes.

[38] These changes can be due to "errors", as done by Cooper (1993b) and Lyneis *et al.* (2001), but also can be generically classified as "initiated" and "emergent" changes, as suggested by Eckert *et al.* (2004) (Section 1.3). Section 6.3.4 makes this distinction.

[39] IA quality improvement depends on the span of changes over time throughout the design process. Consequently, modifying the time interval between changes can misconstrue the results of increasing IA quality.

6.3.2 IMPACT ANALYSIS QUALITY

IA quality represents the scope of related rework identified per application of IA and can be interpreted through the identification of sets of changes, including an *initiating*[40] change and its associated *knock-on* changes. An initiating change is the first modification that is identified in the set, and knock-on changes modify the design to correspond with and fulfil this change. Increasing IA quality indicates identifying knock-on changes during earlier applications of IA with other associated modifications in the set. Figure 6.9 portrays IA quality improvement for a change set.

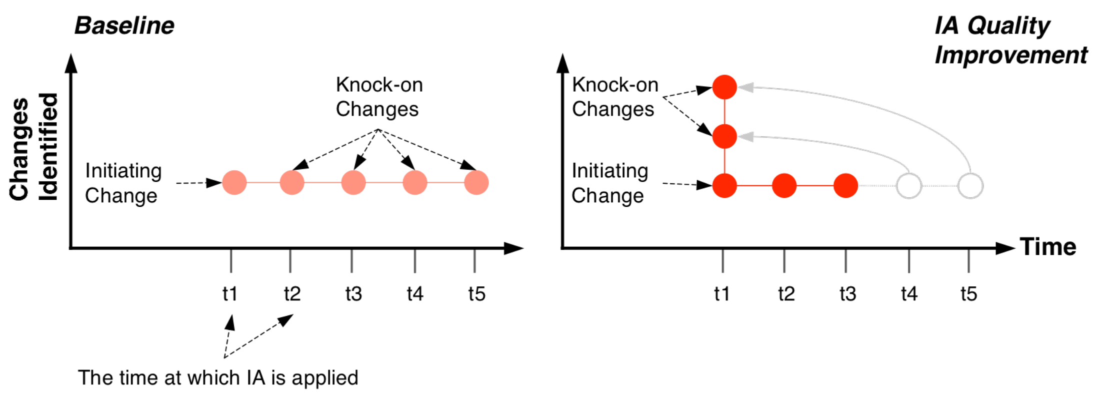


Figure 6.9: IA quality improvement for a change set

As defined by the IA characterisation in Figure 4.2, IA quality can be interpreted through the completeness, correctness, and clarity of IA results. However, the *IA quality* parameter in the adapted rework cycle only focuses on the completeness of IA results (Limitation 1 in Table 6.1) in that increasing IA quality identifies a more comprehensive set of changes.

Distinguishing the size of such sets of related modifications provides a means to measure IA quality improvement; without delineating the rework that could potentially be found earlier, the IA quality parameter has little meaning. As such, the number of modifications included within change sets influences the IA quality parameter interpretation. IA quality can represent finding direct, *first-order*, knock-on effects if only using a small number of modifications to form sets of related changes. Allocating a larger number of modifications within change sets could model indirect, *higher-order*, knock-on modifications[41]. As interpreted within the change database from the aerospace company (Appendix D) used to quantitatively depict the behaviour of adapted rework cycle for this firm (Section 6.4), only changes

[40] *Initiating* changes differ from "initiated" changes defined by Eckert *et al.* (2004).
[41] Section 2.5.1 also highlights the *order* of knock-on effects in terms of *completeness*.

stemming from a specific design area within a functional area (Section 3.4.1) are considered for classification into change sets. Limiting the potential scope of sets of changes confines the analysis to first-order, knock-on changes. The simulation of the adapted rework cycle (Section 7.1) similarly represents IA quality through small change sets (Limitation 2 in Table 6.1).

In order to capture a larger scope of knock-on effects occurring to more than one design area, the adapted rework cycle would require the input of a specific model of functional and design area dependencies. Given that functional and design areas are coupled to different extents, this model could more accurately account for the potential scope of knock-on changes. By assuming the limited scope for initiating and knock-on changes in the simulation of the adapted rework cycle (Section 7.1), such a model is not required since an estimation can be made on the scope of the change sets within design areas (Limitation 3 in Table 6.1). However, these simulation results only indicate how improving IA for first-order changes affects the design process. Nevertheless, this simulation suggests the potential consequences of IA improvement to address the fourth research question (Figure 5.12), and incorporating a product model into the simulation is left for future work (Section 9.3).

The IA quality parameter, which has a fractional value, depicts the scope rework identified per application of IA through the number of modifications identified in the change sets. An IA quality value of 1 indicates that all changes in the set are identified, while an IA quality value of 0.5 represents that half of the remaining changes necessary in the set is found during each application of IA. Figure 6.10 depicts these example parameter values. This fractional definition of IA quality corresponds with the description of the quality of work[42] parameter by Cooper (1993b) in that both indicate that the output of either implementing work or identifying changes is always partial (*i.e.* due to procedures, resources available, etc.). Interviewees at the aerospace company (I-2, I-12, I-15, I-16) also corroborate this definition by their description of finding portions of remaining necessary changes during each IA application. Notably, the same calculation of the value for IA quality applies to different scopes of initiating and knock-on changes, but, as suggested, these values for IA quality have different meanings (*i.e.* for first, second, or third order knock-on changes).

[42] Only fractions of rework identified and re-implemented enter the stock of work really done. The remaining fraction of rework implemented contains errors and is completed during subsequent iterations around the rework cycle.

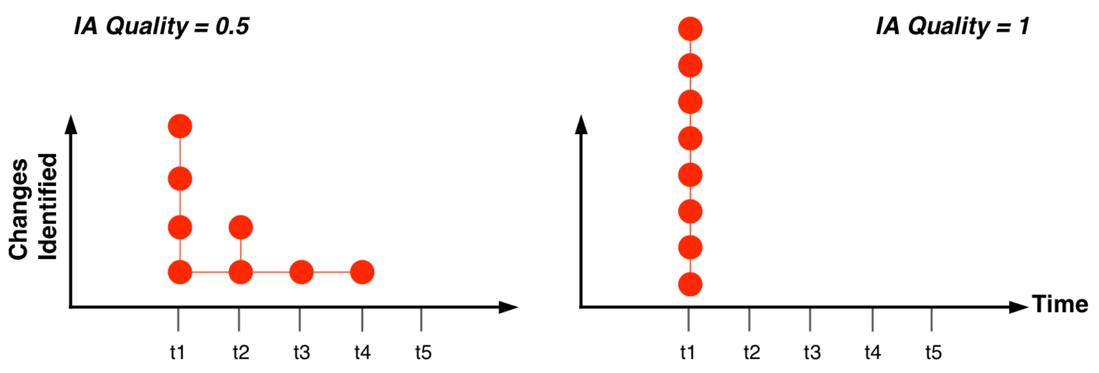


Figure 6.10: Examples of IA quality parameter values

This definition of IA quality in terms of the number of changes assumes that identifying more modifications proportionately increases the amount of rework discovered (Limitation 4 in Table 6.1) and, in turn, the *rework discovery* flow (Figure 6.6). However, the granularity of the changes within sets affects this flow. Changes could be defined at a low level, and change sets could consist of many modifications requiring little rework, such as the correction of spelling mistakes or the modification of a specific line of code. In contrast, high-level changes can include incorporating new functionality into the product design, and such a modification could include many unspecified detail changes and have a large amount of associated rework. Sets containing both high and low-level changes can misconstrue the interpretation of the IA quality value since more rework may be associated with identifying some changes than others. In the data analysis depicting the adapted rework cycle behaviour (Section 6.4), the sets of changes include modifications with different "functional impact" values, which estimate the amount of rework required (Appendix D). However, given that these changes are not defined at extremely high or low levels, representing IA quality according to the number of changes found can approximate the increase in rework found. Alternatively defining IA quality according to the amount of rework identified (*i.e.* through the functional impact values) could limit the investigation of IA improvement. For instance, identifying half of the remaining rework due to an IA quality value of 0.5 could limit this amount of rework to certain changes that sum to the value of half the total functional impact of the change set. In turn, the simulation of the adapted rework cycle (Section 7.1) uses a similar functional impact distribution for changes as the data analysis and depicts IA quality improvement through identifying changes.

In summary, given the IA quality parameterization, the implementation of work products generates a stock of work really done and sets of related changes (Figure 6.11).

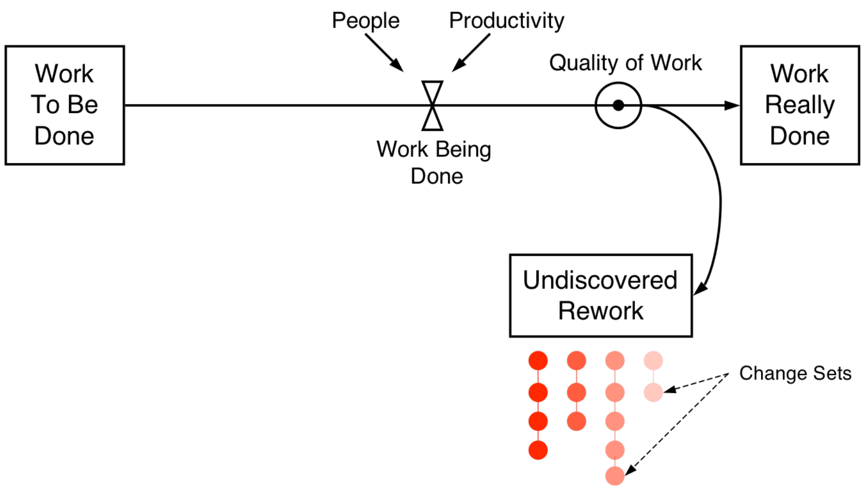


Figure 6.11: Changes spawned during the implementation of work

In turn, a fraction of each change set, depending on the IA quality value, is discovered during an application of IA (Figure 6.12).

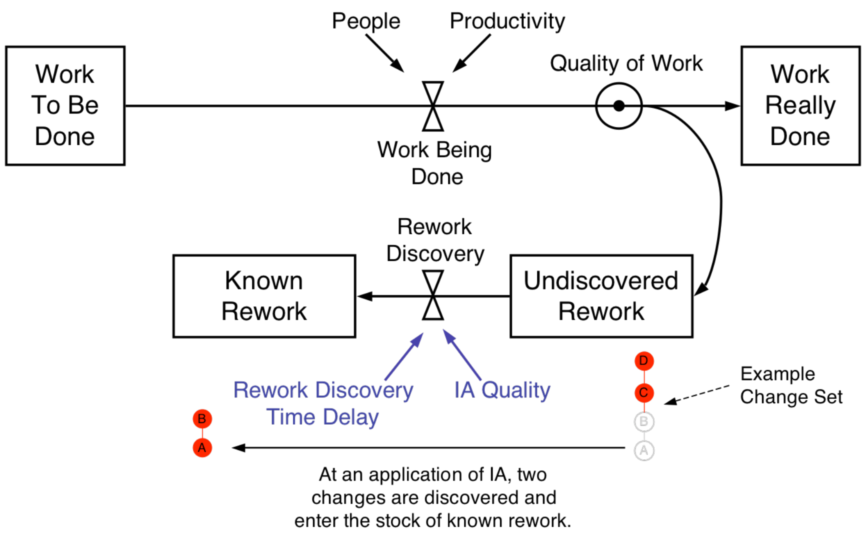


Figure 6.12: Changes discovered through IA

These identified changes are then implemented as rework. Some of this rework performed enters the stock of work really done, while the other fraction includes design errors, denoted by the quality of work parameter. In turn, this erroneous work spawns additional changes (Figure 6.13).

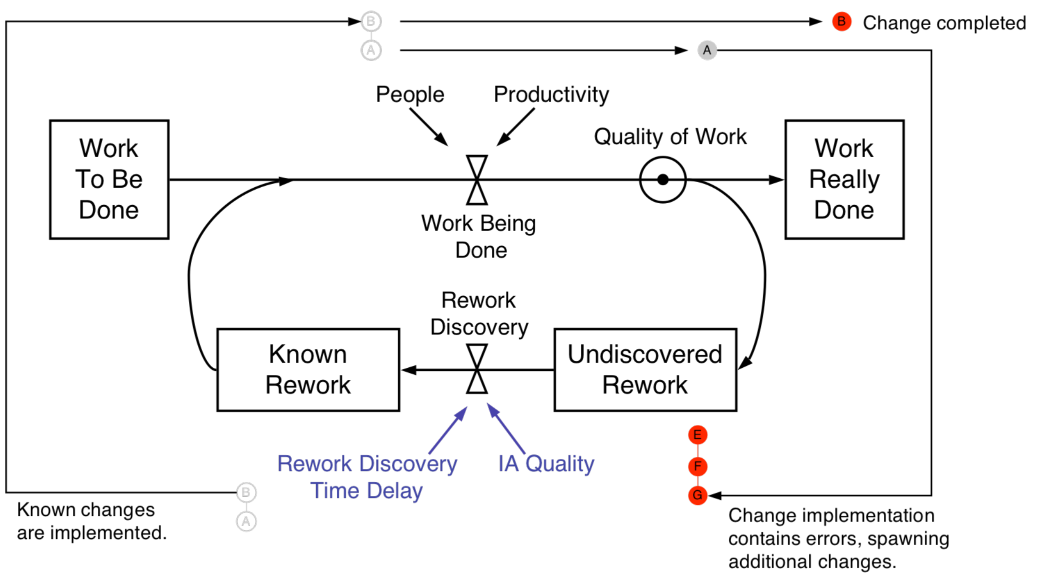


Figure 6.13: Implementation of changes

These spawned changes are assumed to be independent of the original sets of modifications initiated. If they were to be considered knock-on changes to these change sets, then the quality of work variable is effectively coupled to the IA quality parameter since the quality of work affects the size of the change sets, which influences IA quality. Notably, the rework discovery time delay parameter also is not considered to influence IA quality. Smaller rework discovery time delays can occur without a degradation of IA quality. It is assumed that associated knock-on changes can be discovered after finding an initiating change, defining the size of the change set. In turn, this decoupling of the rework discovery time delay, IA quality, and quality of work parameters limits the scope of initiating and knock-on change sets. Using large sets can invalidate these assumptions since errors occurring during work could be included as knock-on changes after early applications of IA, varying the size of the change sets and when changes could be discovered (Limitation 5 in Table 6.1).

6.3.3 CHANGE PACKAGING

Given the discretization of the rework cycle model, modelling change packaging becomes possible (Section 6.1.2). As illustrated in the adapted rework cycle model (Figure 6.6), the *change packaging* valve regulates the flow of rework into the stream of work being done. The stocks of work to be done and known rework can both influence the planning of work. In particular, a change to a design artefact may be packaged with other work to be done on the same artefact or delayed upon expecting additional, related rework. Less rework is expected to occur with effective change packaging, as indicated by the empirical studies (Section 3.4.3), since the interaction among design work is better understood and fewer errors occur. The simulation of the adapted rework cycle embodies this perspective in that fewer changes are spawned due to the quality of work from packaged modifications (Section 7.5).

As suggested in the discussion of IA quality (Section 6.3.2), the change packaging implemented in the simulation of the adapted rework cycle does not rely on a model of functional or specific design areas dependencies, and packaging only occurs for changes within functional areas. Given the limited scope of the initiating and knock-on changes modelled, these modification sets are assumed to equally occur throughout all functional areas (Limitation 6 in Table 6.1). As such, the simulation of the aerospace company's design process randomly assigns changes to functional areas according to the volatility of each functional area, as depicted by the change database analysed (Appendix D). More volatile areas are assigned a larger portion of all changes. The change packaging policies simulated groups changes within these functional areas.

6.3.4 LIMITATIONS

Even though the adapted rework cycle is simplistic, this model depicts sufficient parameters to suit the research questions and examine the consequences of IA improvement (Figure 5.12) and change packaging (Figure 5.14) strategies on design processes. Specifically, the practical IA improvement strategies (Section 5.4) can be investigated in that the rework discovery time delay and IA quality parameters provide a means to model performing more IA through a defined method (*i.e.* improving the rework discovery time delay and IA quality), improving IA through better information sharing (*i.e.* improving IA quality) and allocating more resources and time to IA (*i.e.* improving the rework discovery time delay and IA quality) (Section 7.4.1). Perturbing the values of these parameters and simulating the adapted rework cycle model indicates the relative effects of these strategies, as suggested in Section 6.1.1.

However, as described by the modelling and simulation method applied (Step 1 in Figure 6.4), the limitations of the adapted rework cycle also must be defined in order to fully articulate the model and facilitate the interpretation of the simulation results. Although several limitations have been described in conjunction with the description of the adapted rework cycle model in this section, additional limitations exist beyond these previously mentioned. For instance, this model does not explicitly depict the specific interplay between design and change processes through design task dependencies, task durations, resource allocation, or information availability

(Limitation 7 in Table 6.1) as other models (Browning *et al.* 2006). Nevertheless, the adapted rework cycle can indicate the high-level consequences of IA improvement on design processes and does not intend to forecast the specific effects of IA improvement and change packaging on a particular software development project.

The adapted rework cycle also does not account for changes stemming from "added" and "obsoleted" work, as described by Lyneis *et al.* (2001) and Sterman (2000: 58-59). The addition of new customer requirements or, more generally, "initiated" changes from Eckert *et al.* (2004) (Section 1.3), can cause stock from work really done to require rework during development processes. Such obsolescence can lead to an additional feedback loop, shown in Figure 6.14.

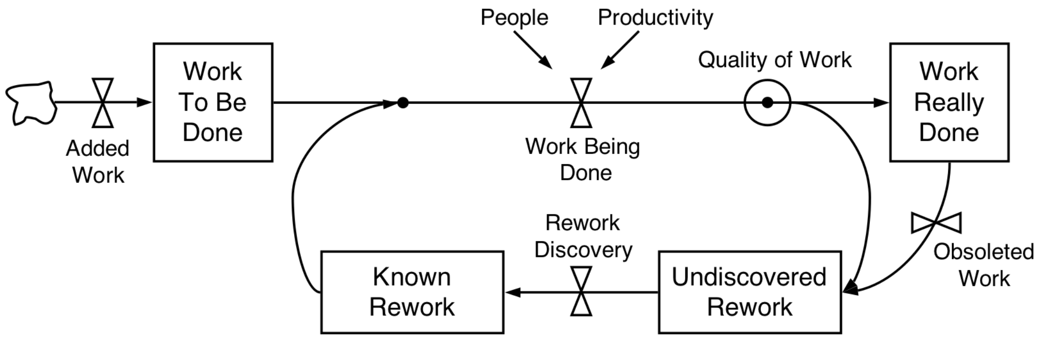


Figure 6.14: The rework cycle with the addition and obsolescence of work

Given this perspective on initiated changes, "emergent" changes as defined by Eckert *et al.* (2004) correlate with the "errors" flowing into the feedback loop described in the rework cycle (Section 6.1.1). The associated modifications of these errors can be considered emergent changes since they stem from the state of the design. Notably, these changes do not correspond with the knock-on changes described by the IA quality parameter (Section 6.3.2). The IA quality parameter distinguishes between sets of related changes in order to conceptualise IA quality improvement. In turn, the modelling of the addition of work in Figure 6.14 is discussed in regard to future work (Section 9.3) to account for IA occurring prior to extending the scope of projects.

Given that the aerospace company's design process does not involve the significant addition or obsolescence of work and the associated product platform is relatively stable, the simulation (Section 7.1) focuses on the principal feedback loop depicted by the adapted rework cycle model (Figure 6.6). Consequently, the results obtained do not account for changes due to obsolescence (Limitation 8 in Table 6.1). Additional data to develop reference modes (Section 6.4) should be collected to appropriately investigate and validate the response of the adapted rework cycle model to dual feedback loops in future work. Moreover, as simulated, the adapted rework cycle does not model the stock of work to be done. The simulation only represents the changes or rework within the design process (Limitation 9 in Table 6.1) since most of the work done at the aerospace company is to implement changes to adapt the product platform functionality. Consequently, the results of the simulation of the adapted rework cycle are indicative when the workflow through the feedback loop is greater than the workflow directly from the stock of work to do. Finally, the simulation of the adapted rework cycle does not model the people and productivity parameters and assumes that such resources do not restrict the workflow (Limitation 10 in Table 6.1). In future work, additional data should be used to appropriately model the rate of work implementation for changes within specific functional areas. Table 6.1 reviews the limitations of the adapted rework cycle model and simulation discussed thus far.

Table 6.1: Summary of limitations of the adapted rework cycle

Limitation	Reference	Affecting	Description
1	Section 6.3.2	Model	The application of IA associated with the IA quality parameter is assumed to produce correct and clear, but possibly incomplete results.
2	Section 6.3.2	Simulation Implementation	For the IA quality parameter, sets of initiating and knock-on changes are scoped within a specific design area.
3	Section 6.3.2	Model	The interdependencies within the product design are not modelled, limiting initiating and knock-on changes sets to small scopes.
4	Section 6.3.2	Model	The IA quality parameter assumes that the number of changes identified corresponds to the amount of rework.
5	Section 6.3.2	Model	The scope of initiating and knock-on change sets cannot be large to ensure that rework discovery time delay, IA quality, and quality of work parameters are decoupled.
6	Section 6.3.3	Simulation Implementation	For the change packaging flow, the portion of changes within each functional area is estimated.
7	Section 6.3.4	Model	The adapted rework cycle does not model task dependencies, task durations, resource allocation, or information availability for a specific design process.
8	Section 6.3.4	Simulation Implementation	The addition of new work and obsolescence of work are not simulated.
9	Section 6.3.4	Simulation Implementation	The simulation only captures the implementation progress of changes or rework.
10	Section 6.3.4	Simulation Implementation	People and productivity do not restrict the flow of work as simulated.

Despite these limitations of the adapted rework cycle, the modelling and simulation implemented can suggest the potential consequences of IA improvement on the aerospace company's design process. Limitation 1 through 5 in Table 6.1 primarily define the appropriate interpretation of the IA quality parameter and do not restrict the applicability of the conclusions drawn from the modelling and simulation results. In turn, Limitation 6, Limitation 8, and Limitation 9 are reasonable assumptions in that they do not affect the fundamental characteristics of the aerospace company's design process and, thus, do not significantly influence the trends observed through simulation. As such, the adapted rework cycle implemented does not intend to forecast or predict concrete improvements to the aerospace company's design process simulated and focuses on analysing the high-level trade-offs between the modelled parameters. Limitation 7 is also acceptable given that a high-level model and analysis is appropriate to address the research questions to investigate IA improvement (Section 6.1.1). Finally, Limitation 10 restricts the simulation to the trade-offs between the rework discovery time delay, IA quality, and quality of work parameters. However, these parameters are sufficient to evaluate and compare different IA improvement strategies for the aerospace company against each other through simulation since the implementation of software changes is not primarily limited by the people available or this staff's productivity in that a backlog of software design work does not occur in practice due to such a lack of resources. Even though the people and productivity parameters are not simulated, the influences of these variables are accounted for in the heuristics for process improvement developed (Section 8.2), which depict the trade-offs between IA improvement and other process improvement strategies. As such, the other simulation implementation assumptions (Limitation 2, Limitation 6, Limitation 8, and Limitation 9 in Table 6.1) are also addressed when developing these heuristics, and the refinements of these assumptions are discussed as future work (Section 9.3).

Accordingly, the adapted rework cycle meets the requirements to address the research questions in that the high-level trade-offs between process improvement strategies can be investigated, and the limitations of the adapted rework cycle are reasonable and handled within the conclusions drawn from the modelling and simulation results.

6.3.5 EVALUATION OF THE ADAPTED REWORK CYCLE MODEL

As shown within the modelling and simulation method applied (Step 1 in Figure 6.4), evaluation of system dynamics and software process dynamics models is required in order to determine their applicability to policy making. Like other model-based methodologies, evaluating these models involves both *informal* and *formal* aspects, as depicted by Barlas (1996). Specifically, Barlas describes informal model validity as "usefulness with respect to some purpose". As discussed in Section 6.3.4, the adapted rework cycle includes the necessary parameters to investigate the research questions derived from industry (Section 5.6) and, thus, meets this requirement.

In turn, Barlas (1996) categorises the formal validation to include "direct structure tests", "structure-oriented behaviour tests", and "behaviour pattern tests". Direct structure tests involve comparing the model structure against the real system, providing for evaluating the model formulation in the modelling and simulation method applied (Step 1 in Figure 6.4). In particular, each dependency in a stock and flow diagram should be analysed individually using knowledge about these relationships in reality. As such, given that the rework cycle has been validated across many companies and industries (Cooper 1993b), the modelling of IA quality and change packaging in the adapted rework cycle only needs to be reflected on against industry practice. In turn, discussions with industry interviewees and academics familiar with the rework cycle confirm that the modelling of these parameters is logical and fits with practice. Specifically, individuals at the aerospace company (I-2, I-12, I-15, I-16) confirm that IA quality fits with their experience in practice (Section 6.3.2), and interviewees (I-2, I-16) at this company also accept the modelling of change packaging. Additionally, an interviewee at the telecommunications firm (I-39) interpreted the relationships represented in the adapted rework cycle for both systems engineering and agile development processes and indicated that the relationships held for both contexts. In turn, a system dynamics professor indicated the usefulness of including IA quality, but also encouraged the extension of the model to depict the coupling of change sets (Limitation 3 in Table 6.1 and Section 9.3). Finally, Cooper, who developed the rework cycle (Section 6.1.1), encouraged the application of the rework cycle to include change packaging.

The testing of the adapted rework cycle simulation in the modelling and simulation method applied (Step 3 of Figure 6.4) encompasses validation through structure-oriented behaviour tests, which focus on comparing model behaviour against specific expected results (Barlas 1996). Behaviour pattern tests, which determine the accuracy of the model to represent the system as a whole, could also be performed on the simulation, but are left as future work since these tests focus on predictive models that indicate specific, numerical improvements (Section 9.3). The adapted rework cycle simulation does not produce such results since some model parameters are not included in the simulation (Table 6.1). Table 6.2 suggests potential empirical data sources that could be used to perform behaviour pattern tests on a predictive model.

Table 6.2: Potential empirical data sources for a predictive model

Model Parameter	Source Data
Rework Discovery Time Delay	Change Database – The rework discovery time delay can be estimated by correlating the logged times of initiating and related knock-on modifications.
IA Quality	Change Database – IA quality can be approximated by mapping the distributions of knock-on changes logged across time.
Quality of Work	Change Database and Designers' Time Reporting – The quality of work can be assessed by cross-referencing the amount of time used for rework with the amount of time used for work.
People	Designers' Time Reporting – The people working on a project can be analysed using the time logged by employees for work hours.
Productivity	Designers' Time Reporting – Productivity can be estimated by comparing the amount of time worked and the amount of work produced (*e.g.* size of work packages).

6.4 BEHAVIOUR OF THE ADAPTED REWORK CYCLE

Given the description and evaluation of the adapted rework cycle model (Section 6.3), the behaviour of this model over time can now be examined to fulfil the model articulation for Step 1 in the modelling and simulation method applied (Figure 6.4). *Reference modes* capture the intended behaviour of a system modelled and are used as a means to calibrate and test the simulation results in Step 3 in the modelling and simulation method applied. In turn, reference modes are typically depicted through graphs of relevant parameters across a period of time, or *time horizon* (Sterman 2000: 160). Such graphs often are constructed by analysing quantitative information available. However, descriptions from empirical studies or even from literature can also be used to graph or describe the relevant parameters qualitatively (Kellner *et al.* 1999).

As such, the reference modes (Section 6.4.3) for the adapted rework cycle are developed from the qualitative behaviour of this model (Section 6.4.1), which is based on the original rework cycle behaviour depicted by Cooper (1993b), and from quantitative information from a software change database provided by the aerospace company (Section 6.4.2). This change database holds the details of all the requested modifications submitted to the control group CCB (Section 3.4.2.1) for a recently completed project, which differs from the 42 changes for an on-going project discussed in Chapter 5. Appendix D describes the fields of information captured in the database for nearly 1600 software changes.

6.4.1 QUALITATIVE BEHAVIOUR OF THE ADAPTED REWORK CYCLE

Cooper (1993c) depicts the behaviour of the rework cycle (Figure 6.1) qualitatively as shown in Figure 6.15. As illustrated in Figure 6.15 (left), the stock of work to be done decreases as design projects progress, and the stock of undiscovered rework increases. As the stock of undiscovered work is identified, the stock of known rework increases. Cooper indicates that the identification of the undiscovered rework stock tends to occur during "downstream" project phases, such as testing. As such, the known rework curve is delayed (*i.e.* positioned to the right) from the undiscovered rework curve. The stock of known rework eventually is depleted based on the rate of work being done.

Cooper further interprets this behaviour in terms of the percentage of work perceived and actually complete, as shown in Figure 6.15 (right). The perceived percentage of work done for projects does not include the undiscovered rework. As such, as more rework becomes known, the progress of perceived work complete slows. The work really done curve in Figure 6.15 (right) accounts for the undiscovered rework and, thus, is always less than the perceived work done curve.

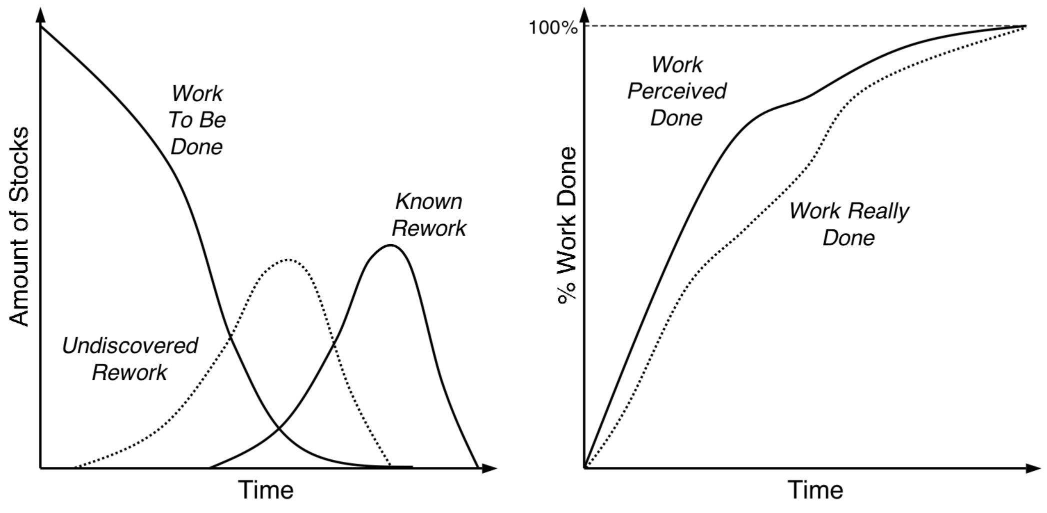


Figure 6.15: Behaviour of the rework cycle (Cooper 1993c)

On this premise, the behaviour of the adapted rework cycle can be constructed qualitatively. The adapted rework cycle introduces the rework discovery time delay and IA quality as parameters (Section 6.3), which influence the known rework curve in Figure 6.15 (left). Figure 6.16 illustrates the expected shifts in this curve based on reducing the rework discovery time delay and increasing IA quality.

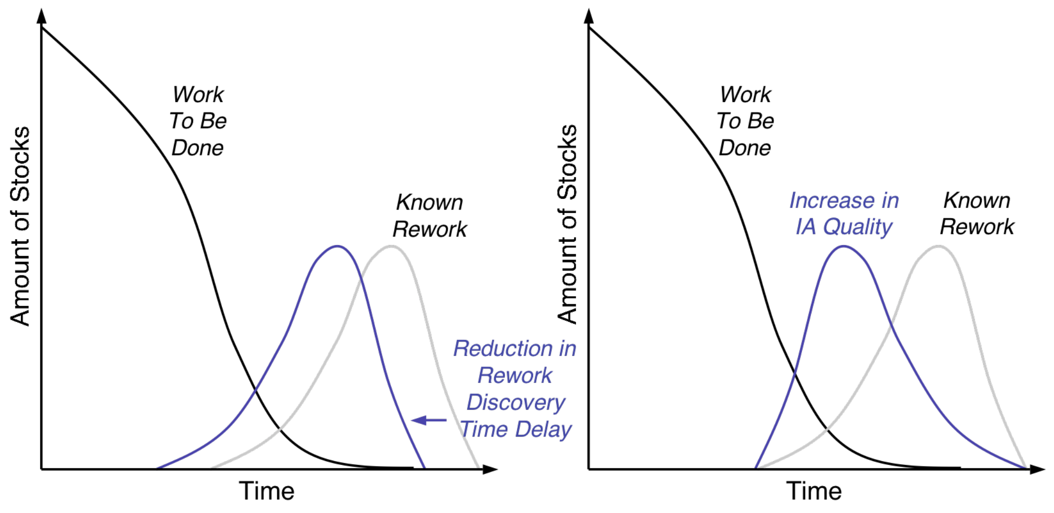


Figure 6.16: Behaviour of the adapted rework cycle – known rework curve shifts

Reducing the rework discovery time delay (Section 6.3.1) should shift the known rework curve earlier in the design process as shown in Figure 6.16 (left). Identifying rework can occur earlier by actively performing IA earlier in product development. In turn, improving IA quality (Section 6.3.2) can increase the amount of rework found at each application of IA. As depicted in Figure 6.16 (right), this improvement corresponds to finding more knock-on changes and the associated rework earlier.

The transformation of the known rework curve due to increased IA quality in Figure 6.16 (right) is notional and specifically depends on (1) the relative amount of rework within the sets versus the total amount of rework and the distribution of (2) initiating and (3) knock-on changes during a design process. The total amount of rework is contained within *independent* changes (*i.e.* initiating changes with no knock-on effects) and modifications in change sets. If change sets are a significant portion of all rework, then improving IA quality can shift a large percentage of knock-on rework earlier. In turn, only a few change sets would negligibly influence the known rework curve.

Assuming decoupled rework discovery time delay and IA quality parameters, as described within the limitations of the adapted rework cycle (Table 6.1), initiating changes from which knock-on modifications stem and independent changes cannot necessarily be found earlier due to IA quality improvement; only reducing the rework discovery time delay can affect the identification of these modifications. Hence, at best, all knock-on rework can be discovered at the time the associated initiating change is found due to improved IA quality. If initiating changes occur mostly towards the onset of rework discovery and no initiating or independent changes occur at the end of the design process, then the duration for finding all rework (*i.e.* the duration of the known rework curve) could be shortened by improving IA quality. Alternatively, if initiating changes occur throughout product development, the process duration may not decrease significantly due to IA quality improvement. Nonetheless, more rework can be found earlier in the design process through increasing IA quality.

Finally, the distribution of knock-on changes after an initiating modification also affects the potential shift of the known rework curve due to IA quality improvement. If knock-on changes in a set occur nearly simultaneously at the end of a span of time from an initiating change, then increasing IA quality can readily decrease the time when most of the rework for the set is known. Alternatively, if these knock-on changes are distributed more uniformly throughout the same span of time, then this mean time does not necessarily decrease as significantly. Translating this behaviour to design processes suggests that the shape of the known rework curve can readily change due to IA improvement when knock-on changes occur close together in their sets. However, if the time span for which knock-on changes occur from an initiating change is small, then the shift in rework may not significantly influence the known rework curve as such. In turn, both the time span for knock-on changes and the distribution of knock-on changes during this time affect IA quality improvement.

Reducing the rework discovery time delay can affect project duration and increase the rate of work done, as described by Cooper (1993b), since the "scheduling" of rework can be improved. Similarly, since more rework is known earlier due to increasing IA quality, the scheduling of this rework also can be planned. Figure 6.17 illustrates this improvement in work progress.

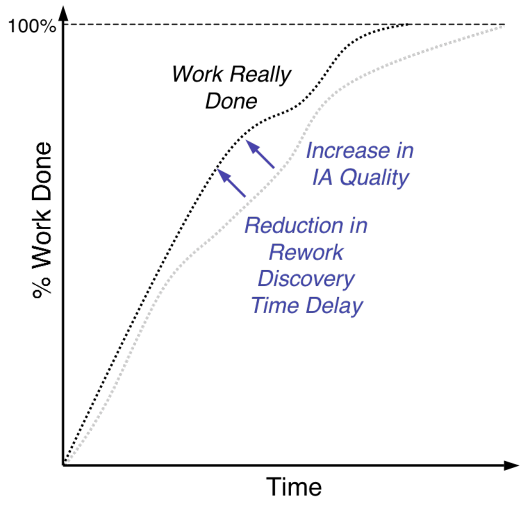


Figure 6.17: Behaviour of the adapted rework cycle – work done curve shift

6.4.2 QUANTITATIVE BEHAVIOUR OF THE ADAPTED REWORK CYCLE

This discussion of the adaptive rework cycle's qualitative behaviour provides the basis to describe the quantitative behaviour of this model specifically for the aerospace company. Through the analysis of a change data set (Appendix D) obtained from the aerospace company, potential shifts in the known rework curve due to IA quality improvement, qualitatively depicted in Figure 6.16, are explored. In turn, the investigation of these curve shifts leads to the development of the reference modes (Section 6.4.3) used in the calibration and testing of the simulation (Step 3 in the modelling and simulation method applied, Figure 6.4). Shifts in the other curves describing the behaviour of the rework cycle in Figure 6.15 are not investigated given that the data available only captures information regarding found changes at the aerospace company and that the simulation focuses on the handling of rework (Limitation 9 in Table 6.1).

For comparison with the change data set, the known rework curve shift due to increasing IA quality in Figure 6.16 can be translated into the qualitative graph in Figure 6.18.

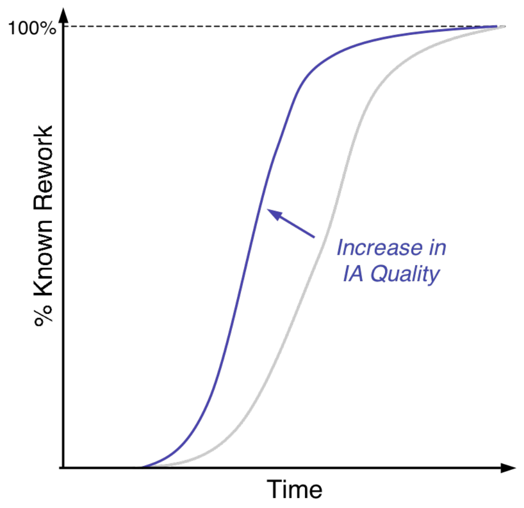


Figure 6.18: Representation of known rework curve (from Figure 6.16) for data analysis comparison

In turn, the information in the change database (Appendix D) can portray Figure 6.18 quantitatively, using the key variables in Table 6.3.

Table 6.3: Key variables used in data analysis (from Appendix D)

Variable	Value Format
Time of Change Request	Date
Functional Impact of Change	Scale Rating (No Impact, Minor Impact, Major Impact)
Knock-On Change	Boolean

The change data set includes the time and functional impact of change requests, indicating the amount of work expected to implement the change. However, the data set does not specify if changes stem from other modifications and are knock-on changes. This parameter is necessary in order to estimate the potential increase in IA quality (Section 6.3.2). In turn, first-order, initiating and knock-on change groups are determined from the other data set parameters for 3 specific design areas of the control system, as discussed in Appendix E, and all changes within these 3 design areas are assigned a Boolean value indicating if they are or are not part of such a knock-on change group. Section 2.5.1 discusses this deterministic (as opposed to probabilistic) approach to classifying changes. The changes within these 3 change data subsets are used to depict 3 example known rework curves, which are used to develop the reference modes (Section 6.4.3).

In order to plot known rework curves for the example design area changes, analogous to that in Figure 6.18, the percent of known rework is interpreted as the percent of functional impact estimated at the time of each change request. As such, the functional impact of each change is converted to a numerical value according to the scale[43] in Table 6.4. Table 6.5 uses this conversion to describe key characteristics of the 3 example design area changes. Subsequently, these characteristics also enable the reference mode development (Section 6.4.3) and define the variables values used in the simulation (Section 7.2).

[43] The trends observed in the example known rework curve shifts are insensitive to the numerical scale chosen.

Table 6.4: Functional impact numerical value conversion

Functional Impact Rating	Numerical Value
No Impact	0
Minor Impact	5
Major Impact	10

Table 6.5: Example design area descriptions

Example	Design Area Changes Description
1	28 changes total, spans the first half of the change database duration (2.5 years) 36% of changes are within change sets 38% of the total functional impact is within change sets About 6 independent modifications per 1 change set 2 knock-on modifications per change set on average 25% of changes are knock-on changes
2	77 changes total, spans most of the change database duration (2.5 years) 56% of changes are within change sets 52% of the total functional impact is within change sets About 4 independent modifications per 1 change set 3 knock-on modifications per change set on average 42% of changes are knock-on changes
3	71 changes total, spans most of the change database duration (2.5 years) 51% of changes are within change sets 46% the total functional impact is within change sets About 4 independent modifications per 1 change set 3 knock-on modifications per change set on average 37% of changes are knock-on changes

By plotting all changes in the each of the 3 example design area change sets, baseline known rework curves are formed. The potential shifts in these curves due to increasing IA quality can be constructed by allocating the functional impact values of the knock-on changes to their respective initiating changes (Appendix E) and plotting all modifications except the knock-on changes. This process represents finding more of rework earlier. If all knock-on changes are analysed as such, this rearrangement of functional impact effectively corresponds to identifying the complete impact of every change at the first application of IA, and the IA quality parameter in the adapted rework cycle has a value of 1 (Section 6.3.2). Figure 6.19 show these maximum potential shifts in the known rework curves (from Figure 6.18) for each example data set (Table 6.5).

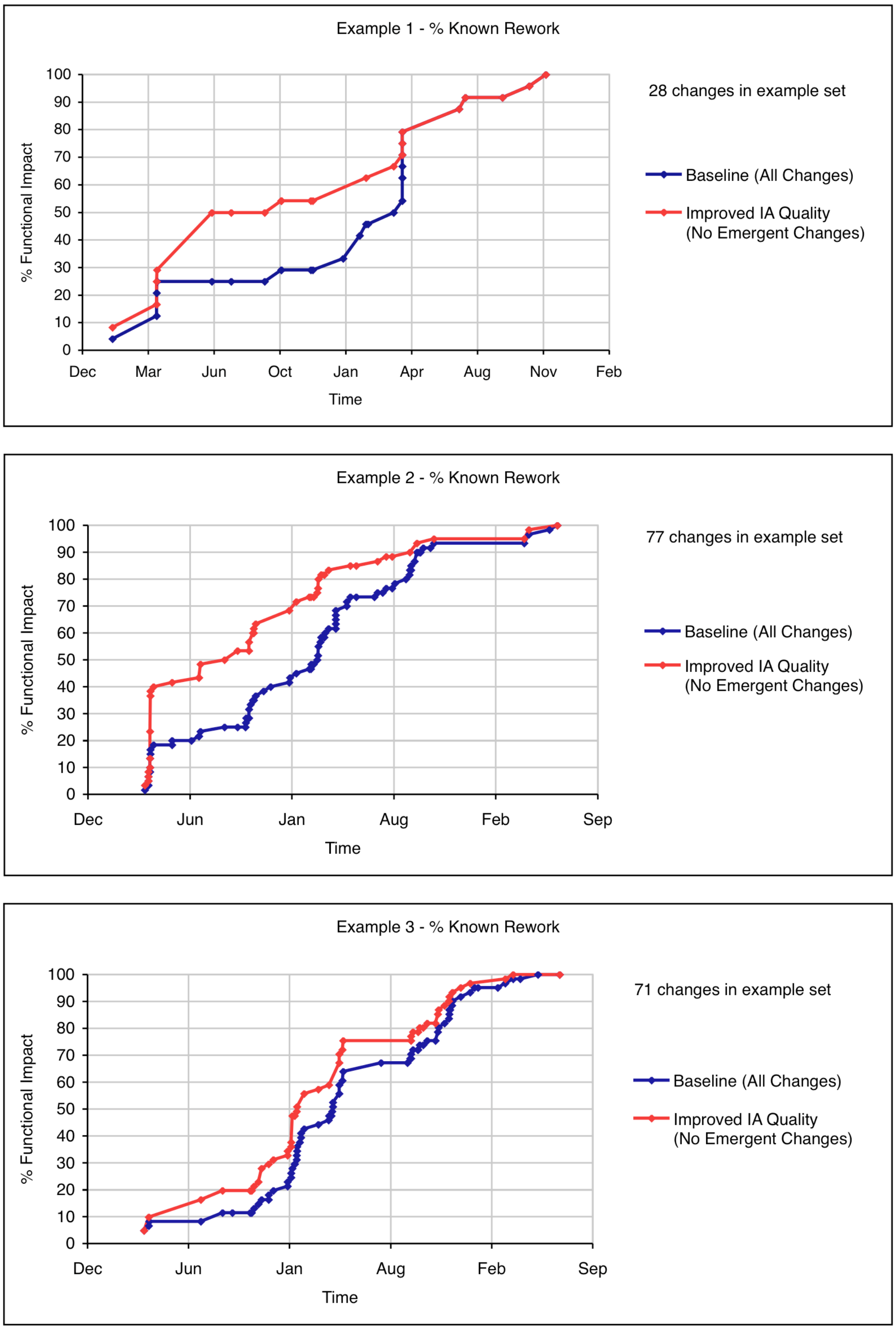


Figure 6.19: Examples of known rework curve shifts due to increasing IA quality from change database

As displayed in Figure 6.19, each example known rework curve has a distinct shape, which differs from the qualitative graph depicted in Figure 6.18. As previously mentioned in the discussion of Figure 6.16, these different shapes can be attributed to the amount and distribution of initiating and knock-on changes. Nevertheless, these examples have consistent key features that are drawn on to develop the reference modes (Section 6.4.3), which, in turn, are used to calibrate and test the simulation in Step 3 of the modelling and simulation method applied (Figure 6.4).

6.4.3 REFERENCE MODES FOR THE ADAPTED REWORK CYCLE SIMULATION

Assuming that these 3 example known rework curve shifts, which depict the potential IA quality improvement, are representative of the aerospace company's design process, the reference modes defining the intended simulation behaviour should embody their characteristics. Using the characteristics of other known rework curve shifts can lead to different interpretations of IA quality improvement and of the consequences of IA improvement strategies. The discussion of known rework curve shifts due to IA quality improvement specifies these characteristics (Section 6.4.1). Specifically, (1) the relative amount of rework within the sets versus the total amount of rework and the distribution of (2) initiating and (3) knock-on changes during a design process indicate the potential shift in the known rework curve. In effect, each of these characteristics as depicted by the example rework curves defines a reference mode for the simulation.

Firstly, the simulation should generate modifications representative of the ratio of independent changes to change sets (Table 6.5) in order to depict the first characteristic of the known rework curve. In particular, the estimated percentage of knock-on changes in the examples (Appendix E) should correspond with the simulation results. Figure 6.20 displays this reference mode.

Reference Mode 1:
The simulation should generate changes with a ratio of independent modifications to change sets and percentage of knock-on changes representative of the data analysis (Table 6.5).

Figure 6.20: Reference mode 1

Secondly, in all the example known rework curves (Figure 6.19), the distribution of initiating changes is consistent throughout the design process; initiating changes occur at the beginning and end of the design process, and the design process duration is not affected by the increase in IA quality. Figure 6.21 describes this reference mode.

Reference Mode 2:
The simulation should generate initiating changes throughout the design process.

Figure 6.21: Reference mode 2

Thirdly, knock-on changes are assumed to have a uniform distribution after an initiating modification since increasing the IA quality of large change sets (*e.g.* example 2 in Figure 6.19) does not cause a significant change in concavity of the curves. Furthermore, as observed in the data analysis (and demonstrated by the examples of Appendix E), knock-on changes for a change set can occur throughout the design process, either shortly after initiating modifications or within the last few changes implemented. As such, the third reference mode (Figure 6.22) specifies that the distribution of knock-on changes should uniformly span the remainder of the design process for each change set.

Reference Mode 3:
The simulation should generate knock-on changes for each change set that uniformly span the remainder of the design process.

Figure 6.22: Reference mode 3

Moreover, the distribution of all changes over the design process can affect the overall concavity of the known rework curve (Figure 6.18). Changes increasingly occurring throughout the design process can cause shifts in this curve that differ from a decrease in modifications identified as development progresses. As a result, a fourth reference mode calibrates the shifts in the known rework curve to the aerospace company's design process through the distribution of changes. Figure 6.23 illustrates the distribution of changes and functional impact across the aerospace company's design process. Based on this data, the fourth reference mode (Figure 6.24) assumes that changes occur at a constant baseline rate throughout the design process, but instances of increases in the number of changes are allowed from this baseline. Specifically, the maximum number of changes in any given month is allowed to be roughly 7 times the baseline number of changes that occurs in other months. (From the top histogram in Figure 6.23, the estimated baseline is approximately 20 changes with a maximum variation to about 140 changes.) Notably, the distributions in Figure 6.23 do not correspond with the patterns depicted by Eckert *et al.* in Figure 1.2. Based on interviews at the aerospace company, this project was affected by staffing and process changes. Project milestones also may have influenced the pattern of these distributions. As such, Figure 6.23 does not necessarily indicate a varying quality of work in that errors were generated more and less frequently, but rather suggests the processes for reporting design modifications changed.

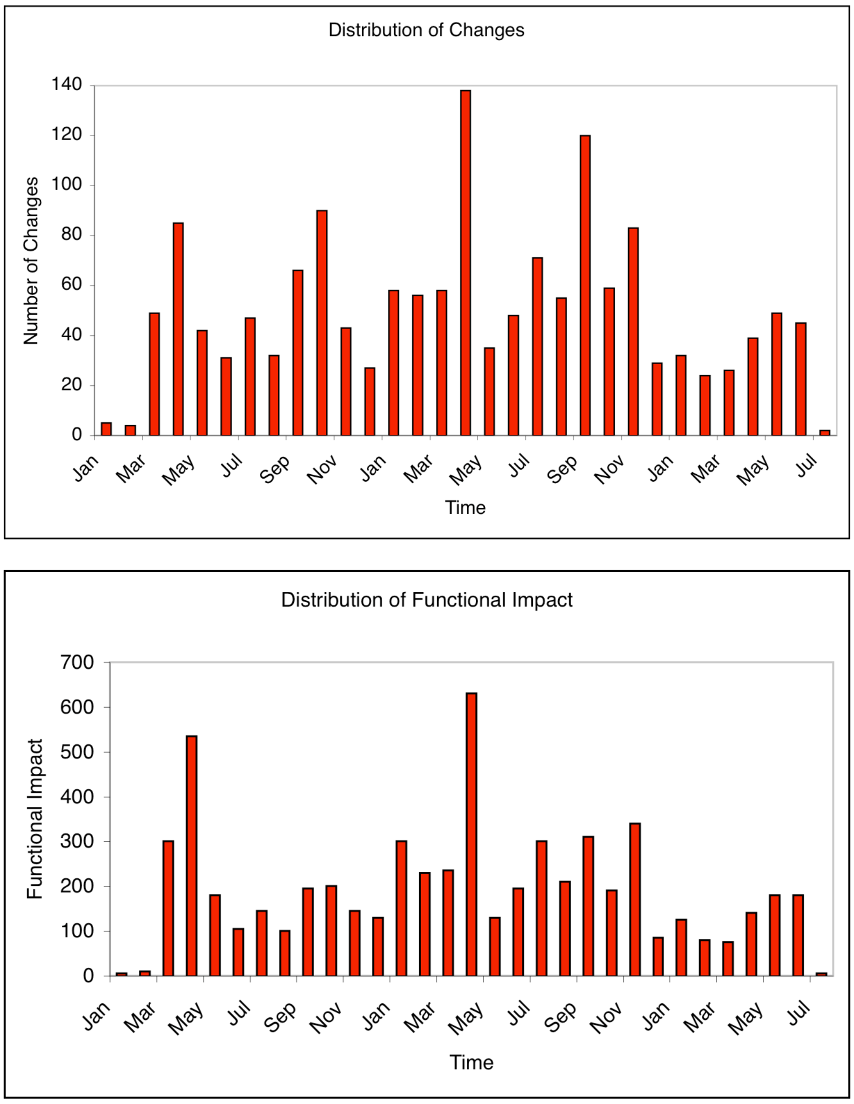


Figure 6.23: Distribution of changes over time at the aerospace company

Reference Mode 4:
The simulation should generate changes at a constant rate throughout the design process, but instances of increases in number of changes are allowed.

Figure 6.24: Reference mode 4

Basing the simulation on these 4 reference modes to represent the aerospace company's design process grounds the investigation of IA quality improvement. By defining how the simulation generates changes, the characteristics of the known rework curves are determined, and, thus, the potential effects of IA quality improvement are scoped.

Although the reference modes complete the model articulation for Step 1 of the modelling and simulation method applied (Figure 6.4), they only describe improvement in terms of shifts in the known rework curve. The translations of this curve can be correlated with transformations in the work really done curve (Figure 6.17) to establish the overall process improvement. Cooper's qualitative results from the rework cycle (1993b) suggest the implications of known rework curve shifts. He specifically indicates that the perceived percent of work complete is exponentially related to the percent of work really done[44] when the quality of work is low and provides the ranges of these relationships for several quality of work values based on his work across many projects. Figure 6.25 generically represents this relationship.

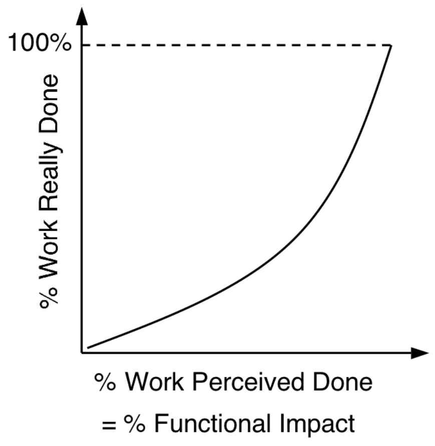


Figure 6.25: Work really done conversion

The percent of work perceived done is a function of the stocks of work to be done, known rework, and work really done, while the percent of work really done also factors in the stock of unknown rework with these elements. Given that the data analysis does not account for the work to be done, the percent of work perceived done is essentially a function of known rework, and the percent of work really done is characterised by known and unknown rework. In turn, the functional impact in the data set can describe the work perceived done, and, consequently, the work really done is related exponentially to the functional impact identified, as shown in

[44] The functional impact values used in the data analysis do not directly reflect the work done since they only capture the expected rework, and additional, unanticipated and unreported rework also occurs in practice to implement change requests (Appendix D).

Figure 6.25. In this case, the unknown rework represents the rework performed that is not captured in the change database.

Consequently, the work really done curves for the 3 examples in the data analysis can be estimated using an exponential conversion. Given a quality of work value of 0.5 (approximately the value at the aerospace company, Section 7.2), the exponential relationship in Figure 6.25 is approximately x^2, according to the simulations by Cooper (1993b). As such, the baseline known rework curves of the examples (Figure 6.19) can be translated to the corresponding exponential work really done curves. However, the improved IA curves with perfect IA quality are translated linearly to the associated work really done curves since no unexpected, emergent changes occur in these ideal cases. Hence, all rework is known and implemented completely in the first instance, corresponding exactly with the work really done. Figure 6.26 shows the potential process improvement due to increasing IA quality for the third example change set, which has a small shift in the known rework curve compared with the other examples (Figure 6.19).

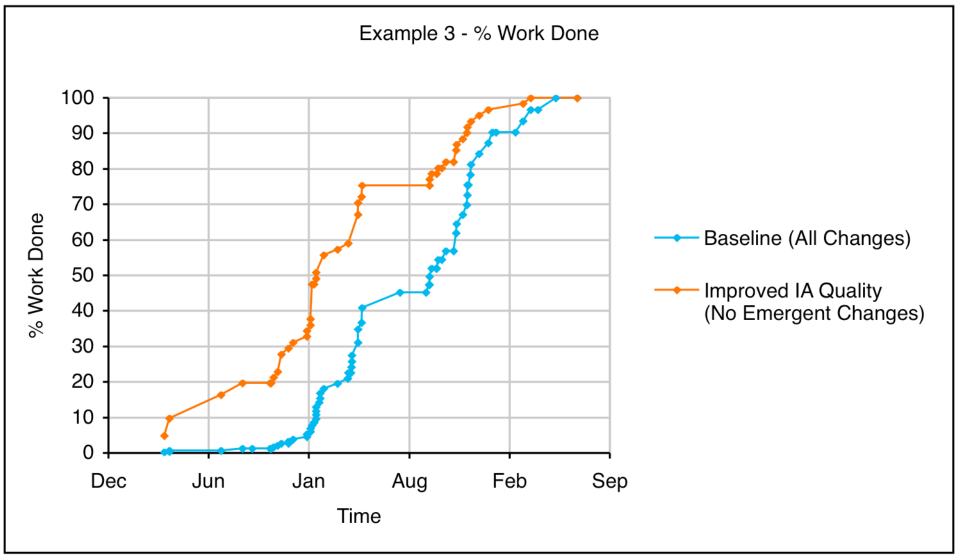


Figure 6.26: Potential process improvement for example 3 (from Figure 6.19)

As suggested by this data interpretation and shown by the gap between the curves in Figure 6.26, improving IA quality can have a significant effect on the work really done and more of the design can be completed earlier. In theory, this improvement can reduce the necessary project resources and cut development costs. However, the analysis of the IA improvement strategies investigated through simulation (Section 7.4) does not include this effect due to the perceived and actual work really done. The simulation focuses on the relative shifts of known rework curves due to varying the adapted rework cycle parameters to depict different IA improvement and change packaging strategies, and the results are interpreted in such relative terms.

Furthermore, in this data analysis, the estimated work done and known rework curves are restricted in the time-axis since only knock-on changes are modelled to occur earlier. Additional changes could be identified earlier since they may be spawned (due to the quality of work) by other modifications that also occur earlier (due to decreasing the rework discovery time delay or increasing the IA quality). Simulation can capture this effect by generating changes as necessary over time. As such, the primary improvement investigated in the simulation is in shifts of the known rework curve due to finding changes earlier.

6.5 SUMMARY

Literature in system dynamics and software process dynamics provide the basis for the examination of the effects of IA improvement on design processes through a structured method and references to similar modelling and simulation research. Specifically, the rework cycle model, developed by Cooper (1993b), is adapted to model IA and change packaging in order to address the fourth (Figure 5.12) and refined research questions (Figure 5.13 and Figure 5.14). Even though this model (Figure 6.6) provides an appropriate, high-level framework to address these research questions and compare a range of process improvement strategies, the adapted rework cycle model has limitations, as outlined in Table 6.1, which are, in turn, accounted for in the development of the reference modes (Section 6.4), simulation (Section 7.1 and Section 7.5), and process improvement heuristics (Section 8.2).

This chapter concludes by developing reference modes to describe the intended simulation behaviour, completing the model formulation in Step 1 of the modelling and simulation method applied (Figure 6.27). The reference modes are based on quantitative data from a change database obtained from the aerospace company. In turn, satisfying these reference modes in the simulation effectively allows for the comparison of the consequences of the different IA improvement and change packaging strategies derived for the aerospace company's design process.

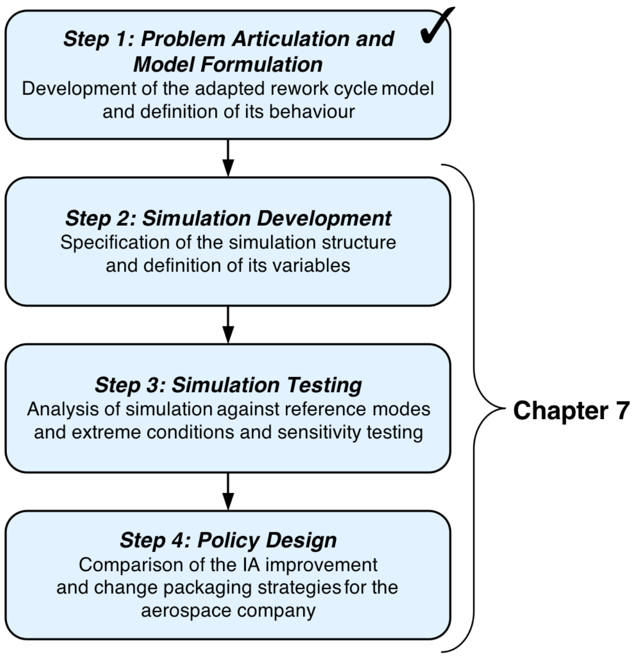

Step 1: Problem Articulation and Model Formulation
Development of the adapted rework cycle model and definition of its behaviour
✓
Step 2: Simulation Development
Specification of the simulation structure and definition of its variables
Step 3: Simulation Testing
Analysis of simulation against reference modes and extreme conditions and sensitivity testing
Step 4: Policy Design
Comparison of the IA improvement and change packaging strategies for the aerospace company
Chapter 7

Figure 6.27: Progress in the application of the modelling and simulation method

7 :: SIMULATING IMPACT ANALYSIS IMPROVEMENT

Simulating the adapted rework cycle model (Figure 6.6) allows for the comparison of IA improvement and change packaging strategies. Scenarios, involving perturbing the model variables to different extents, can demonstrate the outcome of and trade-offs in modifying IA and change packaging practice (Section 6.1.1). Variations in these parameters are investigated through Monte Carlo simulation similar to the implementation of other discrete software process dynamics models (Raffo and Kellner 2000).

In order to perform this examination of improvement strategies or policies (Section 7.4 and Section 7.5), the simulation structure of the adapted rework cycle is first defined (Section 7.1 and Section 7.2), and the simulation is then tested to assure its results are representative of the aerospace company's design process (Section 7.3). This fulfils the remaining steps in the modelling and simulation method applied, summarised by Step 2, Step 3, and Step 4 in Figure 6.27.

7.1 SIMULATION OF THE ADAPTED REWORK CYCLE

As discussed in Section 6.3.4, the simulation structure of the adapted rework cycle (Step 2 in the modelling and simulation method applied, Figure 6.27) only represents the implementation of rework, as opposed to also the stock of work to be done. Similarly, the people and productivity parameters are not distinctly modelled, and the simulation accordingly makes an assumption on the rate of work done. Figure 7.1 highlights the parameters of the adapted rework cycle model represented in the simulation.

7.1.1 SIMULATION DESIGN

The simulation essentially captures rework through modelling the flow of changes through the adapted rework cycle. Not only does this structure allow for the straightforward comparison of the simulation results with the reference modes

(Section 6.4.3), but also interpreting rework as changes fits with the concept of discretely[45] applying IA to determine the rework discovery flow. In turn, the simulation creates an initial stock of changes to represent unknown rework and models the discovery, execution, and completion of these modifications (Figure 7.2). Depending on the quality of work parameter, these changes can also generate additional modifications, depicting the fraction of rework really completed after implementation. This creation of changes corresponds with errors that occur within rework performed.

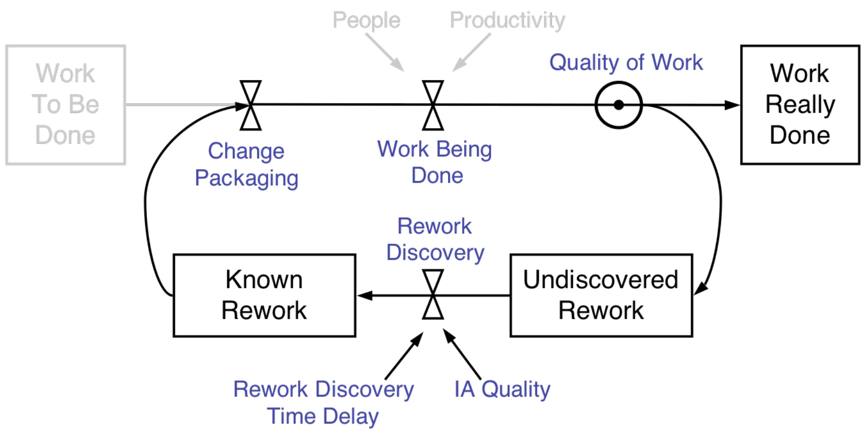


Figure 7.1: The adapted rework cycle simulation parameters

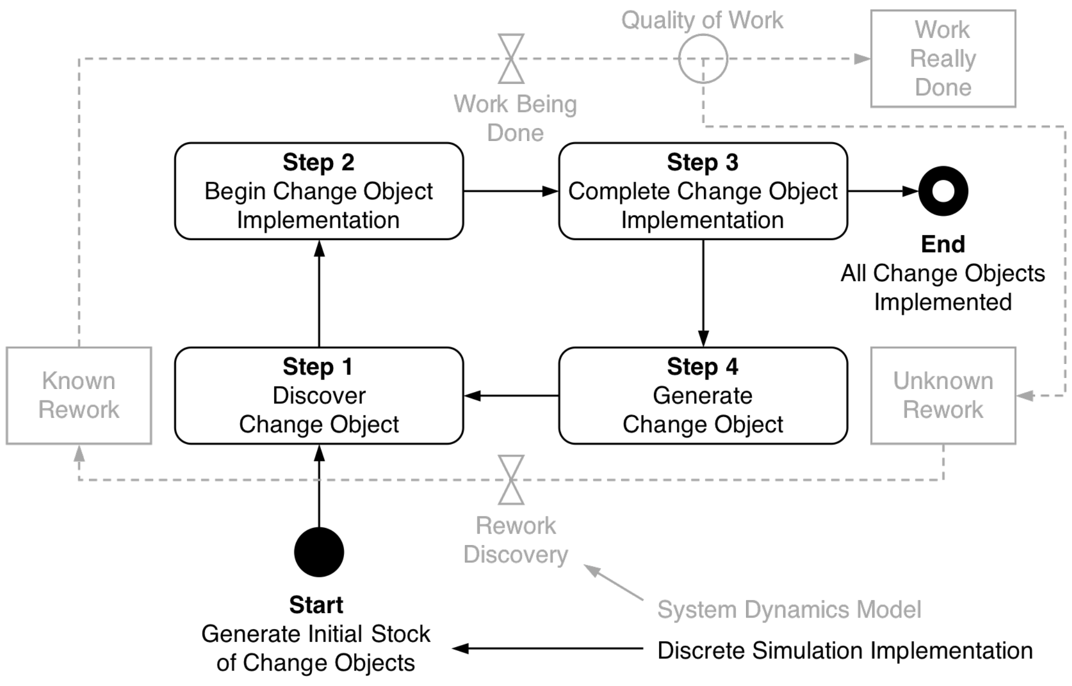

Quality of Work
Work Really Done
Work Being Done
Step 2 Begin Change Object Implementation
Step 3 Complete Change Object Implementation
End All Change Objects Implemented
Known Rework
Step 1 Discover Change Object
Step 4 Generate Change Object
Unknown Rework
Rework Discovery
Start Generate Initial Stock of Change Objects
System Dynamics Model
Discrete Simulation Implementation

Figure 7.2: Steps in the adapted rework cycle simulation

[45] Section 6.1.2 and Section 7.1.2 discuss applying the rework cycle *discretely* due to the simulation of change packaging. A continuous simulation would not allow for the analysis of change packaging.

In the simulation, modifications are modelled through *change objects*, which have associated attributes (Table 7.1). These attributes are updated at each step of the adapted rework cycle simulation (Figure 7.2). The update of each of the change object attributes depends on the input from other variables represented in the simulation, also described in Table 7.1. The remainder of this section details the simulation structure in terms of these change object attributes.

Table 7.1: Attributes of change objects and associated input variables

Attribute	Description	Associated Input Variable(s)
Identification Number	Integer identification numbers are assigned sequentially, starting from 1. The initial stock of change objects are first assigned ID numbers and subsequent change objects generated are allocated numbers successively.	---
Functional Area	Functional areas, depicted by an integer identification number, are randomly assigned to change objects based on the inputted distribution of functional area volatility.	• Number of functional areas • Distribution of functional area volatility
Change Type	Change objects are randomly generated as either independent or change sets, depending on the percentage split between these two categories.	• Ratio of independent modifications to change sets
Functional Impact Value(s)	Independent change objects have a single functional impact value, while change sets have multiple functional impact values. Change objects are randomly assigned functional impact value(s) according to the percentage of changes in each functional impact category and, for change sets, the distribution of the number of changes within change sets.	• Distribution of the number of changes within change sets • Categories of functional impact and their associated numerical values • Percentage of changes in each functional impact value category
Generation Time	Change objects are stamped with a generation time to indicate when they are created. The generation times of the initial stock of change objects are identical, while change objects created through rework are given generation times equal to the completion time of the spawning change objects.	• Quality of work mean and standard deviation
Discovery Time(s)	A discovery time indicates the simulation clock time at which functional impact value(s) within a change object can be found, depending on the IA quality. Independent and initiating modifications in change sets are designated discovery times, and then a number of additional discovery times to identify knock-on changes in change sets are allocated.	• Distribution of independent change discovery over time • Distribution of initiating change discovery over time • IA quality mean and standard deviation • Distribution of knock-on change discovery over time
Implementation Time(s)	An implementation time is associated with a functional impact value, indicating when it is found, and is set as a delay from a discovery time.	• Rework discovery time delay mean and standard deviation
Completion Time(s)	A completion time is associated with a functional impact value and is set as a delay from the related implementation time.	• Implementation delay mean and standard deviation

When change objects are created for the initial stock of modifications or generated from work being done (prior to rework discovery in Step 1 of Figure 7.3), they are given unique identification numbers and are randomly assigned to functional areas. The functional area assigned is used for change packaging (Section 6.3.3). The change objects created are also defined as independent modifications (*i.e.* changes sets with a single initiating change and no knock-on rework) or change sets (*i.e.* including initiating and knock-on modifications). As such, the functional impact values of change sets are generated differently from independent modifications. Change objects designated as change sets are given multiple functional impact values to represent multiple changes, while change objects representing independent modifications have a single functional impact value. These functional impact values are placed into the same three categories (*i.e.* no impact, minor impact, major impact) and have the same numerical values as from the data analysis (Table 6.4). When the initial stock of change objects is created or subsequent change objects are generated, they are also stamped with a generation time. Figure 7.3 displays the timing of these attribute definitions, and Table 7.1 details the input parameter values used in their definition.

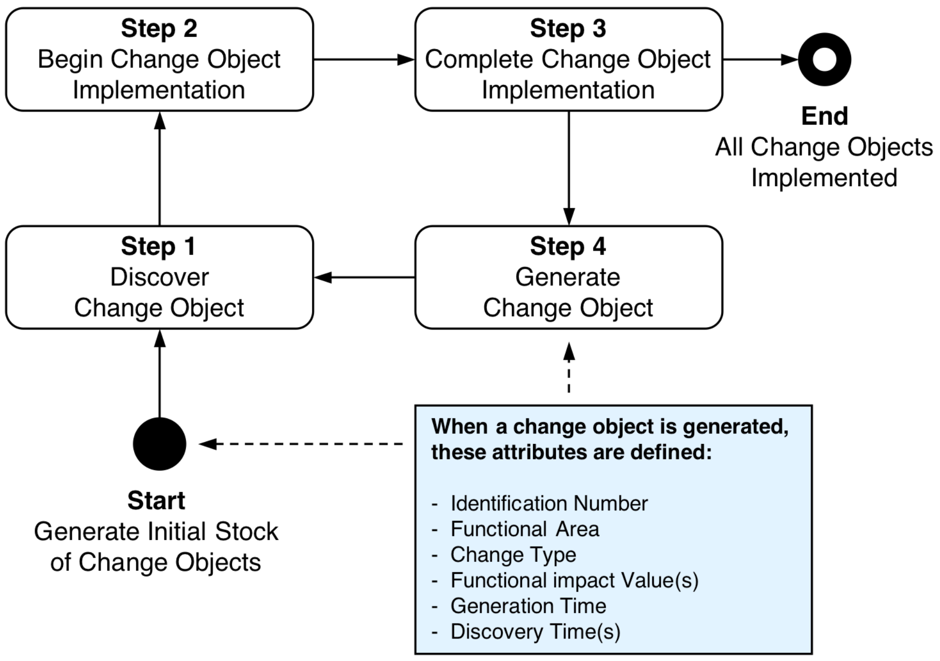

Step 2
Begin Change Object Implementation
Step 3
Complete Change Object Implementation
End
All Change Objects Implemented
Step 1
Discover Change Object
Step 4
Generate Change Object
Start
Generate Initial Stock of Change Objects
When a change object is generated, these attributes are defined:
- Identification Number
- Functional Area
- Change Type
- Functional impact Value(s)
- Generation Time
- Discovery Time(s)

Figure 7.3: Change attributes defined prior to rework discovery

In addition, discovery time(s) are created once a change object has been generated (prior to rework discovery in Step 1 of Figure 7.3). A discovery time represents the time at which IA is performed and changes can enter the stock of known rework, and, in turn, functional impact value(s) within change objects can be discovered when the simulation time clock equals a discovery time (Step 1 in Figure 7.3). An independent modification has a single discovery time to find its single functional impact value, while a change set has multiple discovery times[46] to detect fractions of the functional impact values within the change object.

For the initial stock of change objects, discovery times for independent changes and the first modifications identified in change sets are randomly assigned according to their distributions across the design process. Discovery times of knock-on changes[47] are subsequently designated for change sets after this first discovery time. However, for change objects spawned from work being done (the generation of changes in Step 4 of Figure 7.3), the discovery times for independent and initiating changes are set equal to their generation times, and knock-on changes in changes sets are given additional discovery times according to their distribution across the design process. Setting the generation and first discovery times equal allows the *rework discovery time delay* parameter (Figure 7.1) to completely characterise the delay between generating functional impact values and their discovery.

When the simulation time clock equals a discovery time within a change object, then functional impact value(s) within the change object can be discovered and enter the stock of known rework. In turn, the implementation times for a number of functional impact values can be set, as depicted in Figure 7.4, since it is assumed that changes are immediately scheduled for implementation. For a change object of an independent change, the implementation time is set as a delay from its discovery time, and this delay corresponds with the *rework discovery time delay* model parameter (Figure 7.1). Similarly, if a change object represents a change set, then at each discovery time a fraction of the functional impact values are assigned the same implementation time, according to the rework discovery time delay simulation parameter. The fraction of changes discovered depends on the *IA quality* model parameter (Figure 7.1). In turn, change packaging modifies this basic model to incorporate a planning delay between the change discovery and implementation times, and Section 7.5 details the modification of the simulation design for change packaging.

[46] The first discovery time in a change set finds an initiating change as well as a portion of knock-on changes and leaves a remainder of knock-on changes to identify at later discovery times, according to the definition of IA quality (Section 6.3.2).

[47] The number of discovery times created for change sets depends on the IA quality and number of functional impact values in the change set. Given that IA quality indicates the fraction of changes identified after each application of IA, the number of discovery times generated is calculated such that all changes can just be found. Creating additional, unnecessary discovery times could allow functional impact values to be found earlier than intended.

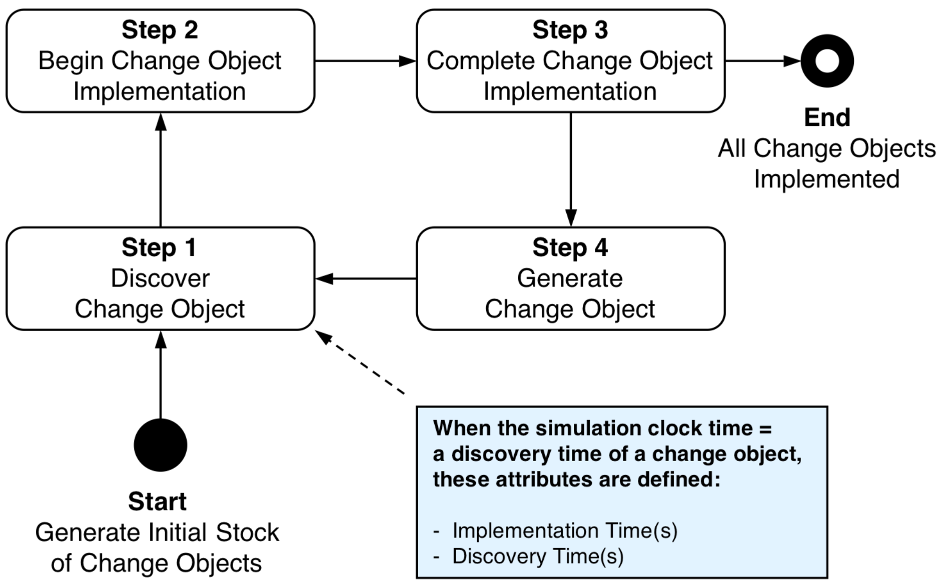

Step 2
Begin Change Object Implementation
Step 3
Complete Change Object Implementation
End
All Change Objects Implemented
Step 1
Discover Change Object
Step 4
Generate Change Object
Start
Generate Initial Stock of Change Objects
When the simulation clock time = a discovery time of a change object, these attributes are defined:
- Implementation Time(s)
- Discovery Time(s)

Figure 7.4: Change attributes defined at rework discovery

IA quality is modelled as the probability of identifying each change in a set during an application of IA. An IA quality variable value of 0.5 indicates that each functional impact value has a probability of 0.5 of being found at each discovery time, and approximately half of the functional impact values remaining in the set are discovered at such a time, as implemented using a random number generator. As such, no specific functional impact value is assigned as the initiating change, and not all functional impact values may be discovered within the allocated number of discovery times. In turn, if there are remaining functional impact values to be discovered, additional discovery times are created according to their distribution across the design process from the last discovery time.

Hence, in this simulation structure, a smaller rework discovery time delay uniformly decreases all implementation times by a fixed amount, while improving IA quality does not change any of these times and only shifts the functional impact found between these points in time, corresponding with the defined behaviour of the adapted rework cycle (Section 6.4.1). Consequently, in order to compare simulation results using different IA quality values, the same number of discovery times is generated for cases being evaluated (*i.e.* a baseline case with no IA improvement and another with IA improvement).

Once an implementation time has been set for a functional impact value, the corresponding completion time can be set, as shown in Figure 7.5. As previously mentioned in Section 6.3.4, the simulation assumes that the people and productivity levels can readily implement the changes found. In turn, the completion time is modelled as a variable delay from the implementation time of each functional impact value, effectively corresponding to the rate of *work being done* parameter (Figure 7.1). Notably, as modelled, this delay is independent from the amount of functional impact identified. Given that the functional impact values only intend to depict changes that are marginally different in size, other factors (*i.e.* scheduling) have more of an influence on this implementation delay.

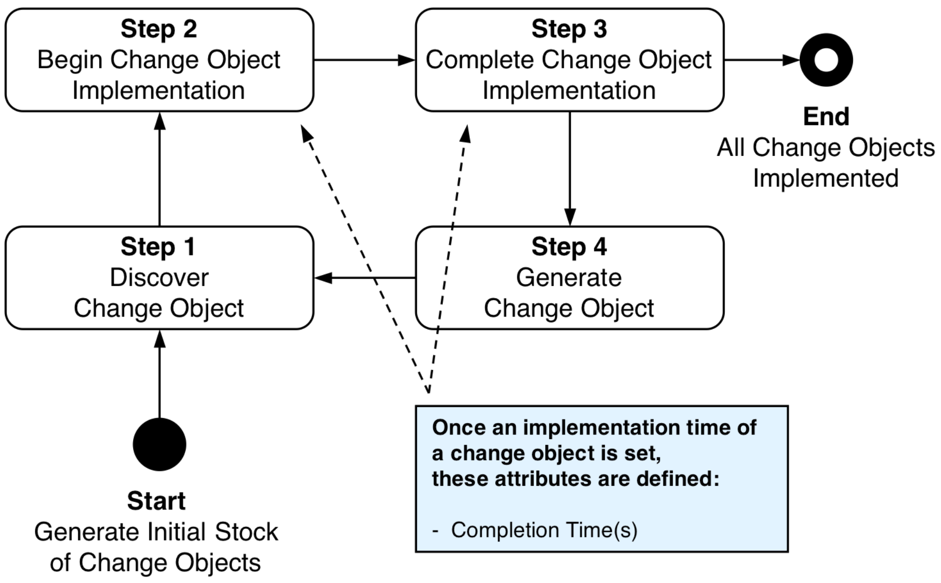

Step 2
Begin Change Object Implementation
Step 3
Complete Change Object Implementation
End
All Change Objects Implemented
Step 1
Discover Change Object
Step 4
Generate Change Object
Start
Generate Initial Stock of Change Objects
Once an implementation time of a change object is set, these attributes are defined:
- Completion Time(s)

Figure 7.5: Change attributes defined at change completion

When a functional impact value is assigned a completion time, it may or may not spawn additional change objects based on the quality of work parameter (Step 4 in Figure 7.5). The simulation treats quality of work as a probability for generating another change object, similar to the IA quality variable. For each functional impact value and given this probability, another change object can be generated with an identification number, functional area, change type, generation time, functional impact(s), and discovery time(s) (Figure 7.3). This modelling of the quality of work directs rework proportionately through the number of change objects spawned.

As such, the modelling of change packaging affects this generation of additional change objects. Specifically, if functional impact values of independent changes or change sets are packaged together (*i.e.* are implemented concurrently), multiple changes may spawn fewer change objects than determined by the quality of work parameter (Section 6.3.3). Section 7.5 discusses the modelling of the change packaging policies in more detail.

The steps outlined for the adapted rework cycle (Figure 7.2) are simulated discretely (Section 6.1.2). Updates to the attributes of each change object occur at distinct time intervals in that the simulation time clock jumps between the discovery times and instantaneously assigns generation, implementation, and completion times. In turn, like many other discrete simulations, an end condition is set in order to stop the continual generation and subsequent implementation of change objects. The adapted rework cycle simulation stops generating changes at a specified time and finishes by giving outstanding functional impact values implementation times according to the rework discovery time delay from this specified time and another delay from a uniform distribution to a specified simulation end time. These changes are then completed according to the implementation delay.

Upon completion of the simulation, the attributes of the change objects generated can be analysed. Specifically, the known rework curve (Figure 6.18), describing the behaviour of the adapted rework cycle (Section 6.4), can be plotted using the functional impact values in all the change objects and their associated completion times. Two simulation cases implemented with different parameter values (Table 7.1) can be compared by constructing their respective known rework curves and determining the time difference (Δt) between them, shown in Figure 7.6.

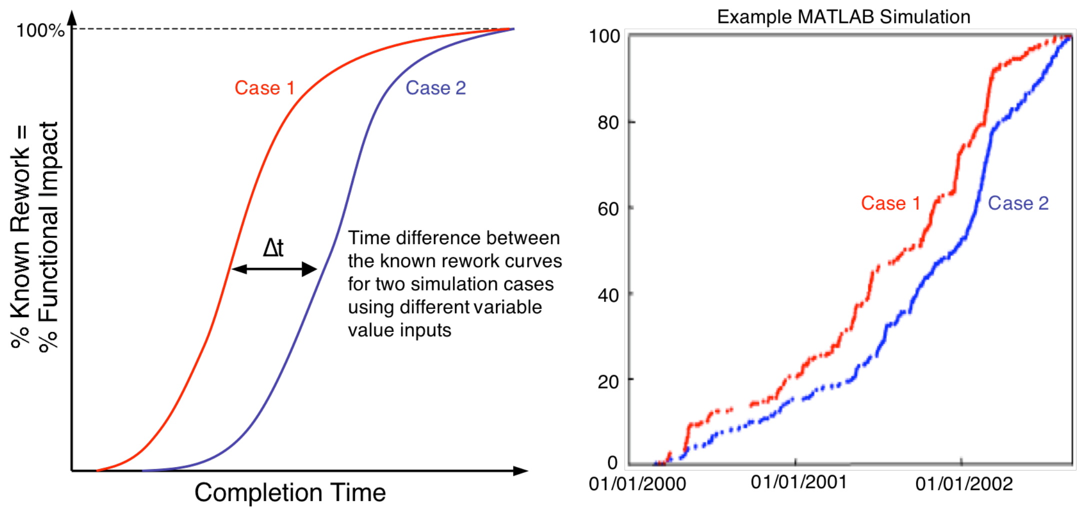


Figure 7.6: Representation of known rework curve (from Figure 6.18) for simulation result analysis

The simulation results for the IA improvement (Section 7.4.2) and change packaging (Section 7.5.2) strategies report the mean time difference between such known rework curves of simulation cases using different variable values. In turn, these time differences suggest the overall process improvement in that more of the design is completed earlier. As such, fewer resources may be necessary, and downstream processes (*e.g.* product testing) may be commenced earlier, reducing the total development process duration. As described by Raffo and Kellner (1999a), performing analyses using such measured deltas in Monte Carlo simulations allows trade-offs to be illuminated. However, this measured time difference and, thus, the heuristics derived from the simulation results (Section 8.2) assumes that design processes only end once all rework is complete. As suggested by the evaluation of these heuristics (Section 8.3), in some situations, design work also may be halted and a product released when design errors are known to exist, but remain unfixed. In turn, this assumption on the analysis of the simulation results is addressed in the heuristics evaluation.

7.1.2 MATLAB IMPLEMENTATION

Although system dynamics software tools, such as Vensim[48], provide automated support to simulate models, this software generally does not provide for implementing discrete simulations. These software tools specifically do not easily accommodate the flexibility required to simulate change packaging. As such, MATLAB[49] was used to implement the adapted rework cycle simulation, given its availability and built-in statistics and graphing capabilities.

MATLAB captures simulation code within m-files, and partitioning code between m-files can allow for the logical division of the simulation design. In order to implement the adapted rework cycle in MATLAB, an m-file, called "wrapper", initialises two sets of simulation variables (Table 7.1 and Section 7.2), depicting two cases to simulate. Another m-file, called "simulate-and-analyse", uses the variable values of these cases and calls the "simulate" and "analyse" m-files in sequence. The "simulate" m-file implements the simulation for each case (Section 7.1.1), and the "analyse" m-file calculates the time difference between the known rework curves of these cases (Figure 7.6). The "simulate-and-analyse" m-file repeats the simulation and analysis of the cases a number of times and collects relevant statistics, such as the mean time difference over all of the simulations performed. Figure 7.7 illustrates this implementation of the adapted rework cycle simulation.

[48] Information about Vensim® can be found at http://www.vensim.com.
[49] Information about MATLAB® can be found at http://www.mathworks.com.

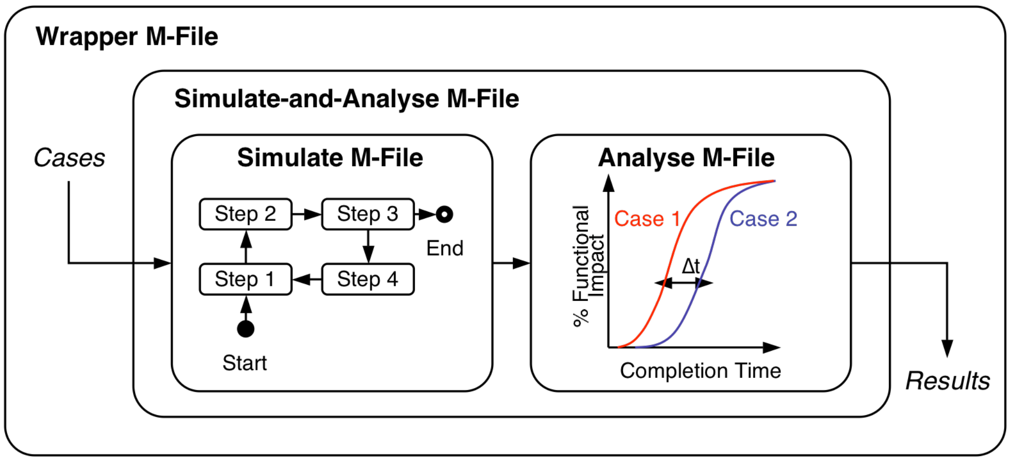


Figure 7.7: MATLAB implementation of the adapted rework cycle simulation

The results of the simulation testing and of the IA improvement and change packaging strategies are based on 200 simulation runs (*i.e.* calling the "simulate-and-analyse" m-file 200 times), unless otherwise noted. This number of trials produces an acceptable standard error of 2.6 days on average in the time difference between cases (Figure 7.6). The standard deviation of the time difference between cases is primarily influenced by the number of change objects generated and does not vary significantly across the cases simulated.

7.2 SIMULATION VARIABLE VALUES AND INITIAL CONDITIONS

In order to run the simulation design depicted in Figure 7.7, the variable values used in the simulated cases, which are summarised in Table 7.1, must be set in the "wrapper" m-file. Some of these variables are held constant throughout the cases simulated (Table 7.2); other values are used to generate the initial set of changes to begin both simulation cases and specify the end condition to end these simulations (Table 7.3); and, finally, the another set of values are the key parameters used to compare and evaluate different IA improvement strategies (Table 7.4). The parameters used to analyse the trade-offs of change packaging strategies are discussed with respect to the modification of simulation design in Section 7.5.

The estimation of the values for the simulation variables is taken from the results from analysing the change database for key characteristics of the 3 example design area changes also used to develop the reference modes (Table 6.5) and from information provided by interviewees at the aerospace company. In particular, the IA quality and quality of work variable values are drawn specifically from estimates by interviewees at the aerospace company (I-2, I-12, I-15, I-16), and the remaining variable values are derived from the examples in the change database.

The following tables indicate the baseline values for these variables used in the simulation testing (Section 7.3) and from which the IA improvement simulation results stem (Section 7.4). Multiple baseline values are specified for the parameters used to compare strategies (Table 7.4) since they cover the range of the interviewees' estimations and are used in the simulation testing. Specifically, the permutations of IA quality and quality of work improvement are run for each test and the results are averaged. The variables in Table 7.4 are perturbed to analyse the IA improvement strategies, and the values used in this analysis are discussed with respect to the simulation scenarios run (Section 7.4).

Table 7.2: Constant variable values

Constant Variable (from Table 7.1)	Value (from Table 6.5)
Number of functional areas	12
Distribution of functional area volatility	2/3 of changes within 4 functional areas, 1/3 of changes within remaining functional areas
Ratio of independent modifications to change sets	1 : 4
Distribution of the number of changes within change sets	Uniform distribution of 2 and 8 changes per set
Categories of functional impact and their associated values	No impact – 0 Minor impact – 5 Major impact – 10
Percentage of changes in each functional impact value category	No impact – 33% Minor impact – 55% Major impact – 12%
Distribution of knock-on changes over time	Uniform distribution from initiating change time to the time to end all change generation
Implementation delay mean and standard deviation	Mean – 31 days Standard deviation – 3.1 days

Table 7.3: Initial and end condition variable values

Initial or End Condition Variable (from Table 7.1 and Section 7.1.1)	Value (from Table 6.5)
Simulation clock start time	1 January 2000
Simulation clock time to end all change generation	1 January 2002
Simulation clock time to end all change implementation	31 July 2002
Number of change objects initially generated	50
Distribution of independent change discovery over time	Uniform distribution from simulation start time to the time to end all change generation
Distribution of initiating changes discovery over time	Uniform distribution from simulation start time to the time to end all change generation

Table 7.4: Variable values used to compare strategies

Comparison Variable (from Table 7.1)	Value Range (Elicited from Interviewees)
Rework discovery time delay mean and standard deviation	Normal distribution with Mean – 93 days Standard deviation – 3.1 days
IA quality mean and standard deviation	Normal distribution with Mean – 0.4, 0.5, 0.6 Standard deviation – 0.05
Quality of work mean and standard deviation	Normal distribution with Mean – 0.4, 0.5, 0.6 Standard deviation – 0.05

As such, this description of the simulation and its implementation (Section 7.1) as well as the variable values used in the simulation (Section 7.2) completes Step 2 of the modelling and simulation method applied (Figure 7.8).

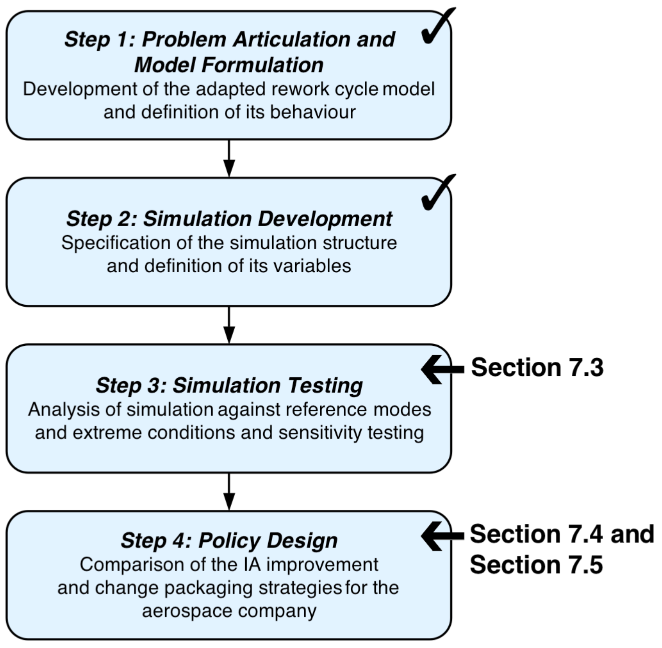

Step 1: Problem Articulation and Model Formulation
Development of the adapted rework cycle model and definition of its behaviour
✓
Step 2: Simulation Development
Specification of the simulation structure and definition of its variables
✓
Step 3: Simulation Testing
Analysis of simulation against reference modes and extreme conditions and sensitivity testing
Section 7.3
Step 4: Policy Design
Comparison of the IA improvement and change packaging strategies for the aerospace company
Section 7.4 and Section 7.5

Figure 7.8: Progress in the application of modelling and simulation method

7.3 TESTING THE ADAPTED REWORK CYCLE SIMULATION

To perform the simulation testing in Step 3 of Figure 7.8, the adapted rework cycle simulation should be tested (1) against reference modes, (2) under extreme conditions, and (3) by analysing the sensitivity of the results to variable values. In turn, the testing of the simulation indicates that the chosen variable values produce a simulated design process representative of the aerospace company's design process and suggests the effects of variable estimation. Although the simulation focuses on trade-offs between parameters, as opposed to concretely predicting decreases in process duration as discussed in Section 6.3.4, these testing outcomes verify the interpretation of the simulation results for the aerospace company's design process.

7.3.1 REFERENCE MODES

The simulation produces results within the constraints of the reference modes derived from the aerospace company's design process (Table 7.5) through the variable values in Table 7.2 and Table 7.3 and given any combination of IA quality and quality of work parameters in the baseline ranges specified in Table 7.4. As such, the interpretation of the results from the IA improvement strategies is indicative of the potential process improvement for this firm.

Table 7.5: Summary of reference modes (from Section 6.4.3)

Reference Mode	States:
1	The simulation should generate changes with a ratio of independent modifications to change sets and percentage of knock-on changes representative of the data analysis (Table 6.5).
2	The simulation should generate initiating changes throughout the design process.
3	The simulation should generate knock-on changes for each change set that uniformly span the remainder of the design process.
4	The simulation should generate changes at a constant rate throughout the design process, but instances of increases in number of changes are allowed.

For the first reference mode (Table 7.5), the value of the ratio of change sets to independent modifications (Table 7.2) is approximated from the data analysis of the 3 example design area changes (Table 6.5). As such, if the simulation also produces acceptable percentages of knock-on changes versus all changes generated (*i.e.* 25 – 42% as also estimated by the data analysis in Table 6.5), then the first reference mode is met.

As interpreted in the data analysis, knock-on changes are classified as all the changes reported after an initiating modification. Notably, the percentage of knock-on changes derived is only an estimate since the database does not report all changes implemented (Appendix E). Specifically, initiating modifications are assumed to include the identification of other simultaneous knock-on changes, which are not explicitly reported. Given that these undocumented first changes can approximately double the average number of knock-on changes classified in the data analysis with the aerospace company's reported IA quality values, excluding these modifications significantly decreases the knock-on percentage estimated. In turn, to correspond with the data analysis interpretation of knock-on changes, the percentage of knock-on changes calculated from the simulation only includes functional impact values found after the first implemented modifications in a change set[50]. Also, since the small number of knock-on modifications from the data analysis can be found individually with the IA quality value estimates, the percentage of knock-on changes calculated from the simulation accounts for functional impact values singularly that are implemented after the first implemented modifications in a change set.

Through this interpretation of the knock-on changes defined by the data set and variable values used, the first reference mode is met in that the knock-on changes calculated for the simulation account for between 25 – 42% of all changes generated with the IA quality values estimated. More functional impact values may be discovered initially and fewer subsequent knock-on values may exist due to increasing IA quality. Table 7.6 describes the variation of this knock-on change percentage accordingly based on averaging the simulation results in the baseline range of quality of work values stipulated (Table 7.4).

Table 7.6: Simulation results for reference mode 1

IA Quality	Percentage of Knock-On Changes	
	Mean	Standard Deviation
0.4	38%	4%
0.5	32%	4%
0.6	27%	4%

The second and third reference modes (Table 7.5) are met based on the modelled distributions over time for initiating and knock-on changes. Initiating changes are created throughout the design process given the initial conditions (Table 7.3) and are also spawned in Step 4 of the simulation due to rework (Figure 7.2) until the simulation clock time equals the time specified at which all change generation stops. Similarly, knock-on modifications for each change set are defined to uniformly span from an initiating change to the end time when change generation stops (Table 7.2).

Finally, the simulation meets the fourth reference mode (Table 7.5) based on the distribution of changes generated across the simulated design process time frame. Given the modelling of the second and third reference mode, an increase in the number of changes can occur towards in the end of the simulation time for some

[50] This portion of the knock-on changes simulated (*i.e.* knock-on changes not implemented with the initiating change) can estimate the small average number of knock-on changes classified in the data analysis (*i.e.* approximately 3 modifications from Table 6.5). As such, the *number of changes within change sets* variable (Table 7.2) is approximated through the number of modifications found after the first implemented modifications and the IA quality values estimated. Consequently, the simulation represents the amount of rework within change sets indicated by the database analysis.

combinations of low IA quality and quality of work values, which leave many changes to be implemented at the end of the design process. However, observing the distribution of the number changes completed over 50 simulations with permutations of IA quality and quality of work values suggests that the simulation approximates changes occurring at more or less constant rate with peaks in the number of changes typically less than those discussed with respect of the fourth reference mode (Figure 6.23). Figure 7.9 displays an instance of this distribution with IA quality and quality of work values of 0.4, which leaves changes to be implemented late.

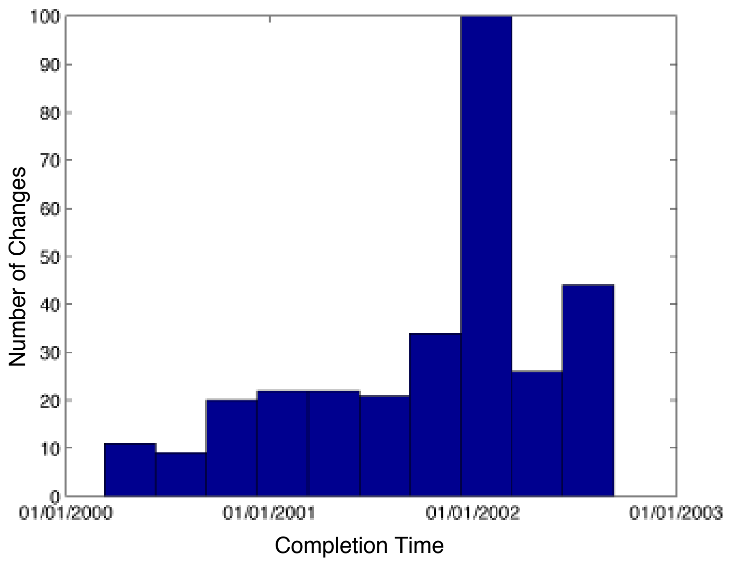


Figure 7.9: Example distribution of changes from simulation

7.3.2 EXTREME CONDITIONS

The simulation can also be tested against extreme variable values to establish the useful domain of the model. Such boundary conditions occur when the model behaviour cannot be relied upon. Specifically, the rework discovery time delay, IA quality, and quality of work values (Table 7.4) as well as the implementation delay (Table 7.2) and number of initial change objects generated (Table 7.3) are chosen for investigation since these variables directly affect the flow of changes around the rework feedback loop. Table 7.7 displays any limits for these variables given the values of the other parameters specified (Section 7.2) and describes the cause for the boundaries. In turn, the variable values used for the IA improvement strategies investigated (Section 7.4) are within these limits.

Table 7.7: Extreme values for simulation variables (from Section 7.2)

Variable	Lower Bound	Upper Bound	Description
Rework discovery time delay mean	---	---	Given that the rework discovery time delay uniformly shifts the known rework curve, this value is not limited by the intended simulation behaviour.
IA quality mean	0.3	---	IA quality values less than 0.3 can leave many changes to be implemented between the time to end all change generation and the simulation end time. In turn, the modifications within this interval systematically causes an uncharacteristic change in slope of the known rework curve, which may affect the time difference estimation between cases (Figure 7.6), and the distribution of all changes increases towards the end of the design process, interfering with the fourth reference mode (Table 7.5). Note that this extreme condition is estimated based on the observation of 50 simulation runs of incremental potential lower bound values. Large IA quality values do have these effects since most changes are completed within the specified time interval.
Quality of work mean	0.2	---	Similar to the IA quality lower bound, small quality of work values leave many changes to be implemented after the change generation end time, producing a disjunct in the known rework curve concavity and interfering with the fourth reference mode. Note that this extreme condition is also estimated based on the observation of 50 simulation runs of incremental potential lower bound values.
Implementation delay mean	---	---	Similar to the rework discovery time delay, the implementation time delay uniformly shifts the rework curve in time and does not change its characteristics.
Number of change objects initially generated	50	100	Using a small number of initial change objects generates a sparse number of additional change objects throughout the design process. As such, the interpolated time difference between cases can highly vary. Increasing the initial number of changes generated systematically decreases the standard deviation of this time difference, but also increases the simulation run time. Fifty initial changes produce simulation results demonstrating statistically significant improvements, even though a lower number of initial change objects generated may also be sufficient. Using more than 100 initial change objects begins to interfere with the fourth reference mode (Table 7.5) in that many more changes are implemented towards the end of the design process, making the change distribution more noticeably exponential. Note that this extreme condition is also estimated based on the observation of 50 simulation runs of incremental potential upper bound values.

7.3.3 SENSITIVITY ANALYSIS

Finally, a sensitivity analysis on particular simulation parameters can indicate the influence of inaccurate variable value estimation on the interpretation of the simulation results. The simulation variables that can affect the mean time difference between simulation cases (Figure 7.6), besides the key variables investigated in the strategies investigated (Table 7.4), are the *ratio of independent modifications to change sets* and *distribution of the number of changes within change sets* as well as the *values associated with the functional impact categories, the percentage of changes within each* *functional impact category*, and *implementation delay* (Table 7.2). As such, a sensitivity analysis increases and decreases each of these variables individually by a percentage[51] from their baseline values and then determines the average time difference caused, as illustrated by the time difference between cases in Figure 7.6. The IA quality and quality of work values in Table 7.4 are also varied in this analysis, and the analysis averages the simulation results across these varying IA quality and quality of work values.

The mean time differences calculated in this sensitivity analysis (Table 7.8) suggest that the *ratio of independent modifications to change sets* and *distribution of the number of changes within change sets* variables primarily affect the known rework curves constructed and can influence the prediction of the design process duration. However, given that the estimation of these variables is held constant in the simulations run to compare IA improvement strategies (Section 7.4), changes in the mean time difference do not influence the interpretation of trade-offs between these simulation scenarios run. The other variables analysed are relatively insensitive to perturbations and change the time difference calculated at most by 2%.

Table 7.8: Sensitivity of simulation variables (from Table 7.2)

Variable	Percent Change in Variable	Percent Change in Mean Time Difference
Ratio of independent modifications to change sets	+ 20%	+ 7%
	- 20%	- 7%
Distribution of the number of changes within change sets	+ 25%	+ 7%
	- 25%	- 6%

Notably, the percentage changes in the *ratio of independent modifications to change sets* and *distribution of the number of changes within change sets* variables in this sensitivity analysis cause the calculated percentage of knock-on changes (Section 7.3.1) to slightly exceed the stipulated limits (*i.e.* 25 – 42%) of the first reference mode, as shown in Table 7.9 and Table 7.10. Nevertheless, given the estimation of these percentages for this reference mode, the variation of these values can still construct a representative baseline design process for the aerospace company. Moreover, these perturbations still allow the simulation to be representative of the fourth reference mode and, in turn, do not significantly affect the applicability of the simulation results to the aerospace company's design process.

[51] All values are varied by 20%, except for the distribution of the number of changes within change sets. This distribution is varied by 25% to allow the associated ratio to make physical sense in that fractional changes cannot occur in practice.

Table 7.9: Sensitivity of reference mode 1 to ±20% change in the ratio of independent modifications to changes sets

IA Quality	Percentage of Knock-On Changes	
	Mean (+20% / -20%)	Standard Deviation (+20% / -20%)
0.4	41% / 33%	4% / 5%
0.5	36% / 28%	4% / 5%
0.6	30% / 23%	4% / 4%

Table 7.10: Sensitivity of reference mode 1 to ±25% change in the distribution of the number of changes within changes sets

IA Quality	Percentage of Knock-On Changes	
	Mean (+25% / -25%)	Standard Deviation (+25% / -25%)
0.4	40% / 34%	4% / 4%
0.5	35% / 28%	4% / 4%
0.6	29% / 24%	4% / 4%

Considering these simulation results further suggests that process improvement is more sensitive to increasing IA than reducing the number of changes generated, which occurs through improving the quality of work. Specifically, the normalised sensitivity coefficient (Table 7.11) for reducing the number of knock-on changes is about 0.3 since a 6 – 7% improvement effect can occur from a 20 – 25% decrease in the number of knock-on modifications (from Table 7.8). In turn, this coefficient is approximately 0.6 for improving IA quality because a 10% increase in IA quality approximately accounts for a similar 20% decrease in the number of knock-on modifications (from Table 7.9 and Table 7.10). In other words, improving IA quality can be more beneficial than improving the quality of work.

Table 7.11: Normalised sensitivity coefficients

Normalised Sensitivity Coefficient for:	Value
Quality of Work	0.3
IA Quality	0.6

7.4 EVALUATION OF THE IMPACT ANALYSIS IMPROVEMENT STRATEGIES

Finally, after defining the adapted rework cycle model and describing and testing its simulation (Step 1, Step 2, and Step 3 in the modelling and simulation method applied, Figure 7.8), the adapted rework cycle simulation can be used to examine and compare the practical IA improvement strategies derived from the empirical studies (Section 5.4) through varying the key simulation parameters (Table 7.4). Embodying these strategies through simulation scenarios with different parameter values indicates each strategy's effect on the aerospace company's design process, and, in turn, the strategies can be evaluated in terms of their simulated consequences.

Given the discussion of the modelling and simulation limitations shown in Table 6.1 (Step 1 of model definition in the modelling and simulation method applied), the quantitative simulation results do not indicate the exact outcome of implementing these IA improvement strategies in that the un-modelled added work, obsoleted work, people, and productivity parameters as well as the focus on modelling changes and rework also can have an influence on the design process. However, assuming the negligible influence of these simulation limitations since they do not alter the fundamental characteristics of the aerospace company's design process (Section 6.3.4) and given that the simulation produces results are applicable to this firm's design process (Step 3 of simulation testing in the modelling and simulation method applied), the simulation results can be compared quantitatively to estimate the relative potential for process improvement through IA improvement. In turn, Section 8.2 further explores the effectiveness of these IA improvement strategies in comparison to other process improvement strategies and takes into account the limitations of these simulation results.

7.4.1 SIMULATION OF THE IMPACT ANALYSIS IMPROVEMENT STRATEGIES

The practical IA improvement strategies (Section 5.4 and Table 7.12) can be interpreted through shifting the means and standard deviations of the rework discovery time delay and IA quality variables. In particular, the first strategy addressing method definition and implementation can be represented by improvements in both the rework discovery time delay and IA quality variables. By defining a process for applying IA throughout design processes, IA may be applied earlier and produce higher quality results. In turn, the second strategy to improve information quality may only affect IA quality, while the third strategy can also allow for IA to be implemented earlier as well as improve IA quality given supplementary time and resources available for IA. Table 7.12 correlates the strategies with these parameters.

Table 7.12: IA improvement strategy and simulation variable correlation

IA Improvement Strategy	Strategy Description (from Section 5.4)	Perturbed Simulation Variables (from Table 7.4)
1	Delineating the use of existing or new IA techniques, providing resources to administer these techniques, and educating designers on these techniques	Rework discovery time delay, IA quality
2	Defining processes for sharing and updating information used in IA techniques and tools	IA quality
3	Providing sufficient resources and time for IA	Rework discovery time delay, IA quality

In turn, Table 7.13 depicts the key variable values selected for four simulation scenarios, each examining shifts in the rework discovery time delay and IA quality means or standard deviations separately. As such, the quantitative results of these simulation scenarios suggest the potential impact of improving the rework discovery time delay and IA quality parameters and are then correlated to the IA improvement strategies through their interpretation in Section 7.4.3. Given that the adapted rework cycle simulation can demonstrate the trade-offs between parameters, the variables represented in these strategies (rework discovery time delay and IA quality) are also simulated with different quality of work values to determine their relative influence on the design process. The comparison of the IA improvement variables to the quality of work is later referenced when examining IA improvement strategies against other design process improvement strategies in Section 8.2.

In these scenarios, the rework discovery time delay mean is varied at approximately monthly intervals, and the IA quality and quality of work variables are increased from their baseline values of 0.5, as estimated by the aerospace company (Section 7.2). Given that the simulation only suggests the relative consequences of the improvement of such parameters, as opposed to indicating the exact improvement in process duration (as shown in Figure 7.6), these values are chosen notionally to demonstrate their potential influence, and the interpretation of these simulation results in Section 7.4.3 discusses the effectiveness of the IA improvement strategies, taking into account the relative improvement in the magnitude of these values.

Table 7.13: Simulation scenarios for IA improvement strategies

Simulation Scenario	Variables	
	Variable	**Value(s)**
1 *Perturbing: rework discovery time mean*	Rework discovery time delay mean	31, 62, and 93 days
	Rework discovery time delay standard deviation	3.1 days
	IA quality mean	0.5
	IA quality standard deviation	0.05
	Quality of work mean	0.5, 0.6, 0.7, and 0.8
	Quality of work standard deviation	0.05
2 *Perturbing: rework discovery time standard deviation*	Rework discovery time delay mean	31, 62, and 93 days
	Rework discovery time delay standard deviation	15, 31, and 62 days
	IA quality mean	0.5
	IA quality standard deviation	0.05
	Quality of work mean	0.5, 0.6, 0.7, and 0.8
	Quality of work standard deviation	0.05
3 *Perturbing: impact analysis mean*	Rework discovery time delay mean	31 days
	Rework discovery time delay standard deviation	3.1 days
	IA quality mean	0.5, 0.6, 0.7, and 0.8
	IA quality standard deviation	0.05
	Quality of work mean	0.5, 0.6, 0.7, and 0.8
	Quality of work standard deviation	0.05
4 *Perturbing: impact analysis standard deviation*	Rework discovery time delay mean	31 days
	Rework discovery time delay standard deviation	3.1 days
	IA quality mean	0.5, 0.6, 0.7, and 0.8
	IA quality standard deviation	0.01 and 0.1
	Quality of work mean	0.5, 0.6, 0.7, and 0.8
	Quality of work standard deviation	0.01

7.4.2 Simulation Results

For the first scenario investigating variations in the rework discovery time delay mean (Table 7.13), the simulation shows that decreasing this mean improves the design process in that a smaller mean increases the rate of work done. Specifically, the simulation indicates that decreases in the rework discovery delay mean increases the time difference (Δt) between cases simulated[52] with all other variable values held constant (Figure 7.10).

[52] This time difference is calculated across 200 simulation runs for each case as indicated in Section 7.1.2.

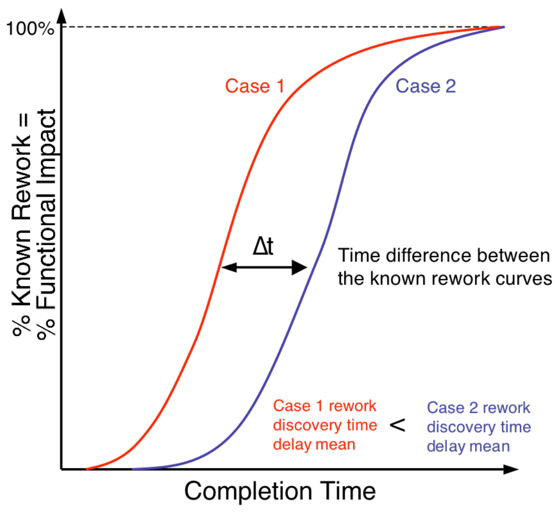

100%
% Known Rework =
% Functional Impact
Case 1
Case 2
Δt
Time difference between the known rework curves
Case 1 rework discovery time delay mean
<
Case 2 rework discovery time delay mean
Completion Time

Figure 7.10: Simulation result analysis for decreasing the rework discovery time delay means between cases (from Figure 7.6)

Figure 7.12 illustrates these increases in time difference for simulated cases with different rework discovery time delay means as well as shows the simultaneous effects of quality of work values changing from 0.5 (Case 2 in Figure 7.10) to an improved quality of work value displayed on the x-axis (Case 1 in Figure 7.10). As expected from the adapted rework cycle behaviour (Figure 6.16), the improvement is uniform based on the rework discovery delay mean values used in compared cases since the known rework curve is shifted uniformly (Figure 7.11). Performing t-tests across these simulation results indicates that compared cases with unique rework discovery time delay differences are significantly different to the 0.05 level, and those with the same differences are significantly similar to the 0.05 level.

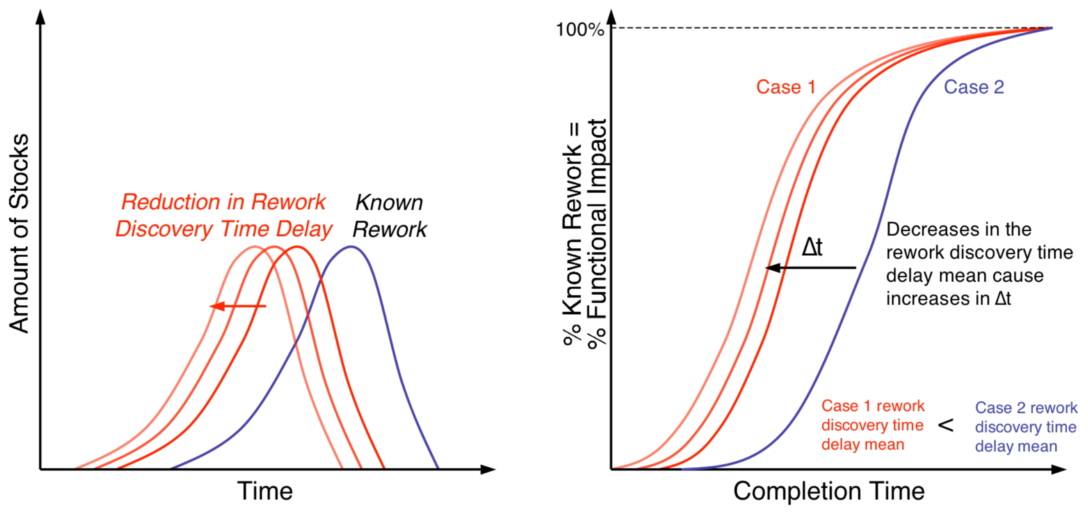

Amount of Stocks
Reduction in Rework Discovery Time Delay
Known Rework
Time
100%
% Known Rework =
% Functional Impact
Case 1
Case 2
Δt
Decreases in the rework discovery time delay mean cause increases in Δt
Case 1 rework discovery time delay mean
<
Case 2 rework discovery time delay mean
Completion Time

Figure 7.11: Expected shifts in the known rework curve due to decreasing the rework discovery time delay means (from Figure 6.16)

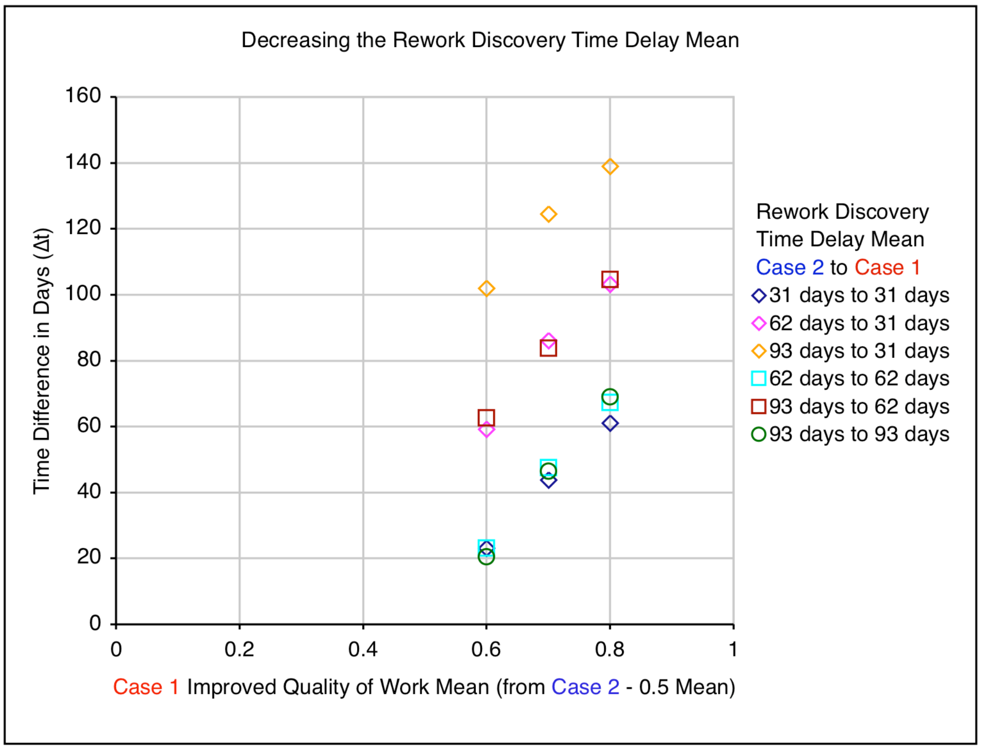


Figure 7.12: Simulation results for decreasing rework discovery time delay means

In turn, for the second simulation scenario examining the rework discovery time delay standard deviation (Table 7.13), large standard deviations of the rework discovery delay can cause insignificant differences to the 0.05 level from the simulation results depicted in Figure 7.12 for the first simulation scenario. Specifically, having a standard deviation of 15 days for each of the rework delay improvements shown in Figure 7.12 does not significantly influence the mean time differences calculated. Fifteen days can be considered large since larger standard deviations, given a rework discovery time delay mean of 31 days, can produce negative delays, which cannot be interpreted in an actual design process. If larger standard deviations are used for the larger rework discovery delay means simulated (*e.g.* a standard deviation of 62 with a mean of 92), such that this positive rework discovery time condition is met, the average time differences also do not vary significantly to the 0.05 level from those in Figure 7.12. Consequently, the simulation indicates that decreasing the mean of the rework discovery time delay can directly increase the mean time difference between cases, while changing the standard deviation has little effect. Figure 7.13 compares the mean time differences (Figure 7.10) for large standard deviations with the standard deviations used in Figure 7.12 for the first simulation scenario.

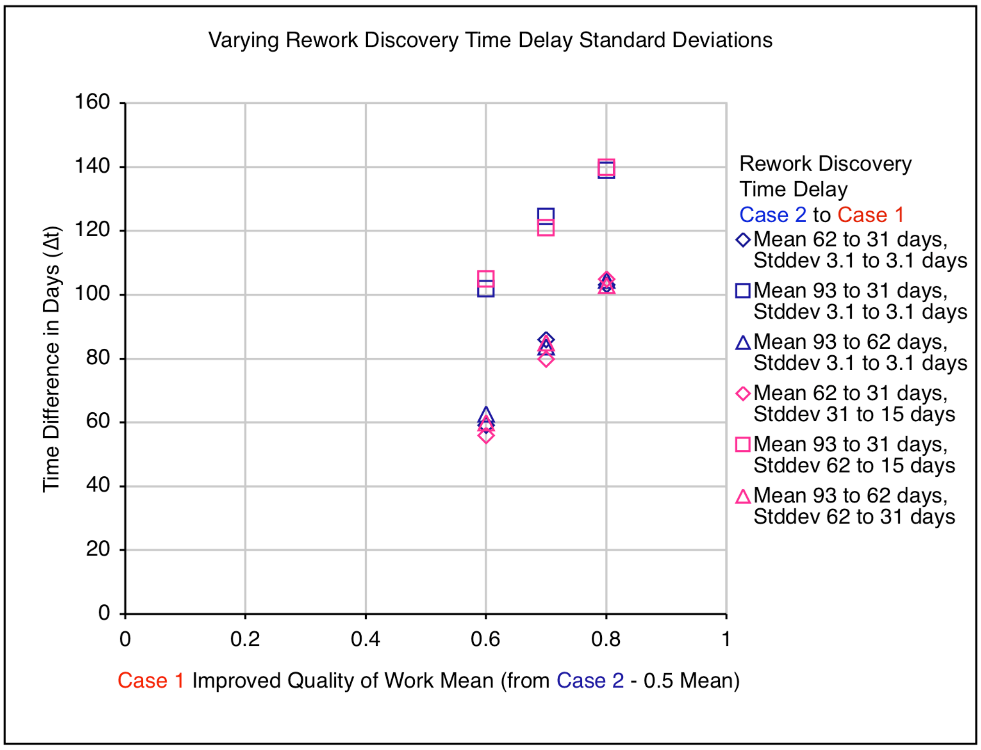


Figure 7.13: Simulation results for varying rework discovery time delay standard deviations

In the third simulation scenario investigating variations in the IA quality mean (Table 7.13), IA quality and quality of work improvements are simulated, and the results indicate that increasing either IA quality or quality of work can cause similar time difference (Δt) reductions. Figure 7.14 and Figure 7.15 show the different types of shifts in the known rework curve due to these improvements, and Figure 7.16 depicts the results obtained from the simulation. T-tests show that these time differences caused by the same increases in either IA quality or quality of work variables are all insignificantly different to the 0.05 level and that they are also significantly different to other increases in these values. In other words, a 0.2 improvement in IA quality is equivalent to a 0.2 improvement in the quality of work, which both differ from the time difference produced by setting either of these values to 0.3. The improvements due to singularly increasing IA quality are expected based on the behaviour of the known rework curve (Figure 7.14). However, obtaining comparable results for increasing either IA quality or the quality of work is unexpected to some extent in that the same improvement occurs through different types of shifts in the known rework curve (further discussed in Section 8.2).

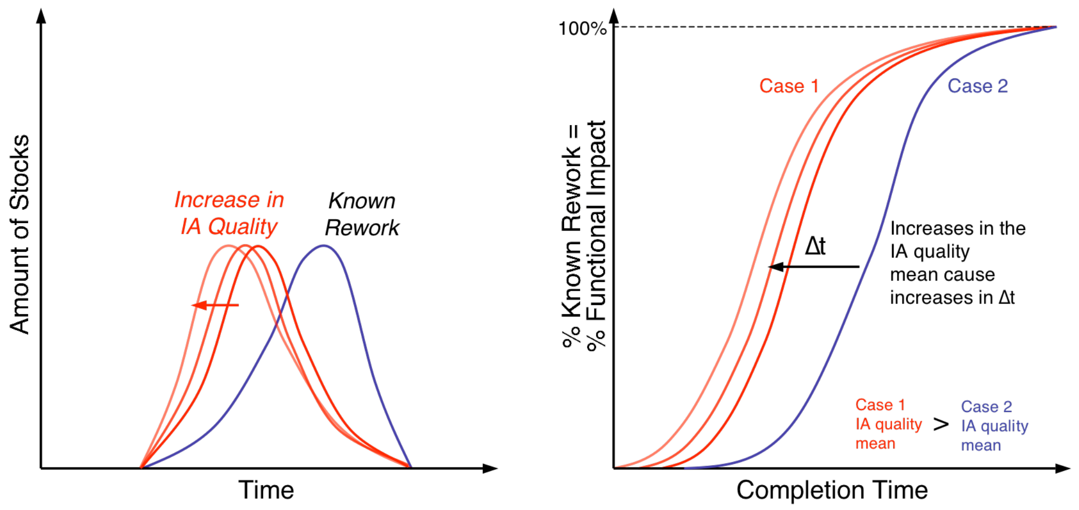


Figure 7.14: Expected shifts in the known rework curve due to increasing the IA quality (from Figure 6.16)

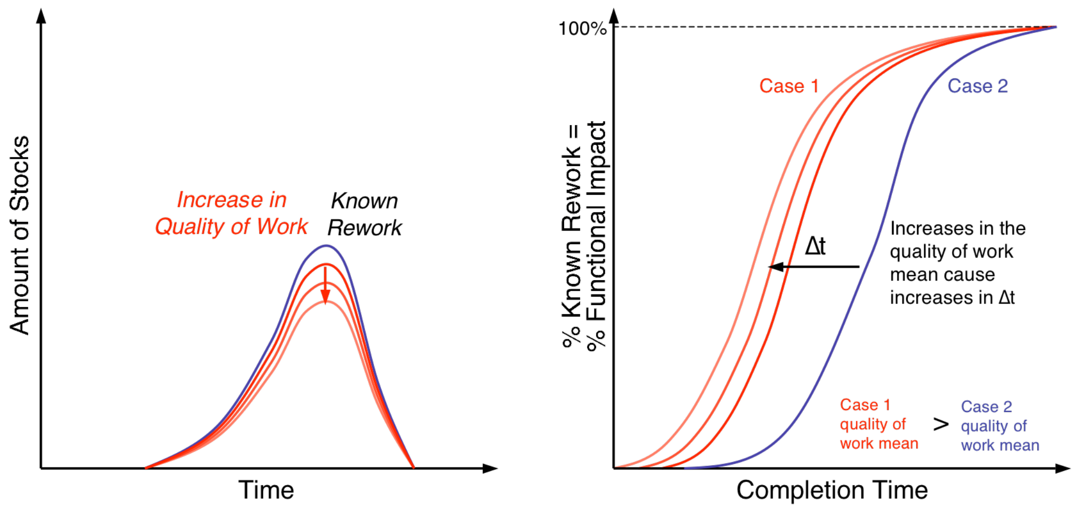


Figure 7.15: Expected shifts in the known rework curve due to increasing the quality of work

For the fourth simulation scenario for the IA quality standard deviation (Table 7.13), large standard deviations in the IA quality variable can lead to insignificant changes in time differences to the 0.05 level to those with small standard deviations, similar to the examination of the rework discovery time delay variability. In this case, a standard deviation value of 0.1 can be considered large in that, given a normal distribution and the range of IA quality values investigated, larger values exceed beyond the IA quality limits of 0 to 1. Hence, increasing the IA quality or quality of work means primarily influences the potential time difference decrease between cases. Figure 7.17 compares the mean time differences for small and large standard deviations for IA quality improvements with a range of quality of work values all with standard deviations of 0.01.

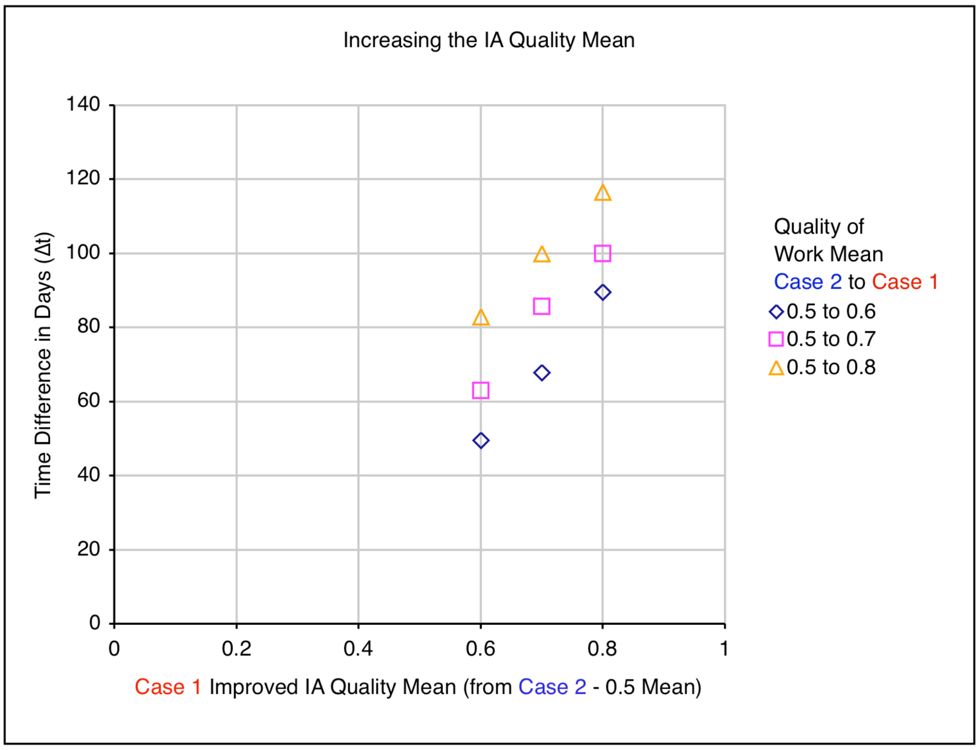


Figure 7.16: Simulation results for increasing IA quality means

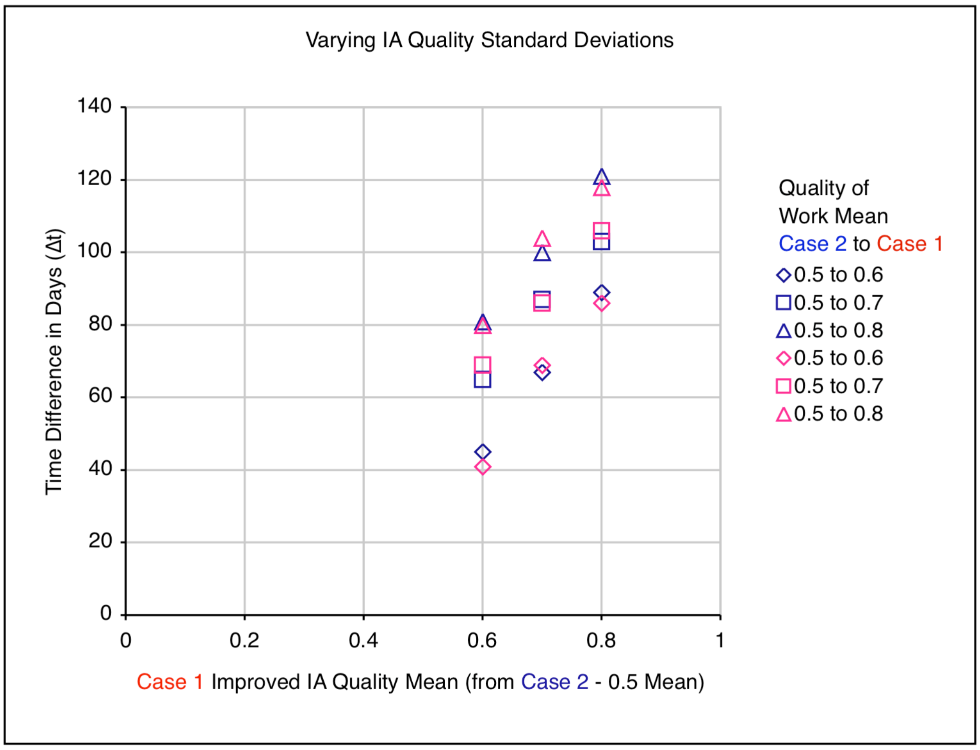


Figure 7.17: Simulation results for varying IA quality standard deviations

7.4.3 INTERPRETATION OF THE IMPACT ANALYSIS IMPROVEMENT STRATEGIES

As suggested by the first strategy for IA improvement (Table 7.12), delineating and supporting the regular application of IA through design process procedures and policies can reduce the rework discovery time delay mean and standard deviation, potentially inducing process improvement. In particular, the standardisation of procedures for implementing IA techniques can require the implementation of IA earlier in the design process and minimise the spread of times to find changes. For example, a policy could mandate that designers must regularly hold design review meetings (a form of experiential IA) and specify procedures to search for new or undiscovered errors and their associated changes[53]. Such a policy could decrease the rework discovery delay mean and standard deviation since IA would be applied consistently and systematically. The application of traceability or dependency IA similarly could be standardised for regular implementation, as suggested by designers to perform "more" IA (Section 5.3.1). In turn, the results for the first simulation scenario investigating the rework discover delay mean show that decreasing this mean can increase the rate of work done. By completing more of the design earlier, the necessary project resources and development costs can be reduced, and the design process can be improved. However, the results for the second simulation scenario for the rework discovery delay standard deviation indicate that decreasing the standard deviation of the rework discovery delay does not significantly affect process improvement. Consequently, attempting to provide detailed procedures to find new or undiscovered errors during IA in order to reduce the variability in identifying changes may not prove beneficial.

Similarly, the third IA improvement strategy (Table 7.12) provides resources and time to perform IA, allowing for the decrease in the rework discovery time delay mean and standard deviation by frequently or thoroughly applying IA. Using additional resources and time to frequently apply IA corresponds to decreasing the rework discovery delay mean and can improve the design process, as depicted by the first simulation scenario. However, as with the first IA improvement strategy, allocating and requiring the use of an excess of additional resources and time to find new or undiscovered changes to reduce the rework discovery time delay standard deviation does not ensure process improvement, as shown by the second simulation scenario.

[53] These errors would correspond to initiating or independent changes, as the discovery of knock-on changes is associated with IA quality improvement (Section 6.3.2).

However, defining and supporting procedures[54] and allocating resources and time[55] to find knock-on changes in the first and third IA improvement strategies can also allow for process improvement through increasing the IA quality mean. As illustrated by the third simulation scenario, increasing the mean IA quality value leads to increasing the rate of work done. However, decreasing the IA quality standard deviation does not enhance this improvement, as shown by the fourth simulation scenario. As such, standardising procedures in detail to define the search for potential knock-on change impacts for each IA techniques available (*i.e.* decreasing the IA quality standard deviation) may not significantly change the basic improvement due to increasing the IA quality mean. Similarly, delineating the resources and time that must be used to find knock-on changes and reduce the IA quality variance may not provide for process improvement.

Comparatively, the second strategy for IA improvement (Table 7.12) can also affect the IA quality mean and standard deviation. Increasing the access to information for IA[56] can increase the IA quality mean and induce process improvement. In turn, procedures can require, for example, that designers gather all information available at an application of IA in order to reduce the IA quality standard deviation. However, such policies may not produce significant gains in process improvement.

In summary, the simulation scenarios suggest that the three practical IA strategies[57] from the empirical studies (Table 7.12) can effectively increase the rate of work done through improving the rework discovery time delay and IA quality means, leading to process improvement in the aerospace company and addressing the fourth research question (Figure 5.12). Although the policies associated with these strategies can also affect the standard deviations of these variables, implementing standardised, detailed procedures specifically to reduce these variances may not improve the design process, as shown by the simulations. As such, a trade-off emerges in that the standardisation of IA practice can cause improvement of the rework discovery time delay and IA quality means to an extent, but over-standardisation may constrain the implementation of IA in practice and require excessive and unnecessary resources and time, only reducing the standard deviations

[54] For instance, a policy could require that designers perform multiple IA techniques at every application of IA to provide for the identification of knock-on changes. Alternatively, as elicited from designers in Section 5.3.3, new, advanced methods and tools for IA to find knock-on changes can be developed.

[55] In this case, resources and time are provided to search for knock-on changes once an error has been discovered.

[56] Section 5.3.2 discusses elicited suggestions by designers to increase information quality for IA.

[57] These practical IA improvement strategies can be implemented through the specific policies suggested by designers during the empirical studies (*i.e.* contained within the categories: (1) including more IA within change processes, (2) improving the input quality to IA, and (3) advancing the IA techniques performed), as discussed in Section 5.3.

of these variables. Consequently, the first IA improvement strategy, suggesting the definition of IA implementation processes, may incur additional costs in terms of resources and time than the third IA improvement strategy for the same magnitudes of improvement in the rework discovery time delay and IA quality variables due to its promotion of process standardisation. Furthermore, given that the second IA improvement strategy, focusing on information access, only affects the IA quality variable, it may provide less process improvement than the third strategy. As such, effectively allocating additional resources and time to IA, as indicated by the third strategy and given the range of IA techniques available within the aerospace company (Table 3.1), may be the simplest means for process improvement and provide the most direct benefit out of the three practical IA improvement strategies. In turn, this strategy may require fundamental changes in management decisions to focus on the handling of rework, and, without the commitment of resources to IA, the other two strategies may better enforce IA improvement.

7.5 EXAMINING PROCESS IMPROVEMENT THROUGH CHANGE PACKAGING

As suggested in the description of the adapted rework cycle model (Section 6.3.3), change packaging also can modify the implementation and generation of changes and, in turn, improve design processes. As such, the adapted rework cycle simulation incorporates change packaging to address research question 4c (Figure 5.14) and is used to investigate change packaging policies for the aerospace company to complete the final step, Step 4, in the modelling and simulation method applied (Figure 6.4).

7.5.1 SIMULATION OF CHANGE PACKAGING STRATEGIES

The design of the adapted rework cycle simulation used to assess the IA improvement strategies does not include change packaging and assumes that no planning delay occurs between change identification and implementation(Section 7.1.1). In turn, the modelling of change packaging essentially includes such a planning delay and adds another simulation variable (*i.e.* another change object attribute to Table 7.1), *change packaging time*, to capture this delay. Specifically, the simulation design is modified in that the change packaging time is delayed from the *implementation time* change object attribute, and the *completion time* attribute is set from the change packaging time instead of the implementation time. This change packaging delay from the implementation time is variable since change packaging times are set at fixed intervals throughout the design process, and functional impact values for independent changes and change sets that are packaged together are given the same change packaging times[58]. Consequently, this modelling of change packaging can be interpreted as rework planning occurring at discrete times in the design process as opposed to continuously, which corresponds to practice at the aerospace company (I-2, I-16). Furthermore, only a portion of functional impact values, which are packaged together, is modelled to generate additional change objects according to the quality of work parameter. This improvement factor, captured by the *quality of work improvement factor* simulation variable, accounts for the reduction of errors through the packaging of dependent modifications and is additive to the quality of work value (Section 6.3.3).

The addition of the change packaging time and quality of work improvement factor variables allows for change packaging policies to be investigated. However, given this simple parameterization of the design process through the adapted rework cycle, this examination is essentially limited to the influence of the timing of change implementation with respect to the discovery of changes. As such, the change packaging policies simulated focuses on performing changes quickly versus implementing modifications in bulk later in the design process. The quality of work improvement factor can vary in these two cases in that this factor may tend to increase later in the design process when more changes are packaged together and more design information is available. Table 7.14 describes the change packaging strategies simulated. Variations of these strategies may exist. For instance, changes may be packaged before the next specified packaging interval if a threshold quantity of functional impact values is discovered. However, such policies effectively depend on performing modifications earlier or later in the design process.

Table 7.14: Change packaging strategy and simulation implementation correlation

Change Packaging Strategy	Strategy Description	Simulation Implementation
1	Packaging few changes quickly for implementation	Change packaging times occur at small intervals after the time of the first change in each functional area with a small quality of work improvement factor
2	Packaging many changes slowly for implementation	Change packaging times occur at large intervals after the time of the first change in each functional area with a large quality of work improvement factor

These packaging policies can be investigated through simulating a range of values for the change packaging time intervals as well as improvement factor parameters. In turn, the first simulation scenario compares the improvement due to varying the

[58] In turn, changes are given equal priority and importance despite their discovery time, unlike the two prioritisation methods discussed in software process dynamics literature (Section 6.1.2).

change packaging time interval, and the second scenario examines the impact of a range of improvement factor values. Table 7.15 shows the key variable values used in the simulation scenarios to depict these strategies.

Table 7.15: Simulation scenarios for change packaging

Simulation Scenario	Variables	
	Variable	Value(s)
1 *Perturbing: change packaging time*	Change packaging time interval	15, 31, 62, and 93-day fixed intervals
	Quality of work improvement factor	0
2 *Perturbing: quality of work improvement factor*	Change packaging time interval	31-day fixed intervals
	Quality of work improvement factor	0, 0.05, 0.1, 0.2

These scenarios are run with all other simulation variables (Section 7.2) held constant in order to determine the influence of change packaging. As such, the simulation testing performed for the IA improvement strategies (Section 7.3) applies to these change packaging simulations in that the variable values used are within the acceptable ranges and produce a design process indicative of the aerospace company's design process. Notably, the change packaging time interval and quality of work improvement factor values used in the packaging scenarios (Table 7.15) are notional in that they only characterise improvement as opposed to concretely determining the decrease in design process duration. In turn, the interpretation of the simulation results estimates the relative magnitude of these variables for the change packaging strategy assessment and comparison.

7.5.2 SIMULATION RESULTS

Based on the modelling of change packaging, the first simulation scenario investigating different change packaging time intervals (Table 7.15) shows that decreasing the packaging interval increases the improvement in time difference between known rework curves (Figure 7.18). Accordingly, quickly packaging and implementing changes increases the rate of work done. Figure 7.19 shows this improvement in mean time difference (Δt) for cases[59] with decreases in packaging time intervals.

[59] This time difference is calculated across 200 simulation runs for each case as indicated in Section 7.1.2.

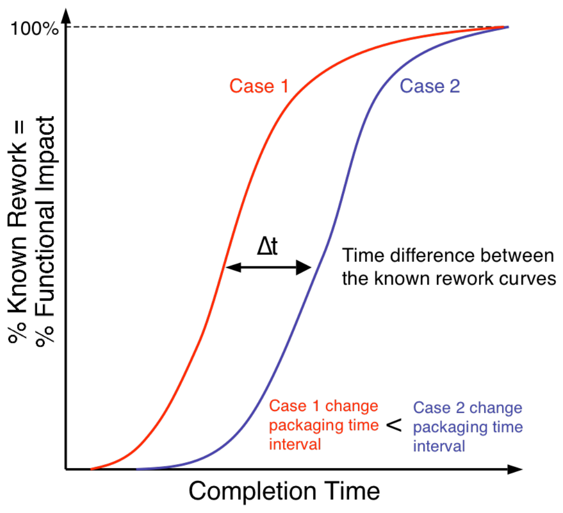

100%
% Known Rework =
% Functional Impact
Case 1
Case 2
Δt
Time difference between the known rework curves
Case 1 change packaging time interval
<
Case 2 change packaging time interval
Completion Time

Figure 7.18: Simulation result analysis for decreasing the change packaging time interval between cases (from Figure 7.6)

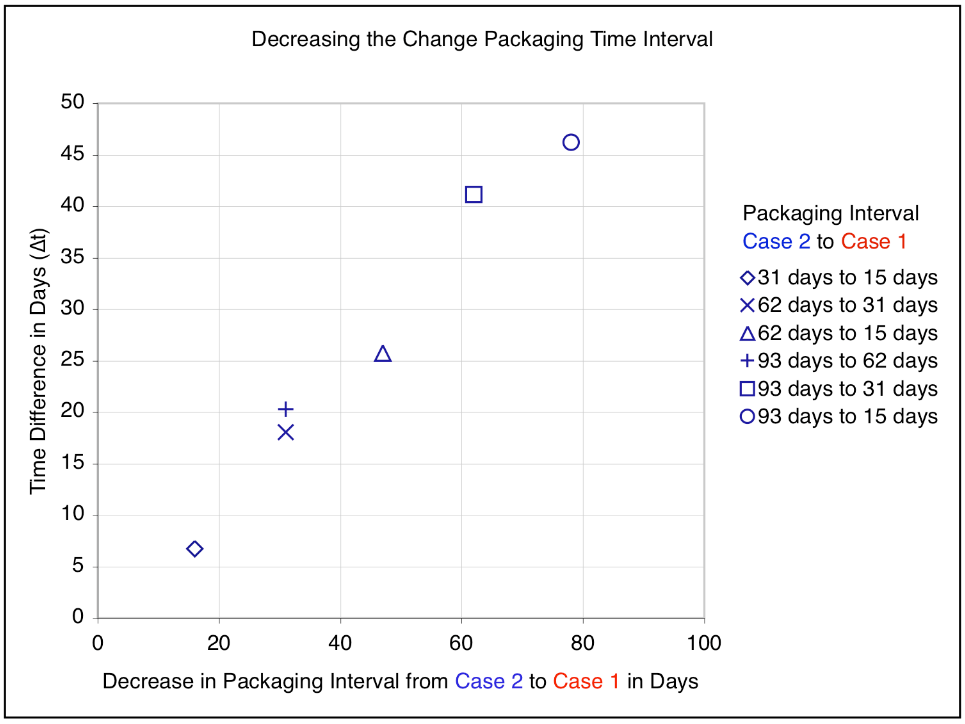


Figure 7.19: Simulation results for decreasing change packaging time intervals

However, change packaging also affects the quality of work parameter. The second simulation scenario (Table 7.15) shows that increasing the quality of work improvement factor with fixed change packaging intervals causes improvements in the mean time difference between known rework curves as expected due to the decrease in the quantity of known rework (shown in Figure 7.15). Figure 7.20 depicts these improvements by comparing baseline cases with quality of work values of 0.4, 0.5, and 0.6 and improvement factors of 0 (Case 2 in Figure 7.15) with improved cases with the same quality of work values and increasing improvement factors (Case 1 in Figure 7.15), shown on the x-axis. Although change packaging is shown to have a slightly greater effect on scenarios with lower quality of work, performing t-tests, these differences are insignificant to the 0.05 level. In turn, given that the improvement factor directly increases the quality of work parameter, the time difference improvements are comparable to IA quality improvement (as discussed in the IA improvement simulations results in Section 7.4.2).

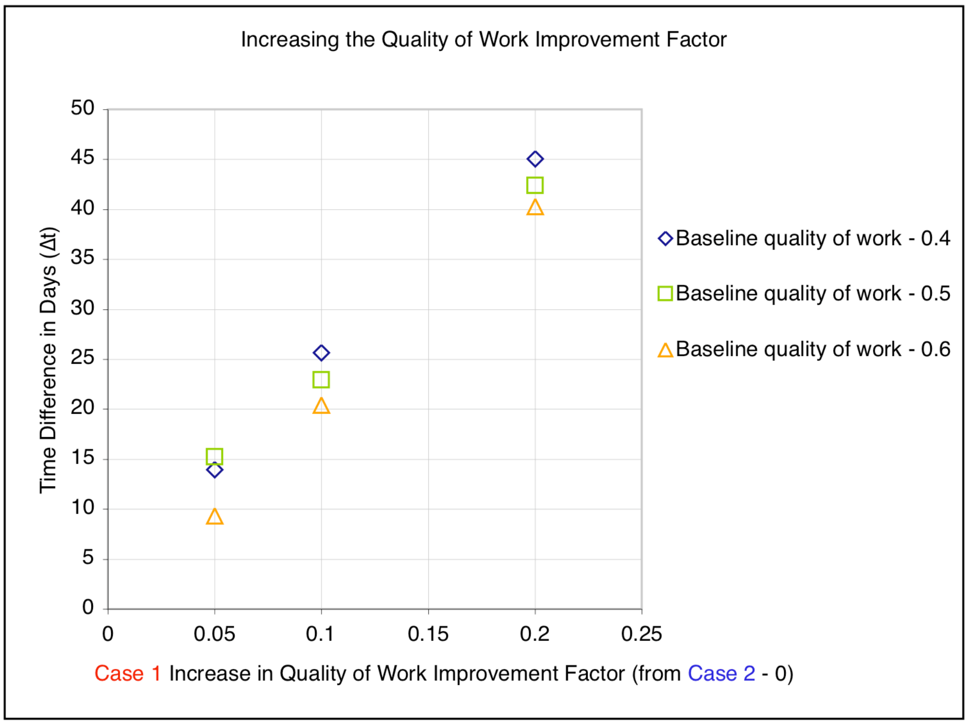


Figure 7.20: Simulation results for increasing improvement factors

7.5.3 INTERPRETATION OF THE CHANGE PACKAGING STRATEGIES

These simulation results for change packaging indicate that decreasing the packaging interval and increasing the quality of work improvement factor can both lead to process improvement and increase the rate of work done. However, as suggested in the description of the change packaging strategies in Section 7.5.1, the change packaging intervals and quality of work improvement factors may be interdependent in that packaging policies affect both variables. Longer packaging intervals to group more design modifications together can have larger quality of work improvement factors. As such, there is a trade-off between packaging and implementing fewer design modifications quickly and delaying work to generate fewer additional, emergent changes. The simulation results exemplify this trade-off in that approximately the same time differences between known rework curves are achieved either by largely decreasing the packaging intervals from 93 days to 31 or 15 days in Figure 7.19 or by significantly increasing the quality of work by a factor of 0.2 in Figure 7.20.

In the case of the aerospace company in which many changes are interdependent, longer packaging delays resulting in higher quality of work values may be a more effective strategy (*i.e.* the second change packaging strategy in Table 7.14) than packaging and implementing changes quickly (*i.e.* the first change packaging strategy in Table 7.14). Implementing this first strategy to significantly improve the design process may be less feasible than the second since decreasing the packaging time intervals to produce such process improvement may be unattainable (*i.e.* a reduction of about 2 months). The regulated processes to document, review, and approve changes for safety-critical certification, which typically can take 2 months, determines the frequency at which changes can be packaged. Significantly increasing the speed of these processes to increase the frequency at which changes can be packaged and implemented may be detrimental to the product's reliability. In turn, given the high interdependency of changes, inducing significant process improvement through change packaging policies that improve the quality of work may be more feasible and prove more beneficial.

As such, this analysis of the two change packaging strategies yields insight to the aerospace company's practice of change packaging in that this firm often packages and implements changes quickly instead of focusing on increasing their quality of work due to pressure by customers to deliver certain design modifications earlier than others. This practice mitigates the potential for process improvement through change packaging, as suggested by the simulations. However, such customer pressure also suggests that packaging strategies may not have the potential to yield as significant process improvement as other strategies in that customer demands often cannot be negotiated and change packaging policies to increase the quality of work cannot be adhered to. For instance, the software design process may be improved more readily through the IA improvement strategies (Section 7.4) in which the aerospace company has more control over their policy implementation.

7.6 SUMMARY

This chapter completes the steps in the modelling and simulation method applied (Figure 7.21), and the interpretation of the simulation results yields insights into the IA improvement and change packaging strategies for the aerospace company. By investigating trade-offs between the simulation parameters of the adapted rework cycle to represent these strategies, the simulation results indicate that:

- The three IA improvement strategies suggested (Section 5.4 and Table 7.12) can all improve the software design process of the aerospace company.
- However, the third IA improvement strategy, focusing on allocating resources and time to IA, may more effectively lead to process improvement than the other strategies.
- In turn, change packaging strategies may not yield process improvement as simply as the IA improvement strategies within the aerospace company in that these policies cannot be consistently implemented and are affected by customer demands.
- Nevertheless, change packaging strategies, which delay change implementation to increase the quality of work, best suit the aerospace company.

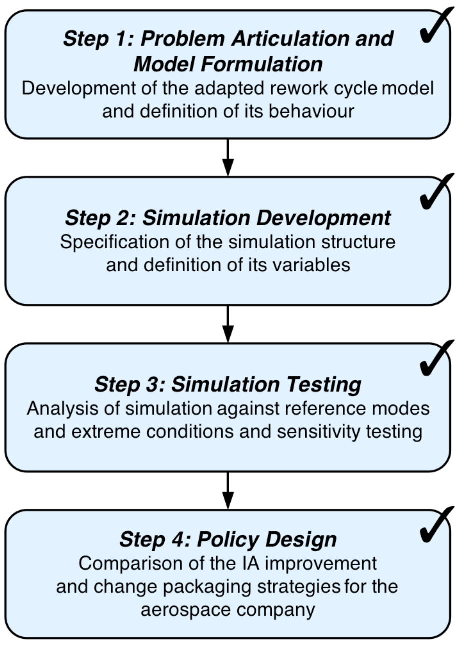

Step 1: Problem Articulation and Model Formulation
Development of the adapted rework cycle model and definition of its behaviour
✓
Step 2: Simulation Development
Specification of the simulation structure and definition of its variables
✓
Step 3: Simulation Testing
Analysis of simulation against reference modes and extreme conditions and sensitivity testing
✓
Step 4: Policy Design
Comparison of the IA improvement and change packaging strategies for the aerospace company
✓

Figure 7.21: Completion of the application of modelling and simulation method

8 :: HEURISTICS FOR PROCESS IMPROVEMENT

The motivation and aim for modelling and simulating the adapted rework cycle is to compare IA improvement and change packaging strategies with other process improvement initiatives (*e.g.* optimising design task scheduling or developing better requirement management practices) in order to ascertain the relative importance of these strategies for the aerospace company. As indicated in Section 6.1.1, the extended rework cycle model (Figure 6.2) provides a basis for such a comparison by depicting a range of strategies affecting the parameters in the basic rework cycle. The magnitudes of these parameters can be estimated due to different strategies and the consequences on the rework cycle determined, suggesting the relative effects of these strategies on the design process. In turn, this chapter adapts the extended rework cycle model to include IA and change packaging and then uses this model to examine the simulation results obtained for IA improvement (Section 7.4) and change packaging (Section 7.5), which perturb the model parameters to different extents, and to compare process improvement strategies. Heuristics for IA improvement and change packaging, indicating when these strategies are best employed compared to other process improvement strategies, are derived from this simulation result analysis and from a qualitative investigation of the trade-offs between the parameters in the adapted rework cycle not simulated. The feedback from the industry partners on these heuristics is then discussed.

Prior to deriving and evaluating these heuristics for process improvement in Section 8.2 and Section 8.3, respectively, the refined research questions from Section 5.6 are revisited in Section 8.1, connecting these questions with the modelling and simulation of the adapted rework cycle performed in Chapter 6 and 7 and with the foreshadowed derivation of heuristics in Chapter 8.

8.1 REVISITING THE REFINED RESEARCH QUESTIONS

As indicated in Section 6.1.1 and Section 6.3.4, the adapted rework cycle model includes sufficient parameters to characterise IA improvement and change packaging. Specifically, improvements to the rework discovery time delay and IA quality variables can embody the practical IA improvement strategies (Section 7.4.1), and these strategies are compared by estimating the magnitudes of improvements in these two parameters and then simulating the effect on the adapted rework cycle. The simulation results of such strategies denote the relative design process improvement through the time difference between known rework curves depicted in Figure 7.6. Larger time differences correspond to increases in the rate of work completed or greater process improvement. Hence, the modelling and simulation of the adapted rework cycle provide a means to investigate IA improvement on the design process through modifications to the rework discovery time delay and IA quality parameters in the adapted rework cycle, addressing research question 4b (*Can modelling and simulation demonstrate the effects of impact analysis improvement on the design process? If so, how?*). Furthermore, IA and other process improvement strategies can be compared by estimating the magnitudes of the other variables in the adapted rework cycle model and determining their relative effects on the design process, as done in Section 8.2 to derive the heuristics for process improvement.

These heuristics developed stem from the simulation results in Chapter 7. However, the limitations of the adapted rework cycle model implementation notably restrict these results (Section 6.3.4). Specifically, the results only suggest the relative, instead of exact, effects of the IA improvement strategies. The simulation of the IA improvement strategies indicate that improving IA through either reducing the rework discovery time delay mean or increasing the IA quality mean can increase the rate of work done, but do not concretely predict the scale this improvement for the aerospace company. Nevertheless, the simulation results imply that these strategies can improve design processes to different extents through small rework discovery time delays and high IA quality, answering research question 4a (*Can the application of impact analysis improve the design process? If so, how?*). The third IA improvement strategy, which proposes increasing the resources and time available for IA, may most directly cause process improvement out of the strategies suggested (Section 7.4.3). As such, the heuristics for process improvement derived in Section 8.2 further respond to this research question by considering the relative effects of IA improvement and other process improvement strategies in terms of the limitations of the adapted rework cycle.

Finally, the adapted rework cycle model allows for the conceptualisation of change packaging. Change packaging essentially controls the implementation time of design modifications and shifts the known rework curve (Figure 7.6), which indicates process improvement as analysed. In addition, change packaging can influence the quality of work parameter in the adapted rework cycle model in that facilitating the recognition of design interdependencies through packaging can reduce design errors. In turn, as suggested by the simulation result interpretation (Section 7.5.3), change packaging policies for the aerospace company should focus on improving the quality of work. Only packaging few changes together quickly for implementation does not yield as significant process improvement. However, for this firm, change packaging policies may not prove as beneficial as improving IA practice because the implementation of change packaging is highly influenced by customer demands and cannot be completely controlled. Nevertheless, implementing change packaging with IA improvement strategies can amplify process improvement. Thus, change packaging does not directly influence IA improvement through modifying the rework discovery time delay and IA quality parameters, but can enhance the results of IA improvement strategies, addressing research question 4c (*Does change packaging affect impact analysis improvement? If so, how?*). The heuristics derived for process improvement in Section 8.2 further relate change packaging and IA improvement strategies by considering other factors influencing the adapted rework cycle model.

8.2 IMPACT ANALYSIS IMPROVEMENT AND CHANGE PACKAGING HEURISTICS

Incorporating other relevant factors and feedback loops into the adapted rework cycle model provides a basis to compare IA improvement strategies and change packaging against other design process improvement strategies (Section 8.2.1), and an analysis of this extended system dynamics model with the simulation results presented in Chapter 7 allows for the extraction of heuristics for IA improvement (Section 8.2.2) and change packaging (Section 8.2.3).

8.2.1 EXTENDING THE ADAPTED REWORK CYCLE MODEL

As described in Section 6.1.1, Lyneis *et al.* (2001) discuss additional factors affecting the people, productivity, and quality of work variables in the basic rework cycle model with obsolescence (Figure 6.14). Figure 8.1 recalls this extension of this model through a causal loop diagram (Appendix C), also pictured in Figure 6.2.

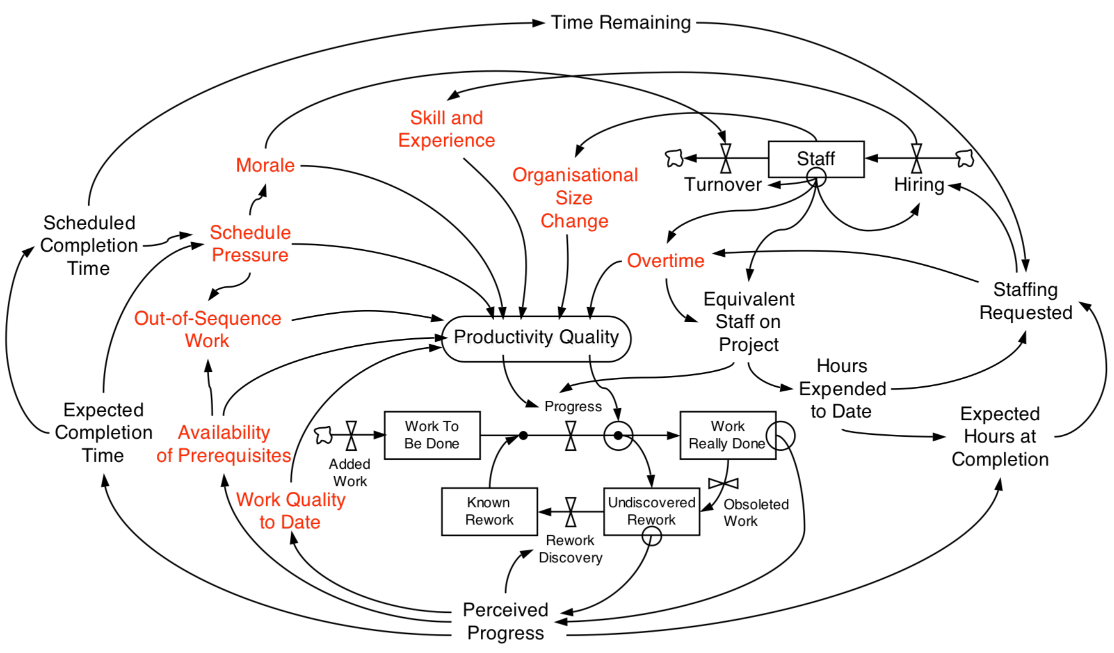


Figure 8.1: The extended rework cycle model (Lyneis *et al.* 2001)

Lyneis *et al.* (2001) focus on several key influences on the productivity and quality of work parameters, including the *work quality to date, availability of prerequisites, out-of-sequence work, schedule pressure, morale, skill and experience, organisational size changes,* and *overtime*. Two of the feedback effects associated with these key variables are positive or reinforcing. Specifically, increasing overtime work or allocating more people to design projects increases the rate of work done. However, the other feedback relationships associated with the key influences depicted are "generally" positive and cause productivity and quality to initially decrease early in design processes and only later induce increases[60]. For example, as resources increase, the average experience level of the staff decreases, which, in turn, decreases productivity and the quality of work since less experienced people work slower and induce more errors. In turn, productivity and quality of work later increase as these individuals gain project experience. As described by Lyneis *et al.*, if resources continue to be added, these negative effects can govern projects and schedule delays can continually increase. Brooks' law embodies this concept and states: "adding manpower to a late software project makes it later" (Brooks 1995).

The key factors identified by Lyneis *et al.* similarly influence the adapted rework cycle model through the productivity and quality of work parameters and also affect the rework discovery time delay and IA quality. For instance, if staff skill and experience decreases, the rework discovery delay may increase and IA quality may

[60] Due to this change in polarity during the projects, link polarities are not explicitly specified in Figure 8.1 according to the notation used in Appendix C.

decrease since the staff may not quickly perform IA and miss necessary changes. The IA technique and task influences observed in the empirical studies (Figure 4.7) exemplify these corresponding feedback loops affecting the rework discovery time delay and IA quality, and Table 8.1 interprets the key feedback parameters depicted by Lyneis *et al.* through these IA influences.

Table 8.1: Correlation between additional factors and IA influences

Extended Rework Cycle Parameter	IA Influence(s) (from Figure 4.7)	Correlation
Work quality to date	Lack of information, Ambiguity of information	Unknown, incomplete, or uncertain information or representation of design dependencies derived from prior work can decrease IA quality. In turn, the perceived work quality can influence when IA is applied and affect the rework discovery time delay.
Availability of prerequisites	Lack of information, Ambiguity of information, Volatility of information, Magnitude of information, Lack of resources	Problematic information or stakeholder availability for IA can influence when IA is applied and cause a decrease in IA quality.
Out-of-sequence work	Partitioning, Synchronisation	IA performed on a portion of a design or asynchronously for related changes can lead to missed knock-on changes and decrease IA quality. In this case, the rework discovery time delay may also increase if previous IA results are assumed complete.
Schedule pressure	Lack of time	Time pressure can cause the rework discovery time delay to increase as less time is used to identify rework, leading to incomplete IA results and decreasing IA quality.
Morale	---	---
Skill and experience	Analysis education	The rework discovery time delay increases and IA quality decreases if inexperienced designers do not understand the methods to apply available IA techniques and tools.
Organisational size changes	Analysis education	As design teams grow after the beginning of a project, new designers must learn about the project in order to quickly implement IA and effectively apply experiential IA techniques. The rework discovery time delay and IA quality may increase and decrease, respectively, as this staff growth occurs.
Overtime	Lack of time	The rework discovery time delay and IA quality may increase and decrease, respectively, if designers are fatigued from analysing many changes.

Notably, the method definition, process conflicts, over-extension, and administration IA influences do not meld with the extended rework cycle parameterization (Figure 8.1) and, consequently, suggest the existence of an additional feedback loop dependent on another parameter, namely, *IA process definition and implementation*. This feedback effect on the rework discovery time delay and IA quality is also "generally" positive since IA processes tend to become more effective and understood towards project completion. However, this parameter also affects the resource allocation to IA directly through the processes prescribed. Arguably, similar effects can influence the productivity and quality of work parameters through the *design process definition and implementation*. For example, the defined sequencing of tasks, which other research at Cambridge EDC investigates (Chalupnik *et al.* 2007; Wynn *et al.* 2006), can be optimised to improve workflow or robustly handle unexpected, adverse factors and minimise the effects of errors. Likewise, specified change packaging processes can affect productivity and the quality of work (Section 8.2.3). Furthermore, morale was not found to influence IA quality in the empirical studies, but can be envisioned to affect other companies and projects[61].

Continuing to use the extended rework cycle model (Figure 8.1) as a framework, the direct means to positively increase the rate of rework discovery can be translated. The direct means to increase the rate of progress is through overtime or staff allocation, and, in turn, improving IA can occur through allocating resources and time to apply IA. As such, the first and second practical IA improvement strategies (Table 7.12) mitigate the negative effects of the "generally" positive, key feedback loops, while the third strategy directly influences the application of IA through resource management, as suggested by the IA simulation interpretation (Section 7.4.3). Figure 8.2 summarises these additional factors affecting the adapted rework cycle through a causal loop diagram.

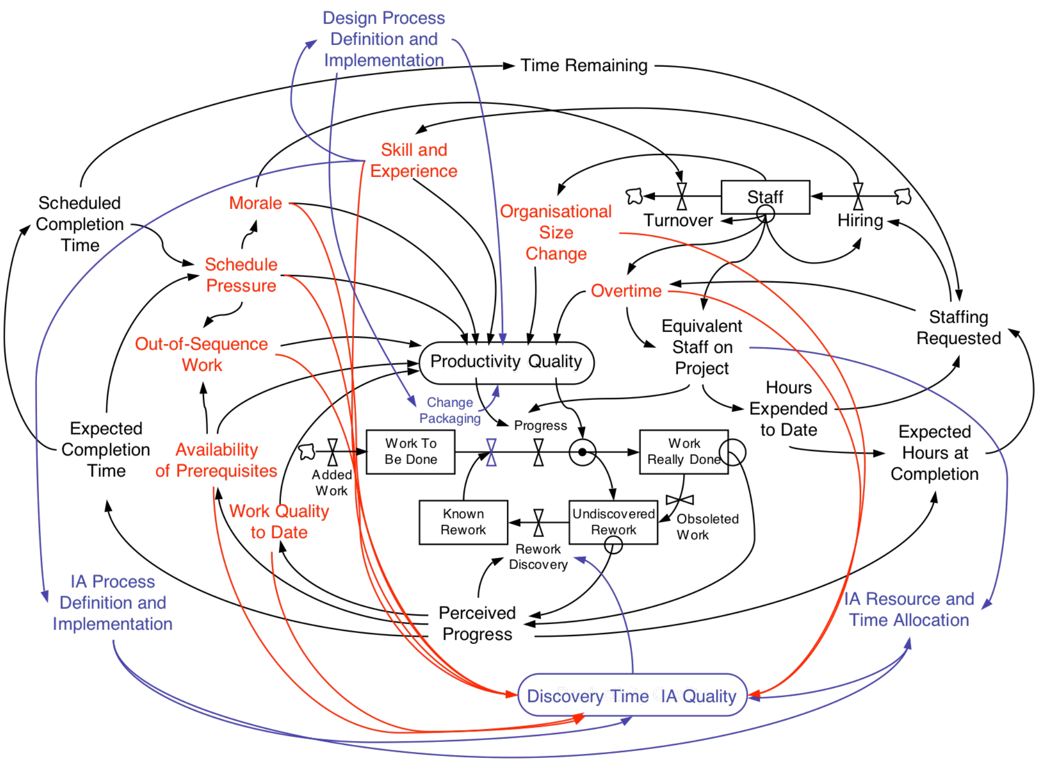


Figure 8.2: The extended, adapted rework cycle model

[61] Even though the adaptation of the extended rework cycle by Lyneis *et al.* (2001) in Figure 8.2 suggests the effects of low morale on IA quality, morale is not included in the IA influences characterisation since this classification only reflects observations in the empirical studies.

As proposed by Lyneis *et al.* (2001), the additional factors depicted in the extension of the adapted rework cycle do not completely cover all of the other potential influences on the design process. Nevertheless, this model provides a framework to examine improving IA and change packaging practice with other process improvement strategies in that trade-offs between parameter magnitudes of the adapted rework cycle model can be determined and a selection of improvement strategies can be correlated with these variables in the extended model[62]. In turn, the discussion of the heuristics derived using the extended, adapted rework cycle model references example process improvement strategies, but the heuristics generically depict the trade-offs between strategies in terms of the parameters in the adapted rework cycle.

8.2.2 EXTRACTION OF HEURISTICS FOR IMPACT ANALYSIS IMPROVEMENT

The IA improvement simulation results (Section 7.4.3) can be further interpreted in order to determine the trade-offs between the parameters simulated (*i.e.* the rework discovery time delay or IA quality vs. the quality of work[63]). In turn, the comparison of these parameters can be related to process improvement strategies through the extended, adapted rework cycle (Figure 8.2) and conclusions can be drawn for when focusing on IA improvement is most beneficial for the aerospace company. The other parameters not simulated (*i.e.* people and productivity) can also be investigated qualitatively in terms of this interpretation of the simulation results. As such, this section compares the following sets of parameters, which are correlated with process improvement strategies:

- IA quality vs. quality of work (Section 8.2.2.1)
- Rework discovery time delay vs. people and productivity (Section 8.2.2.2)
- IA quality vs. people and productivity (Section 8.2.2.3)
- Rework discovery time delay vs. quality of work (Section 8.2.2.4)

The heuristics derived, which are summarised in Figure 8.19, depict the trade-offs between these sets of parameters and guide the comparison of IA improvement (*i.e.* strategies improving the rework discovery time delay or IA quality parameters) with other process improvement strategies (*i.e.* strategies improving the quality of work, people, or productivity parameters). Some of these heuristics are based on the assumptions of the simulation results and, hence, are applicable to the aerospace firm, while others take into account the key limitations of the adapted rework cycle simulation (Table 6.1) and are applicable in other contexts. (Hence, Heuristic 1

[62] As discussed in Section 6.1.1, Lyneis *et al.* (2001) list additional factors that could be included.
[63] The quality of work simulation results were not interpreted in Chapter 7.

through 3 as well as Heuristic 4 through 6 are similar to each other.) Specifically, the change-set size, which influences the interpretation of IA quality, the unmodelled work to do, and obsolescence are discussed (Limitation 2, Limitation 8, and Limitation 9 in Table 6.1, respectively). The other limitations are not addressed since they primarily define the model scope, rather than affect the interpretation, of the adapted rework cycle simulation. Change packaging strategies are analysed in Section 8.2.3 in terms of these heuristics, also accounting for the interpretation of change packaging in the adapted rework cycle (Limitation 6 in Table 6.1).

8.2.2.1 IA QUALITY VS. QUALITY OF WORK

As stated in Limitation 9 in Table 6.1, the adapted rework cycle simulation only models changes occurring and does not encompass the stock of work to do. This assumption correlates to the aerospace company's modification of their design platform to develop new products in that most effort is spent implementing rework to adapt the platform and minimal work ever resides in the stock of work to do. For this situation, the IA simulation results in Section 7.4.3 indicate that improving IA quality or the quality of work means by the same magnitudes can have the same effect on the design process (Figure 8.3).

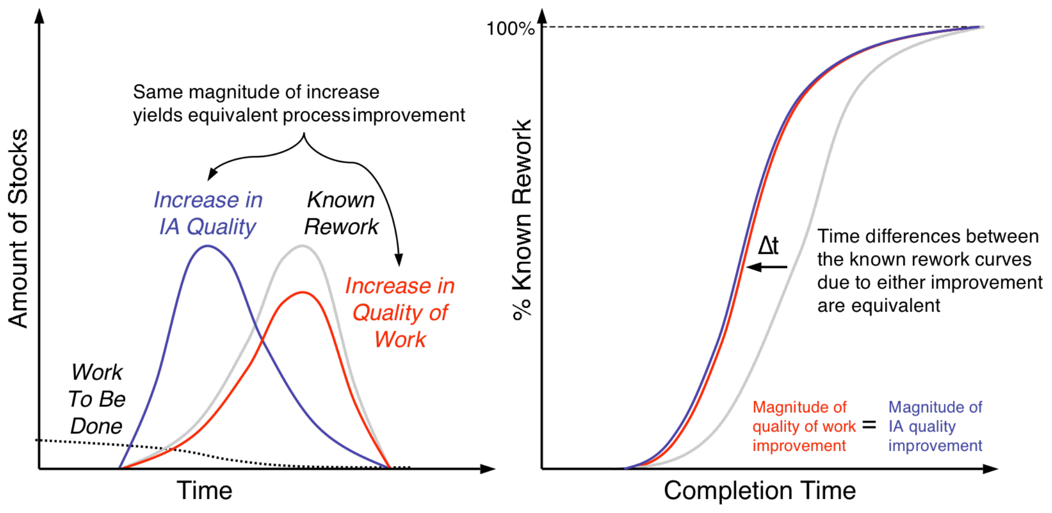


Figure 8.3: Simulation results comparing IA quality and the quality of work (from Section 7.4.3)

However, implementing IA improvement strategies to increase IA quality may be more beneficial in this case since the magnitude of IA quality can be more readily increased through simply allocating more resources and time to IA, as suggested by the third IA improvement strategy (Table 7.12). As simulated, increasing IA quality only necessitates identifying first-order, knock-on modifications (Section 6.3.2), which does not require creating elaborate traceability or dependency IA techniques and can occur through experiential IA. In contrast, improving the quality of work by the same magnitude can require more invasive strategies, such as changing requirement management processes, which can initially disturb productivity significantly, as designers learn the new processes. More generally, as depicted through the causal links in the extended, adapted rework cycle model (Figure 8.2), such imposing strategies can also include improving the *availability of prerequisites*, re-scheduling design tasks to account for *out-of-sequence work*, or providing training to designers to increase their *skill and experience*. Consequently, Heuristic 1 for IA improvement (Figure 8.4) indicates when IA improvement strategies are more beneficial than other process improvement strategies focusing on improving the quality of work.

IA Improvement Heuristic 1:
If the amount of rework is much larger than the amount of work to do, identifying the impact of closely related knock-on changes is more beneficial than decreasing the rework generated.

(i.e. strategies improving IA quality are more beneficial than strategies improving the quality of work)

Figure 8.4: Heuristic 1 for IA improvement

As such, if the stock of work to do is much larger than the amount of rework during the design process, then improving the quality of work may be more effective than increasing IA quality, given that the quality of work influences the amount of rework generated from the work to do and from the rework (Figure 8.5). (For Heuristic 1, improving the quality of work primarily affects the rework generated from rework.) In turn, for this second scenario, improving the quality of work can lead to further reductions of the total amount of rework than that suggested by the simulation results (Figure 8.3) for the same magnitude of increase of this parameter. The associated process improvement caused by this decrease in rework makes improving the quality of work more beneficial than increasing IA quality, as stated in Heuristic 2 for IA improvement (Figure 8.6). Consequently, investing in process improvement strategies to increase the quality of work may prove more useful than implementing IA improvement strategies in this case.

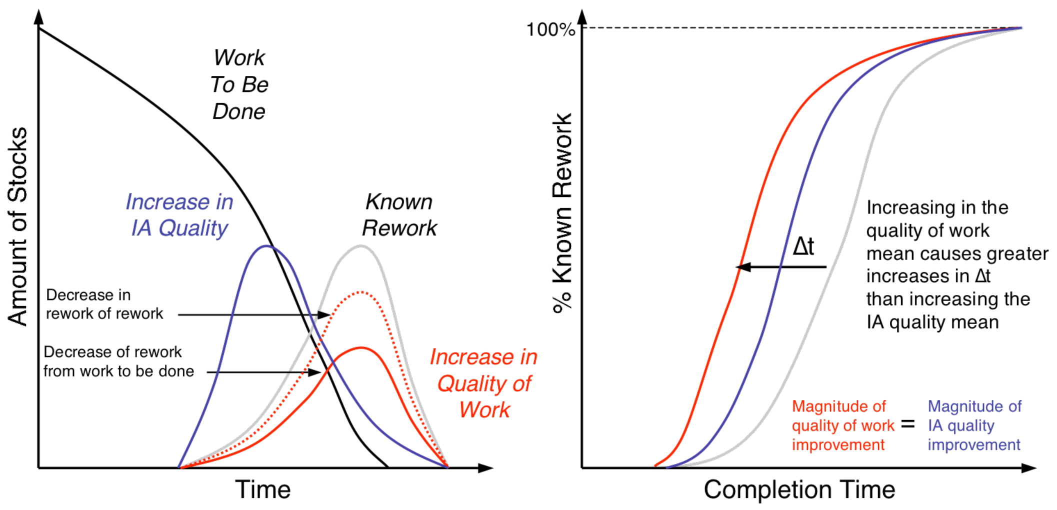


Figure 8.5: Improving the quality of work amplifies the effects on the known rework curve

IA Improvement Heuristic 2:
If the amount of work to do is much larger than the amount of rework, decreasing the rework generated is more beneficial than identifying the impact of closely related knock-on changes.

(i.e. strategies improving the quality of work are more beneficial than strategies improving IA quality)

Figure 8.6: Heuristic 2 for IA improvement

These two heuristics suggest a third in that the amount of work to do typically decreases as the amount of rework increases during product development (Figure 8.7), as discussed by Cooper (1993c). As such, the influence of improving IA quality on process improvement increases as design processes progress, given that rework constitutes more of the work being done. Thus, Heuristic 3 for IA improvement (Figure 8.8) indicates that IA improvement strategies become more beneficial than other strategies that increase the quality of work during design processes. Table 8.2 summarises Heuristic 1 through 3.

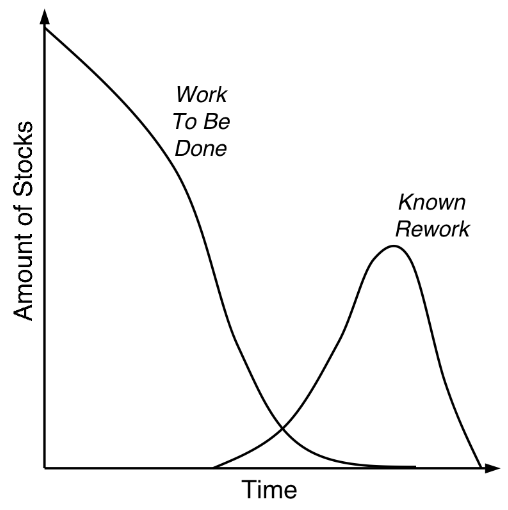


Figure 8.7: Rework dominates the work done towards the end of design processes (from Figure 6.15)

IA Improvement Heuristic 3:
Identifying the impact of closely related knock-on changes becomes more beneficial than decreasing the rework generated as design processes progress.

(i.e. strategies improving IA quality become more beneficial as design processes progress)

Figure 8.8: Heuristic 3 for IA improvement

Table 8.2: Comparison of Heuristic 1 through 3

Heuristic for IA Improvement	Application Context	Comparison of Key Variables
1	Rework >> Work to do	Improving IA quality is better than improving the quality of work.
2	Rework << Work to do	Improving the quality of work is better than improving IA quality.
3	During design processes	Improving IA quality becomes more beneficial than improving the quality of work over time.

The adapted rework cycle simulation does not model the addition or obsolescence of work (Limitation 8 in Table 6.1), which also reflects the aerospace company's focus on modifying their product platform. However, the addition and obsolescence of work can reinforce the previous heuristics derived. From Heuristic 2 (Figure 8.6), if a significant amount of work is added to the stock of work to do, and the stock of work to do is much larger than the amount of rework in the feedback loop, then improving the quality of work remains more effective than increasing IA quality. In turn, if obsolescence of work occurs, extra rework is incorporated into the rework cycle feedback loop. From Heuristic 1 (Figure 8.4), if the amount of rework is much larger than the amount of work to do and obsolescence of work occurs, then improving IA quality also remains more effective than increasing the quality of work. Notably, if both the addition and obsolescence of work occurs, Heuristic 1 and Heuristic 2 also can recommend improving either IA quality or the quality of work given the resulting proportion of work to do and rework.

8.2.2.2 REWORK DISCOVERY TIME DELAY VS. PEOPLE AND PRODUCTIVITY

As denoted by Limitation 10 in Table 6.1, the adapted rework cycle simulation does not include the people and productivity parameters. Nonetheless, as indicated by the extended rework cycle model, increasing either of these variable values directly causes the rate of progress to also increase (*i.e.* there are no feedback loops between these parameters in Figure 8.1), increasing the rate of work completed and allowing for process improvement. Decreasing the rework discovery time delay can be compared to this improvement since this parameter also allows the rate of work completed to increase since rework is known earlier and can be completed subsequently.

As such, similar to the first set of heuristics derived (Table 8.2), improving the rework discovery delay is more beneficial than increasing people or productivity, if the amount of rework is much larger than the amount of work to do, as depicted by Heuristic 4 for IA improvement (Figure 8.10). In this case, decreasing the rework discovery delay can increase the rate of known rework, and consequently, the rate of work completed (*i.e.* work in the stock of work done) and cause process improvement. People may not work at capacity and as productively with less known work to begin with and not influence the rate of work completed (Figure 8.9). In turn, process improvement strategies should focus on improving IA through the rework discovery delay (*e.g.* the first or third IA improvement strategy in Table 7.12) rather than through policies increasing people or productivity (*e.g. hiring* more staff or requiring *overtime*, as shown in the extended, adapted rework cycle in Figure 8.2).

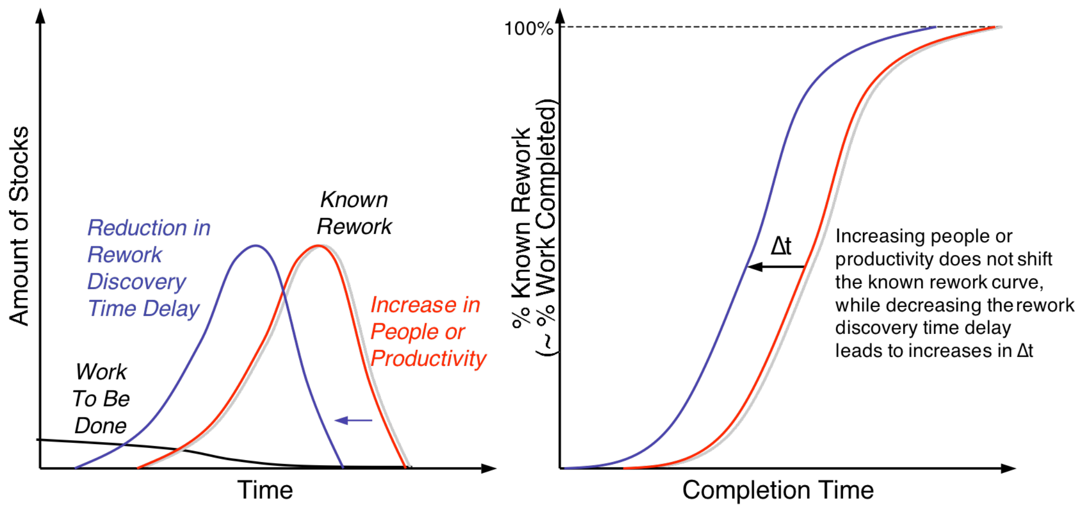


Figure 8.9: Decreasing the rework discovery time delay increases the rate of known rework and work completed

IA Improvement Heuristic 4:
If the amount of rework is much larger than the amount of work to do, identifying the impact of changes quickly is more beneficial than implementing work quickly.

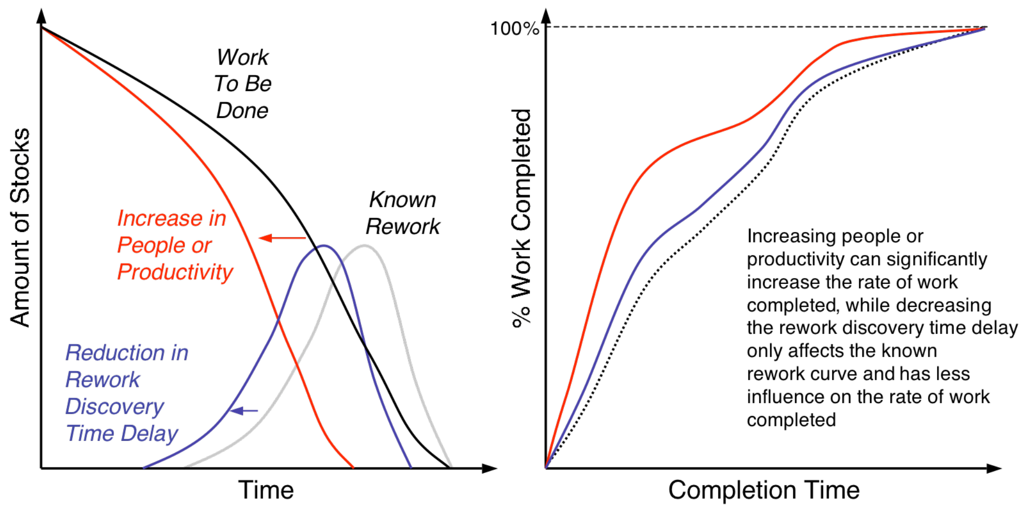
(i.e. strategies improving the rework discovery time delay are more beneficial than strategies increasing people or productivity)

Figure 8.10: Heuristic 4 for IA improvement

Comparatively, if the stock of work to do is larger than the amount of rework, then strategies increasing people or productivity may be more effective than strategies decreasing the rework discovery time delay, as represented by Heuristic 5 for IA improvement (Figure 8.12). Increasing people or productivity with much work to do can more significantly affect the rate of work completed since most of the work is known (Figure 8.11).

Table 8.3: Comparison of Heuristic 4 through 6

Heuristic for IA Improvement	Application Context	Comparison of Key Variables
4	Rework >> Work to do	Improving the rework discovery delay is better than improving people or productivity.
5	Rework << Work to do	Improving people or productivity is better than improving the rework discovery delay.
6	During design processes	Improving the rework discovery delay becomes more beneficial than improving people or productivity over time.



Increasing people or productivity can significantly increase the rate of work completed, while decreasing the rework discovery time delay only affects the known rework curve and has less influence on the rate of work completed

Figure 8.11: Increasing people or productivity increases the rate of work completed (from Figure 6.17)

IA Improvement Heuristic 5:
If the amount of work to do is much larger than the amount of rework, implementing work quickly is more beneficial than identifying the impact of changes quickly.

(i.e. strategies increasing people or productivity are more beneficial than strategies improving the rework discovery time delay)

Figure 8.12: Heuristic 5 for IA improvement

As suggested by the derivation of Heuristic 3 (Figure 8.8), decreasing the rework discovery time delay becomes more effective as design processes progress since more rework than work to do exists (Figure 8.7). Heuristic 6 for IA improvement summarises this trend (Figure 8.13) in that implementing IA improvement strategies affecting the rework discovery delay become more beneficial during design processes. Table 8.3 summarises Heuristic 4 through 6.

IA Improvement Heuristic 6:
Identifying the impact of changes quickly becomes more beneficial than implementing work quickly as design processes progress.

(i.e. strategies improving the rework discovery time delay become more beneficial as design processes progress)

Figure 8.13: Heuristic 6 for IA improvement

8.2.2.3 IA QUALITY VS. PEOPLE AND PRODUCTIVITY

As described by Cooper (1993b) for the rework cycle model, improving the rework discovery time delay may not be useful when the quality of work is low since significant rework continues to be generated. In this case, improving the quality of work first should be the initial strategy. This trade-off holds for the adapted rework cycle. In addition, the adapted rework cycle suggests that increasing people or productivity with low IA quality also may not be effective, if work collects in the stock of unknown rework and limits the rate of work completed, as described by Heuristic 7 for IA improvement (Figure 8.15). Increasing people or productivity does not influence the rate of known rework, as illustrated in Figure 8.9, and identifying quantities of rework more effectively can provide for process improvement in this case (Figure 8.14). Hence, strategies for IA improvement, such as any of those listed in Table 7.12, may be more beneficial than policies affecting people or productivity.

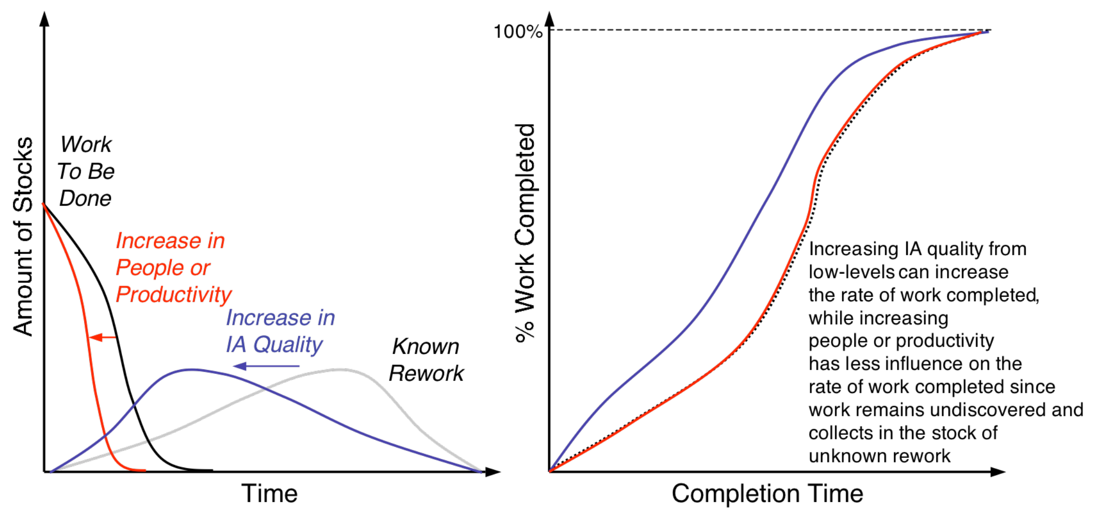


Figure 8.14: Improving IA quality from low-levels

IA Improvement Heuristic 7:
Improving very low IA quality is more beneficial than increasing staff size or productivity.

Figure 8.15: Heuristic 7 for IA improvement

As discussed in Section 6.3.2 and highlighted in Limitation 2 of Table 6.1, the small size of the change sets simulated suggests the impact of improving IA quality for first-order, knock-on modifications. Modelling larger change sets reflects the influence of performing IA to find higher-order, indirect, knock-on changes.

However, even improving IA when progress is limited by the amount of rework that can be discovered earlier, as is the case for Heuristic 7, can provide for process improvement. This variability in the change set size modelled also suggests a trade-off in that strategies focusing on performing IA to find higher-order, knock-on modifications, which can require additional IA resources and time, may not be worth the decrease in people or productivity from diverting resources. Other factors modelled in the extended, adapted rework cycle (Figure 8.2), such as the *quality of work to date*, *availability of prerequisites*, and *out-of-sequence work*, also can significantly influence IA quality, as the IA applied can involve more extensive investigation through traceability and dependency IA. Another simple heuristic cannot determine the management of this trade-off, given that increasing the resources and time for IA may not solely allow for such improvement, and a predictive model simulation is required (Section 9.3).

8.2.2.4 REWORK DISCOVERY TIME DELAY VS. QUALITY OF WORK

Finally, Cooper (1993b) also indicates that decreasing the rework discovery time delay at high quality of work levels may not yield significant improvements since the work performed is directly completed. Although this relationship holds in the adapted rework cycle, in comparison, at high IA quality levels, increasing people or productivity can still increase the rate of work done since unknown rework does not limit the amount of work that can be performed. Consequently, high IA quality enables improvement in design processes involving rework (*i.e.* when the quality of work is not at high levels), but does not guarantee process improvement as high quality of work does. Figure 8.16 displays Heuristic 8 for IA improvement. In turn, strategies focusing on improving the quality of work can ensure process improvement, unlike IA improvement policies. Nevertheless, IA improvement strategies can provide immediate means to handle rework and be less invasive than policies to increase the quality of work, as suggested in the discussion of Heuristic 1 (Section 8.2.2.1).

IA Improvement Heuristic 8:
High IA quality enables process improvement, while high quality of work guarantees process improvement.

Figure 8.16: Heuristic 8 for IA improvement

8.2.3 EXTRACTION OF HEURISTICS FOR CHANGE PACKAGING

In the adapted rework cycle simulation, change packaging influences the delay between discovering necessary design modifications and their implementation as well as includes a factor that can increase the quality of work parameter (Section 7.5). As such, change packaging can directly affect the productivity[64] and quality of work parameters, as shown in the extended, adapted rework cycle in Figure 8.2, and heuristics for change packaging, summarised in Figure 8.19, can be derived from the previous heuristics delineated for IA improvement (Section 8.2.2).

As suggested by Heuristic 3 for IA improvement (Figure 8.8), increasing the quality of work is more beneficial than improving IA quality early in design processes, and Heuristic 6 for IA improvement (Figure 8.13) indicates increasing productivity is more effective than decreasing the rework discovery delay early in design processes. Consequently, change packaging policies can have most impact on process improvement early in design processes since they can influence productivity and the quality of work. Figure 8.17 displays Heuristic 1 for change packaging. Although this heuristic suggests change packaging strategies may prove more effective than IA improvement strategies early in design processes through these heuristics for IA improvement, without the ability to implement of change packaging policies, as described occurring in the aerospace firm due to customer demands (Section 7.5.3), IA improvement may still be the better focus for process improvement.

Change Packaging Heuristic 1:
Change packaging is most beneficial early in design processes.

Figure 8.17: Heuristic 1 for change packaging

In turn, the simulation results (Section 7.5.2) suggest a trade-off in that the quick packaging of few design changes (with little quality of work improvement) can decrease the delay to implementing work and increase productivity and the rate of work done, while packaging more design modifications together slowly improves the quality of work, also increasing the rate of work completed. Although an argument for focusing on increasing the quality of work can be made based on the characteristics of the aerospace company's design process (Section 7.5.3), given

[64] For example, long delays between packaging changes can decrease productivity as less work is scheduled to be performed.

Heuristic 8 for IA improvement (Figure 8.16) that discusses the significance of improving the quality of work, focusing on improving the quality of work through change packaging strategies may be most effective for other design processes as well. Once a high quality of work is established, work is completed directly and is only limited by the staff size and their productivity. Alternatively, increasing productivity with mediocre quality of work can lead to the generation of significant rework and slow the rate of work completed. Figure 8.18 embodies this heuristic for change packaging.

Change Packaging Heuristic 2:
Packaging many changes together is more beneficial than quickly packaging smaller packages of design modifications.

Figure 8.18: Heuristic 2 for change packaging

Notably, Limitation 6 in Table 6.1 influences this second heuristic for change packaging. Specifically, if modifications occur across many decoupled design areas, such that few modifications are interdependent, change packaging may not significantly increase the quality of work parameter. As such, policies to quickly implement changes may be more beneficial in this case.

8.3 EVALUATION OF THE HEURISTICS

Evaluating the heuristics in terms of specific design processes indicates their relevance to industry practice as well as the scope for their potential application or limitations. In turn, the evaluation should include feedback from a multitude of interviewees working in a variety of industry sectors and could also encompass a comparison with other academic research. Although this reflection would not prove the validity of the heuristics, the boundaries of the heuristics could be probed. This dissertation leaves the implementation of such a thorough evaluation for future work (Section 9.3) and simply presents feedback from interviewees at the aerospace and telecommunications firms in this section. Despite this narrow evaluation performed, heuristics, nevertheless, provide a useful means to incorporate lessons from modelling and simulation into industry projects, as discussed by Lyneis and Ford (2007). Lyneis and Ford reason that research cannot definitively quantify simulation results for all projects, and creating such guidelines can enable the practical use of academic literature.

The heuristics were evaluated similarly to the IA characterisations by determining the "adequacy" of these heuristics for use in practice (Section 4.5). The same interviewees from the aerospace and telecommunications companies (I-2, I-16, I-39) reflecting on the IA characterisations discussed the heuristics in terms of the *falsehood*, *utility*, and *transferability* criteria (Section 4.5.2). In addition, these evaluators were asked to suggest their perception of the *applicability* of these heuristics in design processes. Specifically, the interviewees responded to the following questions on the heuristics (summarised in Figure 8.19):

- **Falsehood:** Do the heuristics reflect IA and change packaging improvement? If so, why do you agree? If not, what cases are not captured?
- **Utility:** Do the heuristics explain how to improve IA and change packaging? Do the heuristics provide reasonable predictions for improvement? If so, why do you agree? If not, how should they be modified?[65]
- **Transferability:** Do the heuristics correspond with other guidelines used in practice? Do other guidelines contradict the heuristics? If so, what guidelines?
- **Applicability:** For what application are these heuristics useful?

The following sections discuss the evaluation of the heuristics against each of these criteria. Interviewees' suggestions to modify the wording of the heuristics for clarity and to increase their usability are already incorporated into their presentation in Section 8.2.2 and Section 8.2.3.

8.3.1 FALSEHOOD

The interviewees from the aerospace company (I-2, I-16) agreed that the heuristics indicate means for process improvement. They both described Heuristic 4 through 8 for IA improvement as "intuitive" based on their understanding of areas for process improvement in practice and the description of the adapted rework cycle model. However, the manager (I-2) acknowledged that the first three heuristics for IA improvement (Table 8.2) were not as "instinctive" for him and stated:

> *From a system dynamics point of view, I understand that (Heuristic 1 for IA improvement), but, from a management point of view, it doesn't sound right. You always put the effort into stopping things coming around this loop (from the quality of work), rather than stopping them once they have happened (in practice). That (Heuristic 1 for IA improvement) is counter-intuitive.*

He discussed that projects tend to naturally focus on improving the quality of work parameter and later concluded: "We need to think about them (Heuristic 1 through 3 for IA improvement) a bit more. They would change our behaviour".

[65] The interpretation of utility in terms of explaining and predicting practice is derived from Bacharach (1989).

In turn, the evaluator from the telecommunications company (I-39) introduced a different perspective on the heuristics. He agreed that Heuristic 4 through 7 for IA improvement were logical and discussed how rework accumulated during projects when designers did not acknowledge the need for changes or perform IA regularly. The agile development processes prescribed particularly focus on decreasing the rework discovery time delay in order to manage such necessary changes within the company's product and service development projects. For the remaining heuristics for IA improvement, this interviewee indicated that they only apply to certain projects in the firm. He explained that the telecommunications company use to focus on finding all of the knock-on effects and errors caused by design changes within all projects, but this policy caused the late delivery of their products and services. He rationalised:

> *If you don't deliver to the shelf life, then there was no point in having the product in the first place. It's better to deliver something that works most of the time with some errors than to have nothing on the shelf at all.*

In turn, Heuristic 1 through 3 and Heuristic 8 for IA improvement do not necessarily suggest process improvement in practice if improving IA quality or the quality of work delays the development process for these time-critical projects, which can span only 3 months. The investment in IA techniques, even to identify closely related changes, may not be beneficial if the project is delayed and the opportunity to make profit passes. Nevertheless, this evaluator noted that other projects, which require robust products that contain minimal design errors, do fit these remaining heuristics for IA improvement. He said: "In traditional projects, these heuristics (Heuristics 1 through 3 and Heuristic 8) fit exactly right. Some parts of the company still work that way. I suppose it's the obvious and natural way of working". Consequently, the evaluation of the heuristics for IA improvement suggests that some only apply to design processes with the criteria of removing all errors for completion. However, in this context, the heuristics meld with the interviewees' perspectives on process improvement and are relevant to describing process improvement in practice.

All the evaluators (I-2, I-16, I-39) agreed with the change packaging heuristics. The manager from the aerospace company (I-2) stated: "These heuristics feel right". He then discussed the company's packaging policies in terms of increasing the quality of work. Similarly, the evaluator at the telecommunications firm (I-39) agreed with the change packaging heuristics and indicated that they applied across the projects at this company. He noted that change packaging is particularly critical since many of the products and services this company delivers work with and build upon each other. Without change packaging, he said: "The project becomes out of control". As such, the evaluation of the change packaging heuristics implies that these heuristics agree with industry practice.

8.3.2 UTILITY

The evaluators from the aerospace company (I-2, I-16) indicated that the heuristics for IA improvement were useful. For instance, the manager (I-2) discussed how policies to heighten the emphasis on IA during design processes would meet resistance from project managers in terms of Heuristic 1 through 3 and then stated:

> *It is a mindset thing. Their (the project managers') instinctive reaction is to go around the (rework cycle) loop as fast as you can. It's useful. The model is useful to explain this. It would be interesting to see how it modelled dynamically.*

This interviewee subsequently suggested further investigation to quantify the trade-offs presented in the heuristics for IA improvement. As such, a more predictive model, including data on all the parameters, would also be useful (Section 9.3).

Given the telecommunications evaluator's (I-39) response to the falsehood criterion questions (Section 8.3.1), he noted that Heuristic 1 through 3 and Heuristic 8 are not necessarily useful for all projects. However, this interviewee discussed how Heuristic 4 through 7 conform to process improvement in practice and are useful to note during any project. He said: "What you can't do is trick yourself into thinking you can deliver something before you really can". In turn, within the limitations of Heuristic 1 through 3 and Heuristic 8 for certain projects (Section 8.3.1), this evaluator's comments suggest the heuristics for IA improvement provide a useful means to explain and guide practice.

All evaluators (I-2, I-16, I-39) discussed that some changes in practice have high priority and are implemented quickly based on customer or stakeholder demand. Heuristic 2 for change packaging does not capture this exception that occurs in practice. As such, the interviewees proposed that they ideally strive to implement Heuristic 2, but other pressures often make it impossible. In addition, the interviewees from the aerospace company (I-2, I-16) noted that the change packaging heuristics assume that the system and software architecture is modular and changes can be packaged to improve the quality of work. In turn, for Heuristic 1 for change packaging, they discussed the need to develop a design capable of change packaging early in design processes as well. Thus, the heuristics for change packaging describe process improvement, but they do not directly discuss the exceptions and conditions required to implement change packaging in practice. The adapted rework cycle model does not capture these exceptional factors, and, hence, the heuristics derived are useful to describe ideal practice as opposed to offering suggestions on how to address these other influences.

8.3.3 TRANSFERABILITY

At the aerospace company, the process engineer (I-16) stated that he encouraged project managers to improve IA practice, similar to Heuristic 1 through 3 for IA improvement, using his guideline: "Pay now, or pay later". Project managers should admit rework exists and put the effort into finding design errors sooner rather than later. In essence, these heuristics derived from the adapted rework cycle formalise this guideline. This interviewee also suggested that the design processes delineated and prescribed by the aerospace company generally encourage the other heuristics for IA improvement. The evaluator at the telecommunications company (I-39) similarly indicated that the heuristics for IA improvement conform to the prescribed processes and tools implemented within more "traditional" projects in this firm (Section 8.3.1). As such, the heuristics for IA improvement formalise implicit guidelines used as a basis for the prescribed processes in the aerospace and telecommunication companies.

The interviewees from the aerospace company (I-2, I-16) discussed the firm's initiatives to further modularise the product platform and viewed Heuristic 1 for change packaging as transferable to these strategies. Notably, these evaluators did not mention that change packaging is not typically enforced during the informal change processes and only occurs during formal change processes (Section 3.4.2). As such, early change packaging may not occur rigorously in practice. Thus, the practised processes in the aerospace company may counter Heuristic 1 for change packaging, even though the interviewees confirmed this heuristic as valid by the falsehood and utility criteria (Section 8.3.1 and Section 8.3.2, respectively). The evaluator at the telecommunications company (I-39) indicated that Heuristic 1 is reflected by company procedures. In turn, Heuristic 2 for change packaging follows the change packaging policies in place in both companies. However, as noted in Section 8.3.2, exceptions to this policy occur in practice. Hence, this evaluation suggests that the change packaging heuristics generally conform to prescribed practice, but also highlights a potential area for modification to the aerospace company's informal change processes implemented.

8.3.4 APPLICABILITY

The evaluators at the aerospace company (I-2, I-16) envisioned several applications for the IA improvement and change packaging heuristics. The manager (I-2) suggested that management practices across mechanical and hardware design teams should be investigated in terms of the IA improvement heuristics. He stated: "I am wondering what it means at a broader project-level. I don't see very much IA at this point (outside of the control system design group)". This application of these heuristics could highlight areas for more broadly improving change management. Similarly, the design process engineer (I-16) commented that the adapted rework cycle model and these heuristics also encourage the active tracking of changes in the mechanical and hardware design teams (Section 4.5.2.3). This evaluator also noted that the change packaging heuristics could be useful for hardware design and cited a specific project in which many hardware changes were not packaged, causing much rework.

In turn, at the telecommunications company, the design process engineer (I-39) stressed that Heuristic 1 through 3 and Heuristic 8 for IA improvement are only applicable for design processes with the completion criterion of removing design errors (Section 8.3.1). He described how some of the design processes implemented emphasised stopping when the product was "good enough". However, he suggested that all of the heuristics for IA improvement and change packaging are more applicable at a higher level across projects since the platforms developed for products and services delivered should be stable in the long run. If products must work together, then performing IA to ensure their integration becomes vital.

Thus, the IA improvement and change packaging heuristics potentially have broader applications than to the system and software design changes of the aerospace firm investigated through simulation.

8.4 SUMMARY

Using the extended rework cycle model developed by Lyneis *et al.* (2001) as a basis, which depicts the trade-offs between strategies for process improvement, the adapted rework cycle model is similarly extended. Heuristics for IA improvement and change packaging (Figure 8.19), suggesting when these strategies are beneficial in comparison to other process improvement strategies, are derived by interpreting the simulation results from Chapter 7 in terms of this model and also qualitatively analysing the trade-offs between model parameters.

IA Improvement Heuristic	States:
1	If the amount of rework is much larger than the amount of work to do, identifying the impact of closely related knock-on changes is more beneficial than decreasing the rework generated.
2	If the amount of work to do is much larger than the amount of rework, decreasing the rework generated is more beneficial than identifying the impact of closely related knock-on changes.
3	Identifying the impact of closely related knock-on changes becomes more beneficial than decreasing the rework generated as design processes progress.
4	If the amount of rework is much larger than the amount of work to do, identifying the impact of changes quickly is more beneficial than implementing work quickly.
5	If the amount of work to do is much larger than the amount of rework, implementing work quickly is more beneficial than identifying the impact of changes quickly.
6	Identifying the impact of changes quickly becomes more beneficial than implementing work quickly as design processes progress.
7	Improving very low IA quality is more beneficial than increasing staff size or productivity.
8	High IA quality enables process improvement, while high quality of work guarantees process improvement.

Change Packaging Heuristic	States:
1	Change packaging is most beneficial early in design processes.
2	Packaging many changes together is more beneficial than quickly packaging smaller packages of design modifications.

Figure 8.19: Summary of heuristics for IA improvement and change packaging

These heuristics indicate that IA improvement strategies can contribute to process improvement more effectively than other process improvement strategies. For example:

- From Heuristic 1 for IA improvement, improving IA quality through strategies simply allocating more resources and time for IA can be more beneficial than more invasive strategies, such as reforming requirement management practices.
- From Heuristic 4 for IA improvement, IA improvement strategies focusing on reducing the rework discovery time delay can be more effective than improving productivity through policies to increase hiring or overtime.

Given that the conditions of these two heuristics are relevant to the characteristics of the aerospace company's design process, strategies to improve IA should be considered at the same level as other process improvement strategies. However, as indicated by an evaluator, improving IA is not "instinctive" for project managers at the aerospace firm. As such, the modelling and simulation of IA through the adapted rework cycle used to develop the heuristics provides justification and explains the heightened importance of IA improvement, as discussed in the evaluation of the heuristics. The grounding of the investigation of process improvement strategies in the adapted rework cycle provides a useful means to systematically weigh the trade-offs in policies. Although limitations to the adapted rework cycle and, in turn, heuristics exist, as noted in the evaluation, these heuristics also can guide the selection of process improvement strategies for other design processes besides that of the aerospace company.

9 :: CONCLUSION

This research project has investigated IA through literature, empirical studies, and the modelling and simulation of the adapted rework cycle. Based on this examination of IA as a distinct element of design processes, this dissertation addresses the research questions posed and contributes to understanding change management. The following sections highlight the key conclusions (Section 9.1) and contributions of this dissertation (Section 9.2) and summarise the future work stemming from this research project (Section 9.3).

9.1 KEY CONCLUSIONS STEMMING FROM THE RESEARCH QUESTIONS

The research questions summarised in Table 9.1 and their responses (Section 5.5 and Section 8.1) illuminate key conclusions from this investigation of IA.

Table 9.1: Summary of research questions

Research Question	States:
1	How are change processes and impact analysis prescribed at the system and software engineering interface within industry?
2	Does impact analysis influence the management of emergent changes in practice? If so, how?
3	What are the challenges in using impact analysis to manage emergent changes at the system and software engineering interface?
4a	Can the application of impact analysis improve the design process? If so, how?
4b	Can modelling and simulation demonstrate the effects of impact analysis improvement on the design process? If so, how?
4c	Does change packaging affect impact analysis improvement? If so, how?

For research question 1, the empirical studies indicate that the aerospace and telecommunication companies studied implement formal and informal change processes. In either of these change process performed, processes to apply IA are often not explicitly defined in practice, and a practical IA improvement strategy targets defining methods to implement IA. As a result, two key conclusions follow:

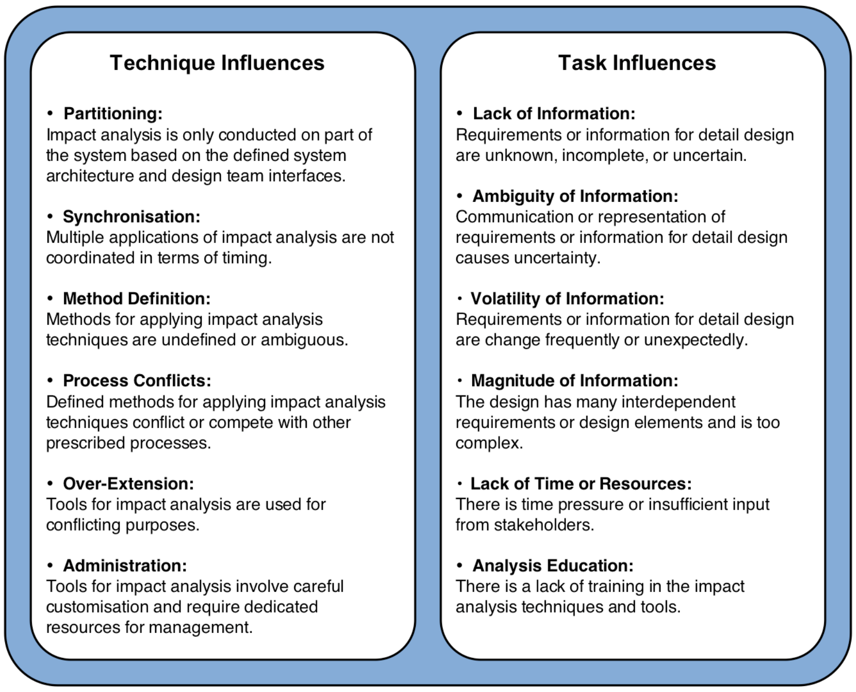
- Prescribed IA techniques are not always implemented in practice for some industry sectors.
- Defining processes to implement IA can support the application of IA in practice.

For research question 2, the empirical studies also suggest that IA is viewed as a key means to manage emergent changes in practice. Without thorough IA results, design modifications can lead to unexpected, emergent changes. In turn, the definition of key concepts, including IA *techniques*, *tasks*, *quality*, and *rigour*, describe how the application and results of IA can vary in practice due to IA *influences*, illustrated in Figure 9.1. This research question leads to three key conclusions:

- Without high-quality IA results, emergent changes are missed, leading to rework, schedule delays, and unexpected costs. As such, commitment to performing IA is a prerequisite to manage the emergence of unanticipated design modifications. Investment in new techniques and tools is not necessarily required as rigorous IA can equally occur through traceability, dependency, or experiential IA, and any combination thereof by managing IA influences.
- The empirical studies at the aerospace company, which operates a highly iterative design process, indicate that the partitioning, synchronisation, and information-related influences most commonly affect IA results (Figure 9.1). In fact, 95% of design modifications elicited across the systems-software interface as well as the interface with external stakeholders involved at least one of these influences (Section 5.2.2.4).
- Addressing such influences can improve the rigour of IA to examine first-order, knock-on design modifications, increasing the quality of IA results and, consequently, significantly improving the design process; as shown by the simulation results in Section 7.4, marginal IA improvement can cause about a 60% decrease in the time to complete design work (Figure 7.16). Rigorously analysing higher-order, knock-on effects to increase IA result quality can require more time and resources and have diminishing returns on process improvement.

For research question 3, the IA influences characterisation depicts the challenges to implementing IA in practice. These influences can decrease the rigour and quality of IA results obtained. However, as suggested by the empirical studies, focusing on dominant influences can lead to more practical process improvement strategies than attempting to mitigate all influences equally. Hence, the conclusion is:

- Targeting specific IA influences to address through IA improvement strategies is a practical means for IA improvement.



Technique Influences

- **Partitioning:**
Impact analysis is only conducted on part of the system based on the defined system architecture and design team interfaces.
- **Synchronisation:**
Multiple applications of impact analysis are not coordinated in terms of timing.
- **Method Definition:**
Methods for applying impact analysis techniques are undefined or ambiguous.
- **Process Conflicts:**
Defined methods for applying impact analysis techniques conflict or compete with other prescribed processes.
- **Over-Extension:**
Tools for impact analysis are used for conflicting purposes.
- **Administration:**
Tools for impact analysis involve careful customisation and require dedicated resources for management.

Task Influences

- **Lack of Information:**
Requirements or information for detail design are unknown, incomplete, or uncertain.
- **Ambiguity of Information:**
Communication or representation of requirements or information for detail design causes uncertainty.
- **Volatility of Information:**
Requirements or information for detail design are change frequently or unexpectedly.
- **Magnitude of Information:**
The design has many interdependent requirements or design elements and is too complex.
- **Lack of Time or Resources:**
There is time pressure or insufficient input from stakeholders.
- **Analysis Education:**
There is a lack of training in the impact analysis techniques and tools.

Figure 9.1: Summary of IA influences (from Section 4.4)

For research question 4a, the modelling and simulation of the adapted rework cycle indicate that effective IA can improve design processes aiming to deliver error-free products and services. The adapted rework cycle model also provides the basis to derive heuristics, describing how IA and other improvement strategies can relatively affect design processes, through the investigation of trade-offs between its parameters. In turn, based on the heuristics for IA improvement derived and their evaluation, two key conclusions follow:

- Strategies to improve IA, which may be less "instinctive" to implement according to an evaluator, can deliver equivalent, if not more effective, process improvement as other strategies, such as optimisation of design task scheduling or requirements management. This assessment is also evinced in the simulation of the aerospace company's design process in that the normalised sensitivity coefficient to improve IA is twice that of the coefficient to decrease the number of changes generated (*i.e.* through implementing other such strategies) (Section 7.3.3).

- The importance of implementing IA should be elevated within design processes and can be explained through the adapted rework cycle.

For research question 4b, the modelling and simulation of the adapted rework cycle provide a means to investigate IA improvement on the overall design process. By using a system dynamics methodology for modelling and simulation, strategies for IA improvement have been investigated through analysing the trade-offs between parameters. In turn, the model simulated demonstrates the impact of improving IA in terms of design process duration, and these results conform to expert evaluation. As such, another key conclusion arises:

- The modelling and simulation of the adapted rework cycle demonstrates and predicts the significant influence of IA improvement on design processes.

For research question 4c, the modelling of change packaging in the adapted rework cycle model and the evaluation of this model show that change packaging can enhance IA improvement strategies. In turn, the heuristics for change packaging describe the ideal implementation of change packaging within design processes, and the following conclusion stems from the analysis of change packaging:

- Introducing change packaging to design processes increases the quality of work being performed and does not detrimentally affect other aspects of the design process. In other words, change packaging strategies provide an opportunity for improvement at minimal risk.

In conclusion, performing IA can lead to design process improvement by enabling the appropriate scoping of changes prior to implementation (*i.e.* before accepting a change request) and allowing for the thorough analysis of modifications during implementation. For products that develop through evolution and rework, historical data and knowledge about design area volatility from previous design iterations can inform the IA implemented during on-going design processes. Past traceability, dependency, and experiential IA results and the actual changes made can be compared and used to inform future assessments. As such, the means for collecting and analysing data on changes should also be examined with regards to enabling IA improvement, as discussed in future work (Section 9.3).

9.2 CONTRIBUTIONS

Through addressing the research questions posed, this research project makes contributions to the field of change management, which can be pinpointed to specific sections of this dissertation as follows:

- *The holistic classification of IA (Section 2.5).* This work explicitly identifies experiential IA as a means to assess the scope of design modifications. This novel interpretation of IA acknowledges the frequent practice of scoping changes through inspection or engineering judgement within industry and melds this practice with the definition of IA types.
- *The scarcities and disparities in literature on IA (Section 2.7).* The literature review assesses IA across a broad range of research areas, including systems engineering, software engineering, requirements engineering, mechatronics, control system design, engineering design, change management, configuration management, and concurrent engineering. Analysis of this breadth of literature highlights previously unidentified gaps in research, which the empirical studies address.
- *The empirical studies on IA* (Section 3.4 and Section 3.5). Describing the results of empirical studies adds to the body of knowledge of change management practice.
- *The IA characterisations* (Section 4.1, Section 4.2, Section 4.3, and Section 4.4). The IA characterisations delineate the disparity between prescribed and practised IA and comprehensively specify the influences to this difference.
- *The elicitation method for IA practice (Section 5.1).* The IA elicitation method provides a means to systematically examine IA practice. The application of this method at the aerospace company shows how to elicit the IA techniques used, the quality of IA, and the influences to the IA applied. The analysis of such results can unearth specific areas for improving design processes.
- *The adapted rework cycle model* (Section 6.1 and Section 6.3). The adapted rework cycle provides a novel means to conceptualise and investigate IA and change packaging within design processes. Other models found to date focus on specific types of IA and do not incorporate the packaging of changes in processes involving rework.
- *The heuristics for IA improvement and change packaging (Section 8.2).* The heuristics derived from the adapted rework cycle explain and specify the implications of IA improvement and change packaging on design processes, which previously only were implicitly acknowledged by the industry collaborators.

9.3 FUTURE WORK

Despite making these contributions to change management research, limitations or areas for improvement in this examination of IA are highlighted throughout this dissertation. Future work can address these points, strengthening the understanding of the implications of IA improvement. Accordingly, the envisioned future work entails:

- As discussed in the evaluation of the IA characterisations (Section 4.5.2.2), further empirical studies determining the IA techniques that produce high-quality IA results for specific change types or product design areas could lead to the development of useful IA guidelines.
- As mentioned in the elicitation of IA practice (Section 5.2.1), the information and attribute ratings collected from the interviews for the 42 modifications discussed could be correlated with the completed change database for this project. In turn, the estimations of IA quality given by designers could be evaluated and inputted into the simulation (Section 7.2).
- As suggested in the description of the adapted rework cycle model (Section 6.3.2 and Section 6.3.3), the adapted rework cycle could be extended to include a product model to allow the coupling of change sets to be modelled. This addition would enhance the investigation of IA quality improvement and change packaging. The system dynamics professor, who evaluated the adapted rework cycle model, also particularly encouraged this addition to the adapted rework cycle (Section 6.3.5).
- As addressed in the limitations of the adapted rework cycle (Section 6.3.4), this model, as simulated for the aerospace company, does not factor in "added" or "obsoleted" work (Figure 6.14) and could be extended to include these elements. Since IA also occurs prior to accepting additional proposed work or modifications (as shown by the change process in Figure 2.10), IA should correspondingly be included when modelling "added" work. As such, the refined model in Figure 9.2 distinguishes between proposed, rejected, and accepted or added work. This model shows that IA performed on new, proposed work occurs at a certain rate based on IA productivity, or how often IA is applied on proposed work (*e.g.* the frequency of meetings with customers to discuss new requirements). IA quality, as defined in Section 6.3.2, can also affect the rate of proposed work analysed in that more in-depth, higher-quality IA can require more resources, reducing this rate. Notably, IA quality for proposed work may differ from the IA quality value for the rework discovery rate since higher IA quality may only be given priority while implementing changes, as observed in the aerospace company (Section 3.4.2.1), and the investigation of higher-order, knock-on effects may also differ in these instances. Nevertheless, high IA quality

prior to adding work to projects can allow for improved scoping of the amount of work to do and, consequently, the time and budget required. Future work should further evaluate the model shown in Figure 9.2 and investigate the implications on the heuristics derived in this dissertation.

- As indicated by the aerospace evaluators of the IA improvement heuristics (Section 8.3.2), implementing the adapted rework cycle as a predictive model would be useful to improve project management and policies. In turn, modelling the work to do, people, and productivity parameters is necessary by collecting additional empirical data, addressing the limitations of the simulation implementation cited in Section 6.3.4 and Section 6.3.5.

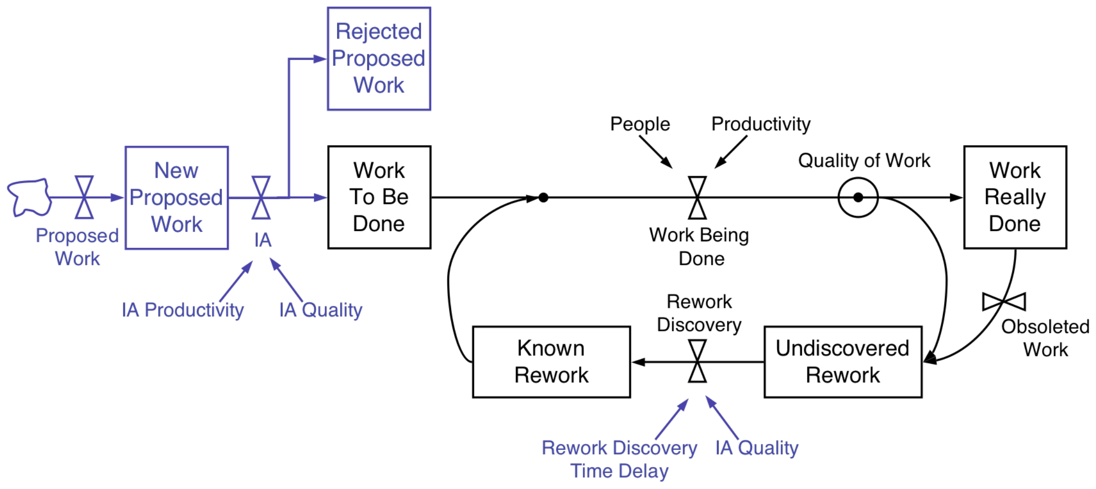


Figure 9.2: The adapted rework cycle with the addition and obsolescence of work (from Figure 6.14)

In turn, additional future work supports these aims:

- The evaluators at the aerospace company (Section 8.3) described the need to develop a strategy to promote the importance of IA to project managers. Without first understanding and addressing their reservations, implementing IA improvement strategies is impeded. Such a strategy could involve workshops and publications, such as pamphlets or workbooks.

- These evaluators also suggested the need to improve the change information captured within the system, software, mechanical, and hardware engineering groups. Investigating what information and how such information is documented could provide for improvements to the predictive modelling of the adapted rework cycle. Specifically, capturing the relationships between initiating and knock-on modification in change sets, also called parent-child relationships by Giffin *et al.* (2007), can support such prediction through improved analysis of empirical data to calibrate simulations.

Furthermore, this research project could be extended:

- The heuristics for IA improvement and change packaging could be evaluated across additional companies and industry sectors developing software. This extension would provide a better understanding of their applicability and limitations to a wider range of system-software design processes.
- The heuristics developed also could be evaluated for mechanical and hardware design processes as well as for the project management of mechatronic product development. This examination would indicate the generality of the adapted rework cycle to design processes.
- Finally, the adapted rework cycle could be incorporated into other system dynamics and software process dynamics models to determine if and how including IA affects their results and policy interpretations.

9.4 SUMMARY

The scoping of changes by engineers through IA techniques and tasks determines the emergence of unanticipated design errors and modifications. With poor quality IA results, software-intensive products can unexpectedly fail or development processes can incur additional costs and project delays. Although no panacea to effective change management may exist, this dissertation demonstrates how improving IA can improve system engineering and software design processes and provides direction to this end by stipulating the challenges of implementing IA in practice. Treating IA as a crucial part of change management is an effective means to improve design processes.

APPENDIX A

Table A identifies the interviews conducted for this research project (Section 3.1). Key discussions, which could not be audio recorded and were only memoed, are also included in this list, including a meeting with multiple managers of the aerospace company (I-23) and the interviews and workshop at the telecommunications company. At the workshop at the telecommunications company, systems, software, and process engineers as well as project managers were present.

This list also includes interviews external to the empirical studies that were conducted at three other companies (a large computer hardware and software development firm, a small company focusing on research and development for computer hardware and software, and a medium-size consultancy working within telecommunications) for general research feedback, advice, and input.

Table A: Interviews conducted within this research project

ID	Position	Company	Date(s)
1	Manager	Computer Hardware and Software Firm	3-Jan-05
			13-Feb-05
			12-Aug-05
			25-Aug-05
2	Manager	Aerospace Firm	30-Aug-05
			28-Apr-06
			3-Dec-07
3	Software Engineer	Aerospace Firm	13-Oct-05
			7-Nov-06
4	Systems Engineer	Aerospace Firm	13-Oct-05
			6-Nov-06
5	Controls Hardware Engineer	Aerospace Firm	17-Oct-05
6	Controls Conceptual Design	Aerospace Firm	18-Oct-05
7	Controls Hardware Engineer	Aerospace Firm	19-Oct-05
8	Software Engineer	Aerospace Firm	19-Oct-05
9	Controls Hardware Engineer	Aerospace Firm	20-Oct-05
10	Mechanical Engineer	Aerospace Firm	25-Oct-05
11	Mechanical Engineer	Aerospace Firm	26-Oct-05
12	Process Engineer	Aerospace Firm	28-Oct-05
13	Systems Engineer	Aerospace Firm	2-Nov-05
			7-Nov-06
14	Systems Engineer	Aerospace Firm	2-Nov-05
			6-Nov-06
15	Systems Engineer	Aerospace Firm	2-Nov-05
			8-Nov-06

ID	Position	Company	Date(s)
16	Process Engineer	Aerospace Firm	16-Nov-05
			4-Dec-06
			3-Dec-07
17	Systems Engineer	Aerospace Firm	16-Nov-05
18	Manager	Aerospace Firm	17-Nov-05
19	Systems Engineer	Aerospace Firm	17-Nov-05
			9-Nov-06
20	Process Engineer	Aerospace Firm	18-Nov-05
21	Systems Engineer	Aerospace Firm	18-Nov-05
22	Systems Engineer	Aerospace Firm	21-Nov-05
23	Managers	Aerospace Firm	9-Dec-05
24	Manager	Aerospace Firm	12-Oct-06
25	Software Engineer	Aerospace Firm	6-Nov-06
26	Systems Engineer	Aerospace Firm	6-Nov-06
27	Software Engineer	Aerospace Firm	6-Nov-06
28	Software Engineer	Aerospace Firm	7-Nov-06
29	Software Engineer	Aerospace Firm	8-Nov-06
30	Systems Engineer	Aerospace Firm	8-Nov-06
31	Software Engineer	Aerospace Firm	8-Nov-06
32	Software Engineer	Aerospace Firm	9-Nov-06
33	Social Scientist	Research and Development Firm	25-Jan-06
34	Software Engineer	Research and Development Firm	25-Jan-06
35	Manager	Research and Development Firm	25-Jan-06
36	Process Engineer	Telecom Consultancy	15-Nov-06
37	Systems Engineer	Telecom Firm	9-May-06
38	Software Engineer	Telecom Firm	9-May-06
39	Process Engineer	Telecom Firm	10-May-06
			8-Aug-06
			5-Dec-07
40	Workshop	Telecom Firm	24-Oct-06
			25-Oct-06
			26-Oct-06
41	Process Engineer	Telecom Firm	24-Oct-06
42	Process Engineer	Telecom Firm	8-Aug-06
43	Process Engineer	Telecom Firm	8-Aug-06
44	Manager	Telecom Firm	31-May-07

APPENDIX B

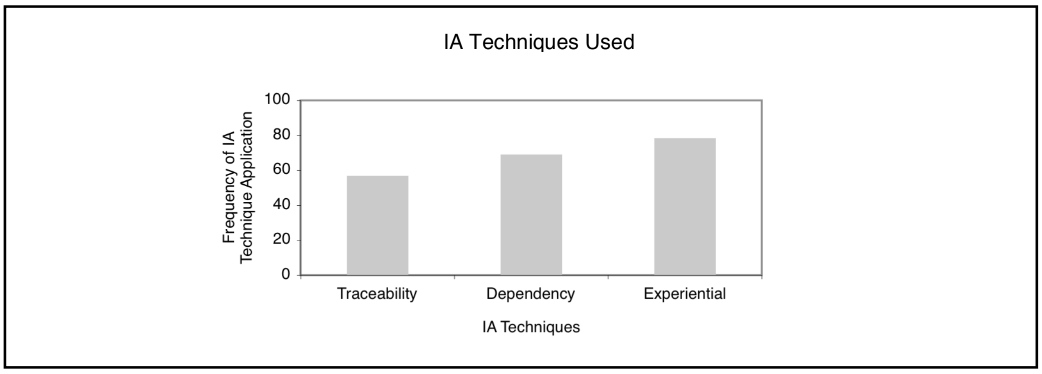
Figure A displays a matrix of the change attributes (Section 5.1.2) and IA (Table 3.1) for the 42 changes elicited during the aerospace IA empirical study (Section 5.2.1). Several engineers also used alternative IA techniques, which they developed individually to suit particular needs, as shown in Figure A. For example, informal documentation stored large volumes of data inputted into the software design, and additional software models were implemented through computer spreadsheets. These additional forms of documentation and other models were also used to perform IA and were updated to reflect design changes.

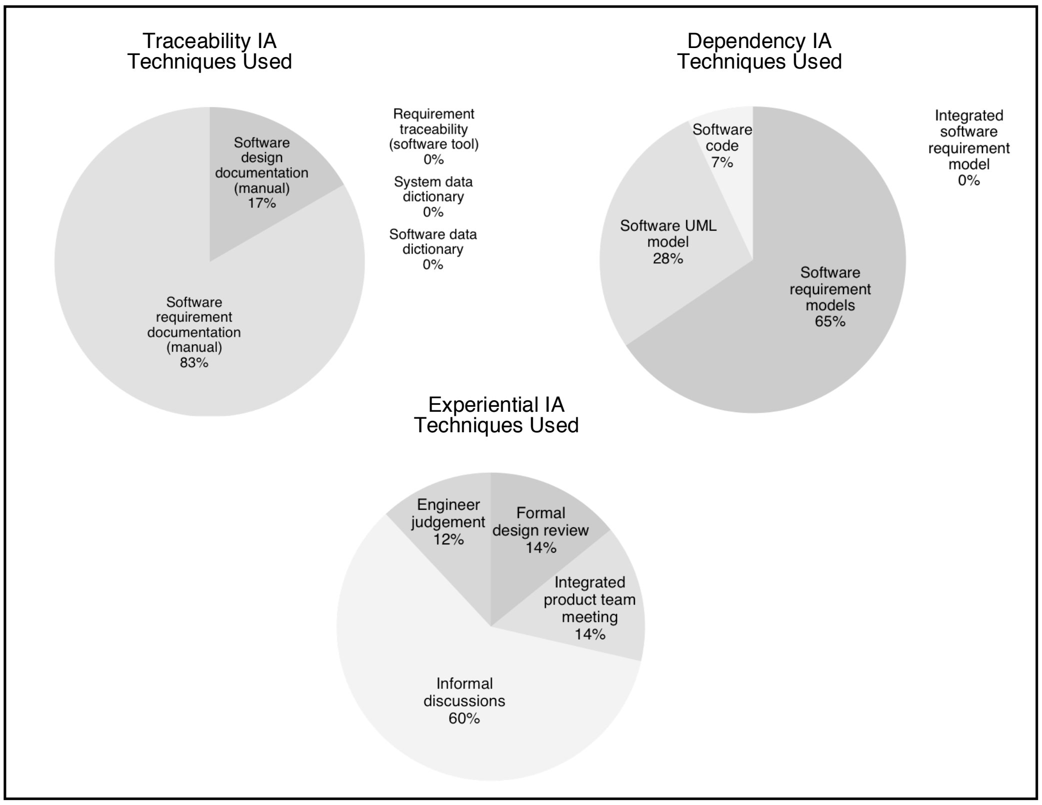
Change ID	C-1	C-2	C-3	C-4	C-5	C-6	C-7	C-8	C-9	C-10	C-11	C-12	C-13	C-14	C-15	C-16	C-17	C-18	C-19	C-20	C-21
Interviewee ID	I-3	I-3	I-3	I-4	I-4	I-14	I-14	I-14	I-14	I-14	I-14	I-15	I-15	I-15	I-19	I-19	I-19	I-25	I-25	I-25	I-26
Change Source	Software design	System design	Software design	External stakeholder	External stakeholder	External stakeholder	External stakeholder	External stakeholder	Software design	Software design	External stakeholder	External stakeholder	External stakeholder	Software verification	External stakeholder	External stakeholder	Software verification	System design	External stakeholder	Software design	External stakeholder
Change Implementation Level	Software design	Software design	Software design	System design	System design	System design	System design	System design	System design	System design	System design	System design	System design	System design	System design	System design	System design	Software design	Software design	Software design	System design
Traceability IA: Requirement traceability (software tool)																					
Traceability IA: Software requirement documentation (manual)				•		•	•	•					•					•			•
Traceability IA: Software specification/design documentation (manual)	•	•	•																		
Traceability IA: System data dictionary																					
Traceability IA: Software data dictionary																					
Dependency IA: Software requirement models				•		•	•	•				•	•					•			•
Dependency IA: Integrated software requirement model																					
Dependency IA: Software UML model	•																			•	
Dependency IA: Software code																					
Experiential IA: Formal design review			•						•			•					•	•			
Experiential IA: Integrated product team meetings				•														•			•
Experiential IA: Informal discussions		•			•		•		•	•	•		•	•	•	•	•	•			•
Experiential IA: Engineering judgement					•	•		•											•		
IA: Using other documentation or model												•	•				•				
Traceability	•	•	•	•		•	•	•					•					•			•
Dependency	•			•		•	•	•				•	•					•		•	•
Experiential		•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•		•
Source	L	M	L	H	H	H	H	H	L	L	H	H	H	L	H	H	L	M	H	L	H
Direction	L	L	L	M	M	M	M	M	M	M	M	M	M	M	M	M	M	L	L	L	M
Level	L	L	L	M	M	M	M	M	M	M	L	M	M	M	M	M	M	L	L	L	M
Formality	M	H	M	H	L	M	M	H	M	H	L	M	M	M	L	L	L	H	L	M	M
Timing	M	H	M	H	L	M	M	M	L	M	L	M	L	L	L	L	L	M	L	M	M

Change ID	C-22	C-23	C-24	C-25	C-26	C-27	C-28	C-29	C-30	C-31	C-32	C-33	C-34	C-35	C-36	C-37	C-38	C-39	C-40	C-41	C-42
Interviewee ID	I-26	I-26	I-27	I-27	I-27	I-27	I-27	I-28	I-28	I-28	I-29	I-29	I-29	I-30	I-30	I-30	I-30	I-31	I-31	I-32	I-32
Change Source	External stakeholder	External stakeholder	System design	System design	System design	System design	Software design	System design	Software design	Software design	Software design	System design	Software design	System design	External stakeholder	External stakeholder	External stakeholder	System design	Software design	Software design	System design
Change Implementation Level	System design	System design	Software design	Software design	Software design	Software design	Software design	Software design	Software design	Software design	Software design	Software design	Software design	System design	System design	System design	System design	Software design	Software design	Software design	Software design
Traceability IA: Requirement traceability (software tool)																					
Traceability IA: Software requirement documentation (manual)	•	•	•	•	•	•	•	•				•		•	•	•	•				
Traceability IA: Software specification/design documentation (manual)										•											
Traceability IA: System data dictionary																					
Traceability IA: Software data dictionary																					
Dependency IA: Software requirement models	•	•	•	•	•	•	•	•								•	•				•
Dependency IA: Integrated software requirement model																					
Dependency IA: Software UML model											•	•	•					•	•	•	
Dependency IA: Software code									•	•											
Experiential IA: Formal design review								•													
Experiential IA: Integrated product team meetings														•	•		•				
Experiential IA: Informal discussions	•	•	•	•	•	•		•						•	•	•				•	•
Experiential IA: Engineering judgement							•														
IA: Using other documentation or model												•		•	•					•	
Traceability	•	•	•	•	•	•	•	•		•		•		•	•	•	•				
Dependency	•	•	•	•	•	•	•	•	•	•	•	•	•			•	•	•	•	•	•
Experiential	•	•	•	•	•	•	•	•						•	•	•	•			•	•
Source	H	H	M	M	M	M	L	M	L	L	L	M	L	M	H	H	H	M	L	L	M
Direction	M	M	L	L	L	L	L	L	L	L	L	L	L	M	M	M	M	L	L	L	L
Level	M	M	L	L	L	L	L	L	L	L	L	L	L	M	M	M	M	L	L	L	L
Formality	M	M	H	H	H	H	L	M	M	M	M	M	M	L	M	M	H	M	H	M	M
Timing	M	M	M	M	M	M	L	M	M	M	M	M	M	L	L	H	H	M	H	H	L

Figure A: Change attribute and IA data collected during the aerospace IA study

Figure B summarises the information in Figure A and correlates with Table 5.1.





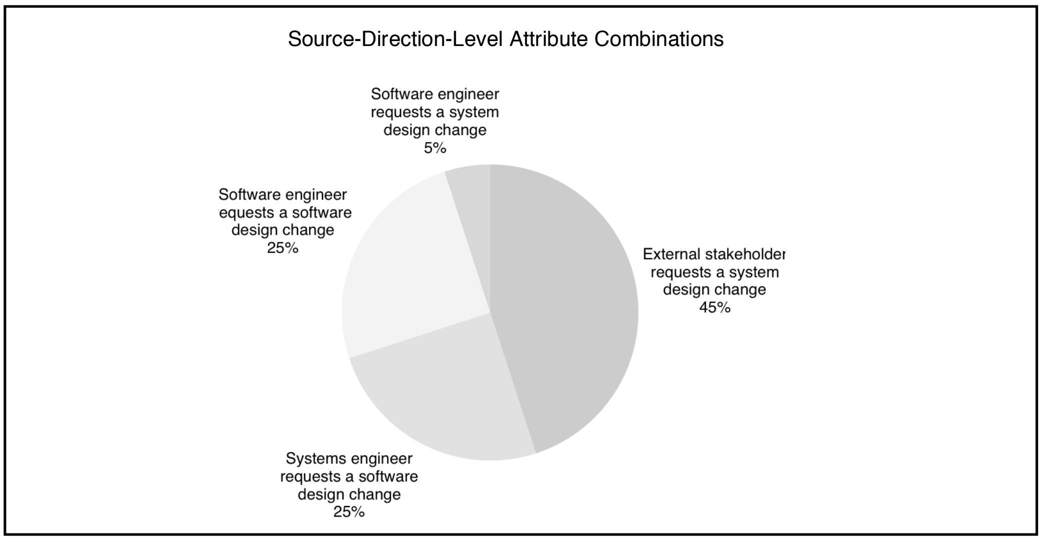


Figure B: Summary of change attribute and IA data collected

Figure C shows the changes and IA elicited from systems engineers (Figure A), grouped according to the attribute ratings, and correlates with Figure 5.8.

Changes from External Stakeholders				
I-4	C-4	Traceability	Dependency	Experiential
I-4	C-5			Experiential
I-14	C-6	Traceability	Dependency	Experiential
I-14	C-7	Traceability	Dependency	Experiential
I-14	C-8	Traceability	Dependency	Experiential
I-15	C-12		Dependency	Experiential
I-15	C-13	Traceability	Dependency	Experiential
I-19	C-15			Experiential
I-19	C-16			Experiential
I-26	C-21	Traceability	Dependency	Experiential
I-26	C-22	Traceability	Dependency	Experiential
I-26	C-23	Traceability	Dependency	Experiential
I-30	C-35	Traceability		Experiential
I-30	C-36	Traceability		Experiential
I-30	C-37	Traceability	Dependency	Experiential
I-30	C-38	Traceability	Dependency	Experiential

Changes from Software Designers				
I-14	C-9			Experiential
I-14	C-10			Experiential
I-14	C-11			Experiential
I-15	C-14			Experiential
I-19	C-17			Experiential

More Formal and More Synchronous Changes				
I-4	C-4	Traceability	Dependency	Experiential
I-14	C-6	Traceability	Dependency	Experiential
I-14	C-7	Traceability	Dependency	Experiential
I-14	C-8	Traceability	Dependency	Experiential
I-15	C-12		Dependency	Experiential
I-26	C-21	Traceability	Dependency	Experiential
I-26	C-22	Traceability	Dependency	Experiential
I-26	C-23	Traceability	Dependency	Experiential
I-30	C-37	Traceability	Dependency	Experiential
I-30	C-38	Traceability	Dependency	Experiential

Less Formal and Less Synchronous Changes				
I-4	C-5			Experiential
I-14	C-9			Experiential
I-14	C-10			Experiential
I-14	C-11			Experiential
I-15	C-13	Traceability	Dependency	Experiential
I-15	C-14			Experiential
I-19	C-15			Experiential
I-19	C-16			Experiential
I-19	C-17			Experiential
I-30	C-35	Traceability		Experiential
I-30	C-36	Traceability		Experiential

Figure C: IA practised by systems engineers according to attribute ratings

Figure D shows the changes and IA elicited from software engineers (Figure A), grouped according to the attribute ratings, and correlates with Figure 5.10.

Changes from System Designers					**Changes from Software Designers**				
I-3	C-2	Traceability		Experiential	I-3	C-1	Traceability	Dependency	
I-25	C-18	Traceability	Dependency	Experiential	I-3	C-3	Traceability		Experiential
I-25	C-19			Experiential	I-25	C-20		Dependency	
I-27	C-24	Traceability	Dependency	Experiential	I-27	C-28	Traceability	Dependency	Experiential
I-27	C-25	Traceability	Dependency	Experiential	I-28	C-30		Dependency	
I-27	C-26	Traceability	Dependency	Experiential	I-28	C-31	Traceability	Dependency	
I-27	C-27	Traceability	Dependency	Experiential	I-29	C-32		Dependency	
I-28	C-29	Traceability	Dependency	Experiential	I-29	C-34		Dependency	
I-29	C-33	Traceability	Dependency		I-31	C-40		Dependency	
I-31	C-39		Dependency		I-32	C-41		Dependency	Experiential
I-32	C-42		Dependency	Experiential					

More Formal and More Synchronous Changes					**Less Formal and Less Synchronous Changes**				
I-3	C-1	Traceability	Dependency		I-25	C-19			Experiential
I-3	C-2	Traceability		Experiential	I-27	C-28	Traceability	Dependency	Experiential
I-3	C-3	Traceability		Experiential	I-32	C-42		Dependency	Experiential
I-25	C-18	Traceability	Dependency	Experiential					
I-25	C-20		Dependency						
I-27	C-24	Traceability	Dependency	Experiential					
I-27	C-25	Traceability	Dependency	Experiential					
I-27	C-26	Traceability	Dependency	Experiential					
I-27	C-27	Traceability	Dependency	Experiential					
I-28	C-29	Traceability	Dependency	Experiential					
I-28	C-30		Dependency						
I-28	C-31	Traceability	Dependency						
I-29	C-32		Dependency						
I-29	C-33	Traceability	Dependency						
I-29	C-34		Dependency						
I-31	C-39		Dependency						
I-31	C-40		Dependency						
I-32	C-41		Dependency	Experiential					

Figure D: IA practised by software engineers according to attribute ratings

Figure E displays the ratings of information, resource, and time availability as well as IA quality for the 23 changes discussed in the IA empirical studies (Section 5.2.1). These ratings are displayed versus the IA techniques implemented, change attributes, and IA influences and are colour coded according to the IA technique or task influences cited (white and grey, respectively). The notes on the IA influences summarise the primary influences cited by designers during the interviews. These notes indicate the outcome of mitigating these influences according to hypothetical outcomes of increasing the information, resource, and time availability. In a few cases, designers suggested such mitigation strategies that actually occurred in practice, and these are stated as well in the notes.

Change ID	Interviewee ID	Change Source	Change Implementation Level	Information Availability	Resource Availablity	Time Availability	IA Result Quality	Traceability	Dependency	Experiential	Source	Direction	Level	Formality	Timing	Notes on IA Influences
C-2	I-3	System design	Software design	M	H	H	H	•		•	M	L	L	H	H	**IA Technique:** Method definition
C-43	I-3	System design	Software design	H	H	H	H	•	•	•	M	L	L	H	H	**IA Technique Mitigatation:** Clear method definition
C-44	I-4	External stakeholder	System design	L	L	H	L	•	•	•	H	M	M	H	H	**IA Task:** Lack of information, Lack of resources
C-44	I-4	External stakeholder	System design	H	H	H	H	•	•	•	H	M	M	H	H	**IA Task Mitigation:** Should improve both information and resources
C-44	I-4	External stakeholder	System design	L	H	H	M	•	•	•	H	M	M	H	H	**IA Task Mitigation:** Improving either information and resources should help
C-44	I-4	External stakeholder	System design	H	L	H	M	•	•	•	H	M	M	H	H	**IA Task Mitigation:** Improving either information and resources should help
C-45	I-4	External stakeholder	System design	M	H	M	M			•	H	M	M	H	H	**IA Task:** Lack of information, Lack of time
C-46	I-4	External stakeholder	System design	H	H	H	H			•	H	M	M	M	H	**IA Task Mitigation:** Improved both information and time
C-6	I-14	External stakeholder	System design	M	H	M	M	•	•	•	H	M	M	M	M	**IA Task:** Ambiguity of information
C-6	I-14	External stakeholder	System design	H	H	M	H	•	•	•	H	M	M	M	M	**IA Task Mitigation:** Improving information availability should help
C-7	I-14	External stakeholder	System design	H	H	M	M	•	•	•	H	M	M	M	M	**IA Technique:** Partitioning, Synchronisation
C-7	I-14	External stakeholder	System design	H	H	H	M	•	•	•	H	M	M	M	M	**IA Technique:** Improving time available does not necessarily help
C-12	I-15	External stakeholder	System design	M	L	L	M		•	•	H	M	M	M	M	**IA Task:** Lack of information
C-12	I-15	External stakeholder	System design	H	L	L	H		•	•	H	M	M	M	M	**IA Task Mitigation:** Improving information should help
C-13	I-15	External stakeholder	System design	M	M	L	M	•	•	•	H	M	M	M	L	**IA Task:** Lack of time, Magnitude of Information
C-13	I-15	External stakeholder	System design	H	M	L	H	•	•	•	H	M	M	M	L	**IA Task Mitigation:** Improving information access should help
C-13	I-15	External stakeholder	System design	M	H	L	M	•	•	•	H	M	M	M	L	**IA Task:** Increasing resources has little effect
C-13	I-15	External stakeholder	System design	M	H	L	M	•	•	•	H	M	M	M	L	**IA Task:** Unlikely to increase time available given project budget and schedule
C-17	I-19	Software verification	System design	L	L	M	M			•	L	M	M	L	L	**IA Task:** Lack of information, Ambiguity of information, Lack of resources
C-17	I-19	Software verification	System design	H	L	M	M			•	L	M	M	L	L	**IA Task Mitigation:** Improving both information and resources should help
C-17	I-19	Software verification	System design	L	H	M	M			•	L	M	M	L	L	**IA Task Mitigation:** Improving both information and resources should help
C-48	I-19	External stakeholder	System design	M	M	M	M	•	•	•	H	M	M	M	M	**IA Task:** Magnitude of information, Volatility of information
C-48	I-19	External stakeholder	System design	H	H	H	H	•	•	•	H	M	M	M	M	**IA Task Mitigation:** Improving information availability should help
C-21	I-26	External stakeholder	System design	M	H	H	M	•	•	•	H	M	M	M	M	**IA Task:** Lack of information
C-21	I-26	External stakeholder	System design	H	H	H	H	•	•	•	H	M	M	M	M	**IA Task Mitigation:** Improving information availability should help
C-47	I-26	External stakeholder	System design	L	H	H	M	•	•	•	H	M	M	M	M	**IA Task:** Lack of information
C-47	I-26	External stakeholder	System design	H	H	H	H	•	•	•	H	M	M	M	M	**IA Task Mitigation:** Improving information availability should help
C-25	I-27	System design	Software design	M	H	M	H	•	•	•	M	L	L	H	M	**IA Technique:** Partitioning, Synchronisation
C-25	I-27	System design	Software design	M	H	M	H	•	•	•	M	L	L	H	M	**IA Technique Mitigation:** Worked with external stakeholder extensively
C-27	I-27	System design	Software design	L	H	M	M	•	•	•	M	L	L	H	M	**IA Task:** Lack of information
C-27	I-27	System design	Software design	L	H	M	M	•	•	•	M	L	L	H	M	**IA Task:** Unclear how to improve information available
C-28	I-27	Software design	Software design	H	H	L	M	•	•	•	L	L	L	L	L	**IA Technique:** Administration
C-28	I-27	Software design	Software design	H	H	L	M	•	•	•	L	L	L	L	L	**IA Technique:** Never enough time for thorough analysis using tools
C-29	I-28	System design	Software design	M	H	H	M	•	•	•	M	L	L	M	M	**IA Technique:** Partitioning, Synchronisation
C-29	I-28	System design	Software design	H	H	H	M	•	•	•	M	L	L	M	M	**IA Technique:** Improving information available does not necessarily help
C-33	I-29	System design	Software design	M	H	L	M	•	•		M	L	L	M	M	**IA Technique:** Over-extension
C-33	I-29	System design	Software design	M	H	H	M	•	•		M	L	L	M	M	**IA Technique:** More information or time available has little effect
C-33	I-29	System design	Software design	H	H	L	M	•	•		M	L	L	M	M	**IA Technique:** More information or time available has little effect
C-36	I-30	External stakeholder	System design	L	H	H	L	•		•	H	M	M	M	L	**IA Technique:** Partitioning, Synchronisation, Method definition
C-36	I-30	External stakeholder	System design	H	H	H	H	•		•	H	M	M	M	L	**IA Technique Mitigation:** Improving communication should help
C-38	I-30	External stakeholder	System design	H	H	H	H	•	•	•	H	M	M	H	H	**IA Technique Mitigation:** Improved communication helped
C-39	I-31	System design	Software design	M	L	L	M		•		M	L	L	M	M	**IA Task:** Lack of information, Volatility of information
C-39	I-31	System design	Software design	M	H	L	M		•		M	L	L	M	M	**IA Task Mitigation:** Improving resources might not help
C-39	I-31	System design	Software design	M	L	H	M		•		M	L	L	M	M	**IA Task Mitigation:** Improving time might not help
C-39	I-31	System design	Software design	H	L	L	H		•		M	L	L	M	M	**IA Task Mitigation:** Improving information should help significantly
C-41	I-32	External stakeholder	Software design	H	M	H	M		•	•	L	L	L	M	H	**IA Task:** Lack of resources
C-41	I-32	External stakeholder	Software design	H	H	H	H		•	•	L	L	L	M	H	**IA Task Mitigation:** Improving input from external stakeholder should help
C-42	I-32	System design	Software design	M	M	L	M		•	•	M	L	L	M	L	**IA Technique:** Partitioning, Synchronisation
C-42	I-32	System design	Software design	M	M	L	M		•	•	M	L	L	M	L	**IA Technique:** Unclear how to synchronise work given tight schedule

Figure E: Information, resource, time, and IA quality ratings and IA influences data collected

Figure F summarises the information in Figure E.

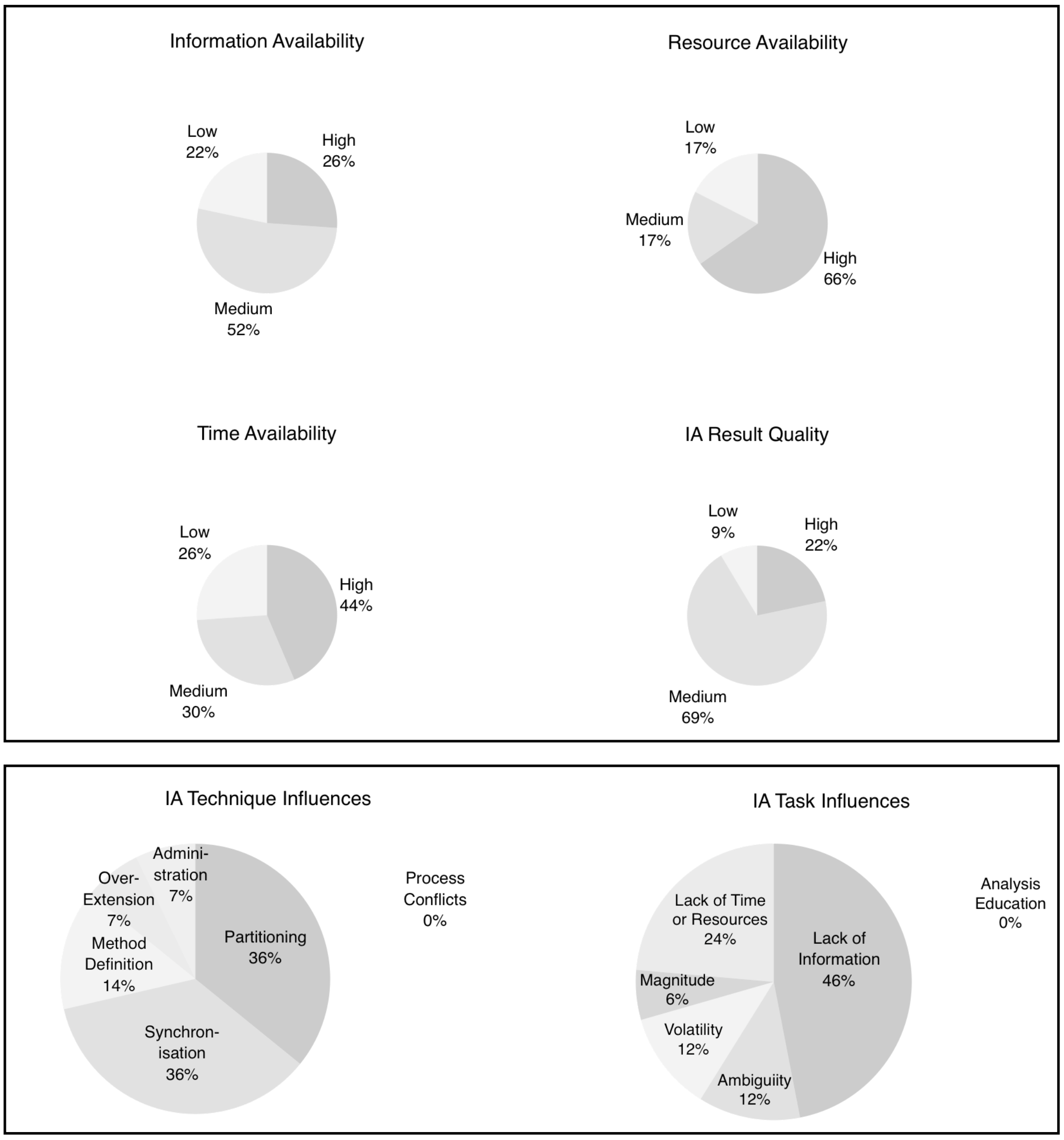


Figure F: Summary of ratings and IA influence data collected

APPENDIX C

Causal loop diagrams illustrate the relationships between variables in system dynamics models. Links between variables are given *positive* or *negative* polarities. Positive links indicate that an increase in the causal variable induces an increase in the related effect. Similarly for positive links, a decrease in the causal variable can lead to a decrease in the affected variable. Negative links denote an increase or decrease in the causal variable corresponds to a decrease or increase, respectively, in the related effect. Feedback loops are specified as *reinforcing* or *balancing* based on the product of the link polarities constructing the loop (*i.e.* multiplying the signs around the loop; two negative signs equal a positive sign). Reinforcing loops have a positive result, while balancing loops have a negative result. Figure G depicts the notation for causal loop diagrams described by Sterman (2000: 138).

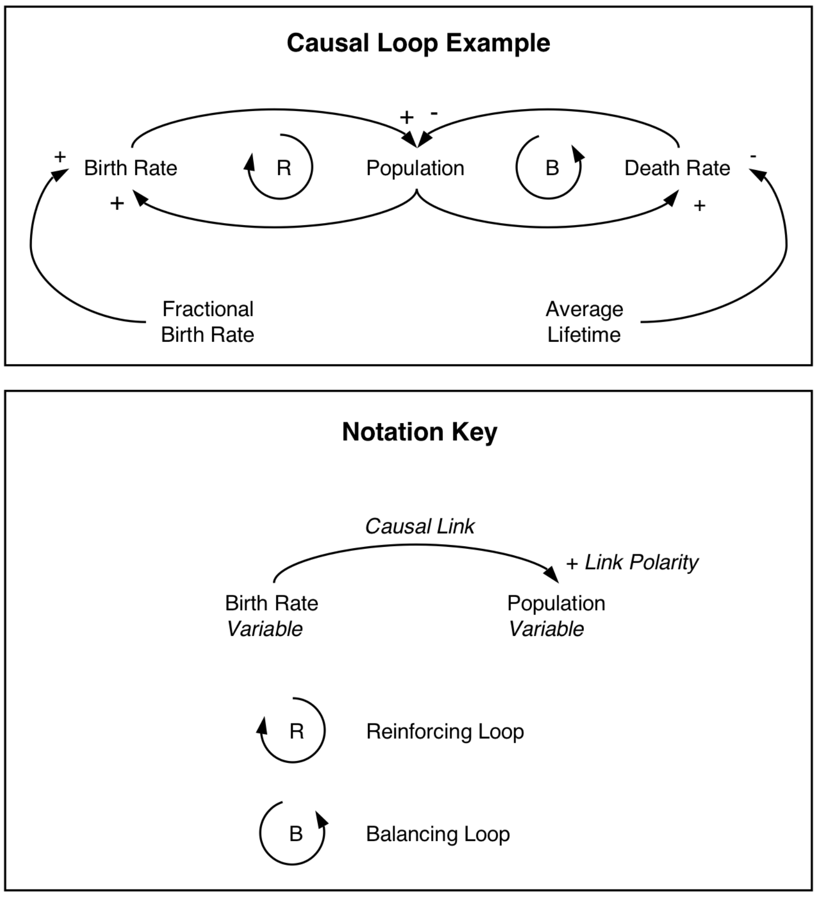


Figure G: Causal loop diagram notation (Sterman 2000: 138)

Stock and flow diagrams represent the feedback loops of causal loop diagrams, but also illustrate the accumulation and flow of material, work, etc. Typically, a bathtub metaphor is used to describe these diagrams. The flow of water into a tub via the tap and the flow of water out of the tub via the drain determine the stock of water in the tub. The tap and drain valves determine the rate of these flows. Sources and sinks are outside of the model boundaries and are assumed to have infinite capacity (*e.g.* water can always flow out of the tap, and water can always drain from the tub). As such, the flow of water with respect to time and the initial stock of water in the tub can be used to mathematically model this bathtub system. Figure H shows the stock and flow diagram notation used by Sterman (2000: 193).

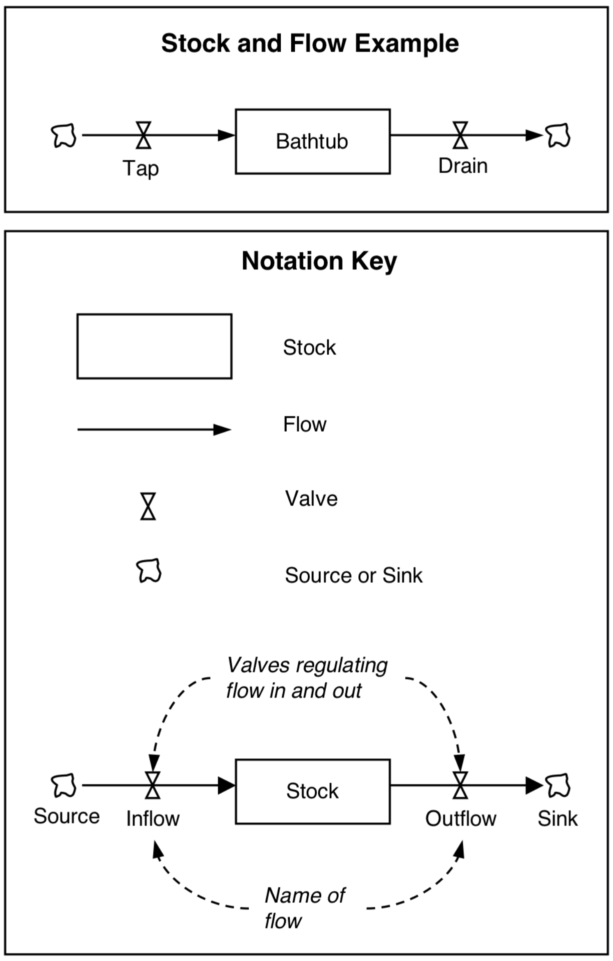


Figure H: Stock and flow diagram notation (Sterman 2000: 193)

APPENDIX D

The software change data set obtained from the aerospace company contains 19 fields for almost 1600 changes. Table B depicts the value formats primarily used since the notation and scales are not completely consistent within the data set. These documented changes were from a completed project at the company, while the 42 changes shown in Chapter 5 and Appendix B were from an on-going project.

Table B: Software change database fields of information

Variable	Description	Value Format
ID Number	Changes are given an identification number chronologically.	Integer
Status	Changes are classified according to their progress through the CCB and implementation.	Classification (Open, Closed, Cancelled, Approved)
Time of Change Request	Changes are entered into the database at the time of their request.	Date
Title	Changes are given a descriptive title.	Text
Safety Impact	Changes are rated according to their estimated impact on safety.	Scale Rating (No Impact, Minor Impact, Safety Hazard)
Safety Impact Description	Rationale for safety impact ratings is described in text.	Text
Functional Impact	Changes are rated according to the expected rework required.	Scale Rating (No Impact, Minor Impact, Major Impact)
Impact on Documentation for Customer	Changes are indicated to affect a particular document provided to the customer.	Classification (No, Documentation Chapter Number)
Software Type	Changes can affect one of two primary areas of the software design.	Classification (Type 1, Type 2)
Task Performed	Changes are classified according to the task that spawned the request.	Classification (Reviewing, Software Requirement Model Analysis, Testing)
Type of Change	Changes are categorised according to their type.	Classification (New Requirement, Improvement, Error)
Change Requester – Major Group	Internal or external stakeholders can initiate changes.	Classification (Controls, Customer, Supplier)
Change Requester – Sub Group	Changes initiated by internal stakeholders are further specified in terms of their primary source.	Classification (Software Requirements, Software Specification, Software Design)
Change Request Work Product	The work product affected by the change request is identified.	Classification (Control System Requirements, Control System Specification, System Design, Software Specification, Software Design, Traceability)
Change Description	Details of changes are described in text.	Text
Name of Change Requestor	Change requests are associated with designer initiating them.	Text
Name of Change Implementer	Change requests are assigned to a designer for implementation.	Text
Confirmation Task	The tasks used to confirm the change request details are classified.	Classification (Reviewing, Software Requirement Model Analysis, Testing)
Reason for Cancel	If a change request is cancelled by the CCB, the reason is captured.	Text

Even though the change database contains copious and detailed information about the changes occurring during the software development project, the data is not necessarily complete and contains subjectivity. For instance, additional modifications are also performed in practice during the implementation of requested changes, and some of the change requests can actually have larger scopes than depicted by their expected "functional impact" value. In turn, not documenting such unanticipated changes causes an underestimation of the rework implemented from analysing the change database. Moreover, classification categories are not always standardised, leading to multiple categories with similar connotations. Also, designers may classify changes non-uniformly since scales and categories can be interpreted differently.

Despite these limitations, this database provides some indication of the changes that occur at the aerospace company. For instance, several basic characteristics of this firm's change processes, which correspond to observations during the empirical studies, can be derived from the database (Table C). Note that not all percentages sum to 100% in Table C since some changes are classified into anomalous categories.

Table C: Characteristics of changes from data set

Characteristic	Information from the Data Set	Interpretation
Change Initiation	72% of changes are initiated by controls. Of which, 62% of changes are classified as "errors". 28% of changes are initiated by external stakeholders. Of which, 72% are "new requirements" or "improvements".	Many changes are generated based on rework from within controls. External stakeholders primarily generate changes to customise the control system platform.
Change Implementation	38% of changes are made in system design. Of which, 90% are in specification documentation. 48% of changes are made in software design. Of which, 50% affect software specifications, and the other 50% affect the other software design artefacts.	System designers primarily develop software requirements, while software designers construct the detail software specifications and implement them.
Change Identification	64% of change requests are confirmed only by review. 33% of change requests are confirmed only by testing. 1% of change requests are confirmed only by model analysis.	Multiple IA techniques may not be actively used during the initial analysis of change requests to identify potential knock-on modifications.

From Table C, the "change initiation" characteristic suggests that the software design process is highly iterative within the control system group and that the customisation of the product platform is driven by input from external stakeholders. In turn, the "change implementation" characteristic corresponds with the primary roles of system and software designers. These characteristics are consistent with the discussion of the empirical studies in Section 3.3 and Section 3.4.1, respectively. The "change identification" characteristic suggests that knock-on changes may only be investigated in detail during downstream tasks of design work (*i.e.* after the initial analysis of change requests) and correlates with the empirical findings in Section 3.4.2.1. IA performed prior to scheduling implementation may not be used to search thoroughly for additional design flaws necessitating changes. Thus, this characteristic hones in on the potential of the aerospace company to improve their application of IA to discover rework earlier.

APPENDIX E

As suggested in Section 6.4.2, 3 example change sets from the change database (Appendix D) were identified in 3 functional areas in order to depict the behaviour of the adapted rework cycle to improved IA quality (Section 6.3.2). The 3 representative change sets were each determined as follows: (a) the database was first combed to determine changes associated with a functional area; (b) changes to a specific design area within this functional area were then determined; and, (c) changes related within this design area were finally identified, grouping initiating changes with knock-on modifications. The examples analysed in Section 6.4.2 consist of the changes associated with specific design areas, outlined in Figure J.

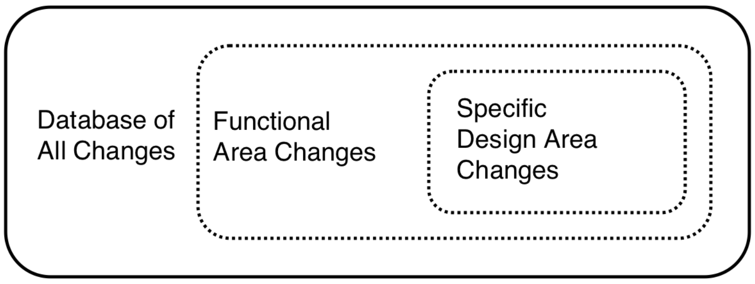


Figure J: The selection of changes to depict the adapted rework cycle behaviour

More specifically, the functional area changes were identified as follows:

1. All changes (either requested or implemented) by a designer were collected from the database, forming a subset.
2. All changes (either requested or implemented) by other designers involved in requesting or implementing changes within this data subset were then included into the subset.
3. Step 2 was repeated again with the refined data subset.

The names of designers were queried because the other data fields do not explicitly associate changes to functional areas, and changes associated with multiple designers were included in the subset since in some cases different designers led the development of a functional area during the 2.5 years of change data.

Since specific design areas within functional areas are assigned acronyms, the change title and description fields were searched for a variety of acronyms appearing in the subset. A specific design area associated with a manageable number of changes (*i.e.* less than 100) was then selected, and example change sets were created:

4. The change title, change description, and safety impact description fields in the change subset were searched for use of the selected design area's acronym or its longhand name. The changes found were allocated to the set of changes for the example design area.
5. All fields in the complete database were then searched again using the acronym and its longhand name to ensure that no changes in the example data set were missed. In all three functional areas investigated, this search produced no additional changes.

Related modifications were then identified within the set of specific design area changes:

6. Change titles, change descriptions, and safety impact descriptions were manually searched to form groups of changes due to the same causes.
7. All changes within the functional area were searched again using keywords from the causes of these change groups. In all three functional areas investigated, this search produced no additional changes.

For the purpose of the data analysis performed (Section 6.4.2), the earliest change in each group of related modifications identified is considered the initiating change and the remaining related changes are classified as knock-on changes. Notably, the change database does not align with the definition of IA quality or how interviewees describe the company's IA quality (I-2, I-12, I-15, I-16) in that the database does not depict applications of IA accounting for fractions of the remaining rework either through reporting decreasing functional impact values over time for related changes or reporting multiple, related changes simultaneously. As such, it is assumed that the reported changes in the database encompass a quantity of other knock-on changes (Section 7.3.1) that are not explicitly denoted to correspond with the definition of IA quality (Section 6.3.2).

As previously suggested, the causes of the modifications in the database description fields were used to relate changes. As opposed to forming groups of changes around design area dependencies in system and software architecture diagrams, this manner of grouping limits the scope of the related initiating and knock-on changes, as mentioned in Section 6.3.2. Such causes identified in the database were no more complicated[66] than the examples in Table D and Table E and, thus, suggest sets of related modifications readily found by IA.

As represented by the changes in Table D, a change to boundary conditions was implemented in the software design and only later changes to the system requirements and software requirement model were requested due to the same cause

[66] Other example causes include changes to the annunciation of faults and validation of signal inputs in requirement and design work products.

(*i.e.* the modified boundary conditions). Table E similarly depicts this pattern of changes affecting related work products to maintain consistency and also shows another pattern in that additional changes to the same work product (*i.e.* including additional error flags) can also emerge due to the same initiating cause (*i.e.* including additional logic) to ensure design completeness. These two patterns were the forms of related changes identified in the three example data sets searched.

Table D: Example A of related changes

Time of Change Request	Change Description	Change Request Work Product
t	Change to boundary conditions	Software Design
t + 1 year, 1 month	Update to boundary conditions in software requirement model	Control System Specification
t + 1 year, 1 month	Update to boundary conditions in software requirements	Control System Specification

Table E: Example B of related changes

Time of Change Request	Change Description	Change Request Work Product
t	Change to allow reading of data input and associated logic	Software Design
t + 1 month	Update to data input reading and logic	Software Specification
t + 1 month	Additional change to logic requirements due to misleading error flag in logic specification	Control System Specification
t + 2 months	Additional error flag included for completeness in design	Control System Specification
t + 3 months	Clarification of requirements requested by software design (ambiguity in logic required)	Control System Specification
t + 4 months	Specification of revised logic	Software Specification
t + 4 months	Implementation of revised logic	Software Design

Although the database was systematically searched, some related changes could have been missed. However, given the limited scope of the initiating and knock-on change groups, few changes, if any, could arguably fit into their specificity based on the defined causes. An increase of several such changes into the example design areas would unlikely change the characteristics of the known rework curves (Figure 6.19) used to derive the reference modes. Missing several changes within a group of related changes, and, therefore, increasing the total functional impact of the related changes, only causes an increase in a single data point as analysed for an impact analysis quality of 1, which does not necessarily affect the concavity of the known rework curve. If several missed changes affect different related change sets, then the impact is also negligible since this increase in rework is small, and the curve retains its shape.

BIBLIOGRAPHY

Abdel-Hamid, T. K. and S. E. Madnick (1991). Software Project Dynamics: An Integrated Approach. Englewood Cliffs, New Jersey, USA, Prentice Hall.

Ackoff, R. L. (2001). "OR: After the Post Mortem." System Dynamics Review **17**(4): 341-346.

Ajila, S. (1995). "Software Maintenance: An Approach to Impact Analysis of Objects Change." Software - Practice and Experience **25**(10): 1155-1181.

Alexander, I. (2002). Towards Automatic Traceability in Industrial Practice. International Workshop on Traceability, Edinburgh, UK.

Ambler, S. (1998). Process Patterns: Building Large-Scale Systems Using Object Technology. Cambridge, UK, Cambridge University Press.

Ambler, S. (1999). More Process Patterns: Delivering Large-Scale Systems Using Object Technology. Cambridge, UK, Cambridge University Press.

Ambler, S. (2002). Agile Modeling: Effective Practices for Extreme Programming and the Unified Process. New York, New York, USA, John Wiley & Sons.

Andriole, S. J. and P. A. Freeman (1993). "Software Systems Engineering: The Case for a New Discipline." Software Engineering Journal **8**(3): 165-179.

Apiwattanapong, T., A. Orso, *et al.* (2005). Efficient and Precise Dynamic Impact Analysis Using Execute-After Sequences. International Conference on Software Engineering (ICSE 2005), St. Louis, Missouri, USA.

Auslander, D. M., J. R. Ridgely, *et al.* (2002). Control Software for Mechanical Systems: Object Oriented in a Real Time World. Upper Saddle River, New Jersey, USA, Prentice Hall.

Bacharach, S. B. (1989). "Organizational Theories: Some Criteria for Evaluation." The Academy of Management Review **14**(4): 496-515.

Baddoo, N. and T. Hall (2003). "De-motivators for Software Process Improvement: An Analysis of Practitioners." Empirical Software Engineering **66**(1): 23-33.

Bailey, D. E. and S. R. Barley (2005). "Return to Work: Toward Post-Industrial Engineering." IIE Transactions **37**(8): 737-752.

Barlas, Y. (1996). "Formal Aspects of Model Validity and Validation in System Dynamics." System Dynamics Review **12**(3): 183-201.

Bass, L., P. C. Clements, *et al.* (2006). Capturing and Using Rationale for a Software Architecture. Rationale Management in Software Engineering. A. H. Dutoit, R. McCall, I. Mistrík and B. Paech. Berlin, Germany, Springer: 255-272.

Bate, R. R. (1998). "Do Systems Engineering? Who, Me?" IEEE Software **15**(4): 65-66.

Beecham, S., T. Hall, *et al.* (2003). "Software Process Improvement Problems in Twelve Software Companies: An Empirical Analysis." Empirical Software Engineering **8**(1): 7-42.

Beedle, M., A. van Bennekum, *et al.* (2001). "Manifesto for Agile Software Development." Retrieved 20 June 2007, from http://agilemanifesto.org.

Binkley, D., N. Gold, *et al.* (2007). "An Empirical Study of Static Program Slice Size." ACM Transactions on Software Engineering and Methodology **16**(2): 8.

Blanchard, B. S. (2003). System Engineering Management. Hoboken, New Jersey, USA, John Wiley & Sons.

Blessing, L. and A. Chakrabarti (2002). DRM: A Design Research Methodology. Proceedings of Les Sciences de la Conception, Lyon, France.

Blessing, L., A. Chakrabarti, *et al.* (1995). A Design Research Methodology. International Conference on Engineering Design (ICED 1995), Prague, Czech Republic.

Bloor, M. (1997). Techniques of Validation in Qualitative Research: A Critical Commentary. Context and Method in Qualitative Research. G. Miller and R. Dingwall. London, UK, Sage Publications.

Boehm, B. (1988). "A Spiral Model of Software Development and Enhancement." IEEE Computer **21**(5): 61-72.

Boehm, B. (2000). "Unifying Software Engineering and Systems Engineering." IEEE Computer **33**(3): 114-116.

Boehm, B. (2002). "Get Ready for Agile Methods, with Care." IEEE Computer **35**(1): 64-69.

Boehm, B. and H. Kitapei (2006). The WinWin Approach: Using a Requirements Negotiation Tool for Rationale Capture and Use. Rationale Management in Software Engineering. A. H. Dutoit, R. McCall, I. Mistrík and B. Paech. Berlin, Germany, Springer: 173-190.

Boehm, B. and R. Turner (2005). "Management Challenges to Implementing Agile Processes in Traditional Development Organizations." IEEE Computer **22**(5): 30-39.

Bohner, S. A. and R. S. Arnold, Eds. (1996). Software Change Impact Analysis. Los Alamitos, California, USA, IEEE Computer Society Press.

Bolton, W. (1999). Mechatronics: Electronic Control Systems in Mechanical Engineering. Harlow, UK, Prentice Hall.

Bradley, D. A., D. Seward, *et al.* (2000). Mechatronics and the Design of Intelligent Machines and Systems. Cheltenham, UK, Stanley Thornes.

Bratthall, L., E. Johansson, *et al.* (2000). Is a Design Rationale Vital when Predicting Change Impact? – A Controlled Experiment on Software Architecture Evolution. International Conference on Product Focused Software Process Improvement, Oulu, Sweden.

Bratthall, L. and M. Jørgensen (2002). "Can you Trust a Single Data Source Exploratory Software Engineering Case Study?" Empirical Software Engineering **7**(1): 9-26.

Bray (2002). An Introduction to Requirements Engineering. Harlow, UK, Addison-Wesley.

Breech, B., M. Tegtmeyer, *et al.* (2005). A Comparison of Online and Dynamic Impact Analysis Algorithms. European Conference on Software Maintenance and Reengineering (CSMR 2005), Manchester, UK.

Briand, L. C., Y. Labiche, *et al.* (2003). Impact Analysis and Change Management of UML Models. International Conference on Software Maintenance (ICSM 2003), Amsterdam, Netherlands.

Briand, L. C., Y. Labiche, *et al.* (2005). "Automated Impact Analysis of UML Models." Journal of Systems and Software **79**(3): 339-352.

Briand, L. C., J. Wüst, *et al.* (1999). Using Coupling Measurement for Impact Analysis in Object-Oriented Systems. International Conference on Software Maintenance (ICSM 1999), Oxford, UK.

Brooks, F. P. (1995). The Mythical Man-Month: Essays on Software Engineering. Reading, Massachusetts, USA, Addison-Wesley.

Brown, W. J., H. W. McCormick, *et al.* (1999). AntiPatterns and Patterns in Software Configuration Management. New York, New York, USA, John Wiley & Sons.

Browning, T. R., E. Fricke, *et al.* (2006). "Key Concepts in Modeling Product Development Processes." Systems Engineering **9**(2): 104-128.

Buckley, J., T. Mens, *et al.* (2003). "Towards a Taxonomy of Software Change." Journal of Software Maintenance and Evolution: Research and Practice **17**(5): 309-332.

Buckner, J., J. Buchta, *et al.* (2005). JRipples: A Tool for Program Comprehension during Incremental Change. International Workshop on Program Comprehension, St. Louis, Missouri, USA.

Buede, D. M. (2000). The Engineering Design of Systems: Models and Methods. New York, New York, USA, John Wiley & Sons.

Bush, M. (2004). "Unlocking the Hidden Logic of Process Improvement: Peer Reviews." CrossTalk: The Journal of Defense Software Engineering **17**(3): 14-17.

Canfora, G. and A. Cimitile, Eds. (2001). Software Maintenance. Handbook of Software Engineering & Knowledge Engineering. Skokie, Illinois, USA, World Scientific Publishing Company.

Centinkunt, S. (2007). Mechatronics. Hoboken, New Jersey, USA, John Wiley & Sons.

Chalupnik, M. J., D. C. Wynn, *et al.* (2007). Understanding Design Process Robustness: A Modelling Approach. International Conference on Engineering Design (ICED 2007), Paris, France.

Chao, L. P., I. Tumer, *et al.* (2004). Design Process Error-Proofing: Engineering Peer Review Lessons from NASA. Design Engineering Technical Conference (DETC 2004), Salt Lake City, Utah, USA.

Chapin, N., J. E. Hale, *et al.* (2001). "Types of Software Evolution and Software Maintenance." Journal of Software Maintenance and Evolution: Research and Practice **13**(1): 3-30.

Charette, R. N. (2005). "Why Software Fails." IEEE Spectrum **42**(9): 42-49.

Chmura, L. J., A. F. Norcio, *et al.* (1990). "Evaluating Software Design Processes by Analyzing Change Data Over Time." IEEE Transactions on Software Engineering **16**(7): 729-740.

Chrissis, M. B., M. Konrad, *et al.* (2007). CMMI: Guidelines for Process Integration and Product Improvement. Upper Saddle River, New Jersey, USA, Addison-Wesley.

Chudge, J. and D. Fulton (1996). "Trust and Co-operation in System Development: Applying Responsibility Modelling to the Problem of Changing Requirements." Software Engineering Journal **11**(3): 193-204.

Ciolkowski, M., O. Laitenberger, *et al.* (2003). "Software Reviews: The State of the Practice." IEEE Software **20**(6): 46-60.

Clarkson, P. J., C. S. Simons, *et al.* (2004). "Predicting Change Propagation in Complex Design." ASME Journal of Mechanical Design **126**(5): 765-797.

CMMI (2006). CMMI for Development, Version 1.2. Pittsburgh, Pennsylvania, USA.

Comerford, R. (1994). "Mecha...what?" IEEE Spectrum **31**(8): 46-49.

Cooper, K. G. (1993a). "Rework Cycle: Why Projects Are Mismanaged." PM Network **7**(2): 5-7.

Cooper, K. G. (1993b). "The Rework Cycle: Benchmarks for the Project Manager." Project Management Journal **24**(1): 17-21.

Cooper, K. G. (1993c). "The Rework Cycle: How It Really Works... and Reworks..." PM Network **7**(2): 25-28.

Cresswell, J. W. (1998). Qualitative Inquiry and Research Design. Thousand Oaks, California, USA, Sage Publications.

Crowston, K. (1997). "A Coordination Theory Approach to Organizational Process Design." Organization Science **8**(2): 157-175.

Curtis, B., H. Krasner, *et al.* (1988). "A Field Study of the Software Design Process for Large Systems." Communications of the ACM **31**(11): 1268-1287.

Damian, D., J. Chisan, *et al.* (2005). "Requirements Engineering and Downstream Software Development: Findings from a Case Study." Empirical Software Engineering **10**(3): 255-283.

Daniels, J. and T. Bahill (2004). "The Hybrid Process That Combines Traditional Requirements and Use Cases." Systems Engineering **7**(4): 303-319.

Davies, R. (2005). "Agile Requirements." Methods and Tools **13**(3): 24-30.

de Micheli, G. and R. K. Gupta (1997). "Hardware/Software Co-Design." Proceedings of the IEEE **85**(3): 349-365.

de Weck, O. L. (2007). "Design for Changeability." Retrieved 10 June 2007, from http://strategic.mit.edu/content/view/25/73/.

DeGrace, P. and L. H. Stahl (1993). The Olduvai Imperative: CASE and the State of Software Engineering Practice. Upper Saddle River, New Jersey, USA, Yourdon Press.

Dick, J. (2005). "Design Traceability." IEEE Software **22**(6): 14-16.

Ebner, G. and H. Kaindl (2002). "Tracing All Around in Reengineering." IEEE Software **19**(3): 70-77.

Eckert, C. M., P. J. Clarkson, *et al.* (2005). Predictability of Change in Engineering: A Complexity View. ASME International Design Engineering Technical Conferences (DETC 2005), Long Beach, California, USA.

Eckert, C. M., P. J. Clarkson, *et al.* (2003). The Spiral of Applied Research: A Methodological View on Integrated Design Research. International Conference on Engineering Design (ICED 2003), Stockholm, Sweden.

Eckert, C. M., P. J. Clarkson, *et al.* (2004). "Change and customisation in complex engineering domains." Research in Engineering Design **15**(1): 1-21.

Eick, S. G., T. L. Graves, *et al.* (2002). "Visualizing Software Changes." IEEE Transactions on Software Engineering **28**(4): 396-412.

Eisner, H. (2002). Essentials of Project and Systems Engineering Management. New York, New York, USA, John Wiley & Sons.

Ellims, M., J. Bridges, *et al.* (2006). "The Economics of Unit Testing." Empirical Software Engineering **11**(1): 5-31.

Emam, K. E. and D. Hoeltje (1997). "Qualitative Analysis of a Requirements Change Process." Empirical Software Engineering **2**(2): 143-207.

Endres, A. and D. Rombach (2003). A Handbook of Software and Systems Engineering: Empirical Observations, Laws and Theories. New York, New York, USA, Addison-Wesley.

Englander, I. (2000). The Architecture of Computer Hardware and Systems Software: An Information Technology Approach. New York, New York, USA, John Wiley & Sons.

Flowers, S. (1996). Software Failure: Management Failure - Amazing Stories and Cautionary Tales. Chichester, UK, John Wiley & Sons.

Ford, A. (1999). Modeling the Environment: An Introduction to System Dynamics Models of Environmental Systems. Washington D.C., USA, Island Press.

Ford, D. N. and J. D. Sterman (1998). "Dynamic Modeling of Product Development Processes." System Dynamics Review **14**(1): 31-68.

Ford, D. N. and J. D. Sterman (2003a). "Overcoming the 90% Syndrome: Iteration Management in Concurrent Development Projects." Concurrent Engineering: Research and Applications **11**(3): 177-186.

Ford, D. N. and J. D. Sterman (2003b). "The Liar's Club: Concealing Rework in Concurrent Development." Concurrent Engineering: Research and Applications **11**(3): 211-219.

Fowler, M. (2004). "Is Design Dead?" Retrieved 20 June 2007, from http://www.martinfowler.com/articles/designDead.html.

French, M. (1998). Conceptual Design for Engineers. London, UK, Springer.

Fricke, E., B. Gebhard, *et al.* (2000). "Coping with Changes: Causes, Findings, and Strategies." Systems Engineering **3**(4): 169-179.

Fricke, E. and A. P. Schultz (2005). "Design for Changeability (DfC): Principles To Enable Changes in Systems Throughout Their Entire Lifecycle." Systems Engineering **8**(4): 342-359.

Gates, B. (2007). "A Robot in Every Home." Scientific American (January 2007).

Gelle, E. (2005). Modeling Multiple System Families. International Joint Conference on Artificial Intelligence, Edinburgh, UK.

George, B. and S. A. Bohner (2004). Software Information Leaks: A Complexity Perspective. International Conference on Engineering of Complex Computer Systems. Florence, Italy.

Gibson, R. (2003). Software and Systems Engineering: Conflict and Consensus. Practicing Software Engineering in the 21st Century. J. Peckham and S. J. Lloyd, IRM Press: 26-41.

Giffin, M., O. L. de Weck, *et al.* (2007). Change Propagation in Complex Technical Systems. International Design Engineering Technical Conference (DETC 2007). Las Vegas, Nevada, USA.

Gilb, T. and D. Graham (1993). Software Inspection. Reading, Massachusetts, USA, Addison-Wesley.

Gilbert, L. S. (2002). "Going the Distance: 'Closeness' in Qualitative Data Analysis Software." International Journal of Social Research Methodology **5**(3): 215-228.

Gonzales, R. (2005). "Developing the Requirements Discipline: Software vs. Systems." IEEE Software **22**(2): 59-61.

Gorsline, G. W. (1986). Computer Organization: Hardware/Software. Englewood Cliffs, New Jersey, USA, Prentice Hall.

Goulding, C. (2002). Grounded Theory: A Practical Guide for Management, Business and Market researchers. London, UK, Sage Publications.

Graaf, B., M. Lormans, *et al.* (2003). "Embedded Software Engineering: The State of the Practice." IEEE Software **20**(6): 61-68.

Grady, J. O. (2006). System Requirements Analysis. Boston, Massachusetts, USA, Academic Press.

Hallin, K., T. Zimmerman, *et al.* (2003). Modelling Information for Mechatronic Products. International Conference on Engineering Design (ICED 2003), Stockholm, Sweden.

Han, J. (1997). Supporting Impact Analysis and Change Propagation in Software Engineering Environments. International Workshop on Software Technology and Engineering Practice (STEP 1997), London, UK.

Harwell, R. M. (1998). Implementation of EIA-632: Process Compliance or Embedded Infrastructure? Digital Avionics Systems Conference (DASC 1998), Bellevue, Washington, USA.

Haskins, C., Ed. (2006). INCOSE Systems Engineering Handbook. Seattle, Washington, USA, INCOSE.

Hass, A. M. (2003). Configuration Management Principles and Practice. Boston, Massachusetts, USA, Addison-Wesley.

Hassan, A. E. and R. C. Holt (2004). Predicting Change Propagation in Software Systems. International Conference on Software Maintenance (ICSM 2004), Chicago, Illinois, USA.

Hayes, J. H., A. Dekhtyar, *et al.* (2005). "Improving After-the-Fact Tracing and Mapping: Supporting Software Quality Predictions." IEEE Software **22**(6): 30-37.

Hayhurst, K. J. and C. M. Holloway (2002). "Aviation Software Guidelines." IEEE Software **19**(5): 107.

Heimdahl, M. P. E. and N. G. Leveson (1996). "Completeness and Consistency in Hierarchical State-Based Requirements." IEEE Transactions on Software Engineering **22**(6): 363-377.

Hicks, P. R. (2006). "Adapting Legacy Systems for DO-178B Certification." CrossTalk: The Journal of Defense Software Engineering **19**(8): 27-30.

Hillier, F. S. and G. J. Lieberman (1995). Introduction to Operations Research. New York, New York, USA, McGraw-Hill.

Hofman, H. F. (2000). Requirements Engineering: A Situated Discovery Process. Wiesbaden, Germany, Deutscher Universitäts-Verlag.

Holt, J. (2001). UML for Systems Engineering: Watching the Wheels. London, UK, IEE.

Homkes, R. (2006). Systems Interfacing, Instrumentation, and Control Systems. Mechatronics: An Introduction. R. H. Bishop. Boca Raton, Florida, USA, Taylor & Francis.

Horwitz, S., T. Reps, *et al.* (1990). Interprocedural Slicing Using Dependence Graphs. Software Change Impact Analysis. S. A. Bohner and R. S. Arnold. Los Alamitos, California, USA, IEEE Computer Society Press**:** 137-171.

Höst, M., B. Regnell, *et al.* (2001). "Exploring Bottlenecks in Market-Driven Requirements Management Processes with Discrete Event Simulation." Journal of Systems and Software **59**(3): 323-332.

Huhn, W. and M. Schaper (2006). "Getting Better Software into Manufactured Products." The McKinsey Quarterly (March).

Hull, E., K. Jackson, *et al.* (2004). Requirements Engineering. London, UK, Springer.

Humphrey, W. S. (2000). Introduction to TEAM Software Process. Reading, Massachusetts, USA, Addison-Wesley.

Humphrey, W. S., M. D. Konrad, *et al.* (2007). "Future Directions in Process Improvement." CrossTalk: The Journal of Defense Software Engineering **20**(2): 17-22.

IEEE (1993). "software engineering". IEEE Standard Glossary of Software Engineering Terminology (corrected edition). New York, New York, USA, IEEE Computer Society Press.

IEEE (2005a). Adoption of ISO/IEC 15288:2002 Systems Engineering - System Life Cycle Processes.

IEEE (2005b). IEEE Std 1220 - IEEE Standard for Application and Management of the Systems Engineering Process.

INCOSE. (2004a). "SE Tools Taxonomy - Requirements Traceability Tools." Retrieved 20 June 2006, from http://www.incose.org/productspubs/products/setools/tooltax/reqtrace_tools.html.

INCOSE. (2004b). "What is Systems Engineering?" Retrieved 6 June 2007, from http://www.incose.org/practice/whatissystemseng.aspx.

ISO/IEC (2002). ISO/IEC 15288 Systems Engineering - System Life Cycle Processes.

Jacobson, I., G. Booch, *et al.* (1999). The Unified Software Development Process. Boston, Massachusetts, USA, Addison-Wesley.

Jalkio, J. (2006). The Role of Modeling in Mechatronics Design. Mechatronics: An Introduction. R. H. Bishop. Boca Raton, Florida, USA, Taylor & Francis.

Jalote, P. (2000). CMM in Practice: Processes for Executing Software Projects at Infosys. Reading, Massachusetts, USA, Addison-Wesley.

Jarratt, T. A. (2004). A Model-Based Approach to Support the Management of Engineering Change. PhD Thesis. University of Cambridge. Cambridge, UK.

Johnson, C. W. (2006). "Why Did That Happen? Exploring the Proliferation of Barely Usable Software in Healthcare Systems." Quality and Safety in Health Care **15**(Supplement 1): 76-81.

Jones, C. (2006). "Social and Technical Reasons for Software Project Failures." CrossTalk: The Journal of Defense Software Engineering **19**(6): 4-9.

Kaindl, H. (2005). System and Software Co-Architecting. Conference on Systems Engineering Research (CSER 2005), Hoboken, New Jersey, USA.

Kanoun, K. and M. Ortalo-Borrel (2000). "Fault-Tolerant System Dependability - Explicit Modeling of Hardware and Software Component Interactions." IEEE Transactions on Reliability **49**(4): 363-376.

Kececi, N. and A. Abran (2001). An Integrated Measure for Functional Requirements. International Workshop on Software Maintenance (IWSM 2001), Montreal, Quebec, Canada.

Kellner, M. I., R. J. Madachy, *et al.* (1999). "Software Process Simulation Modeling: Why? What? How?" Journal of Systems and Software **46**(2-3): 91-105.

Kettunen, P. (2003). "Managing Embedded Software Project Team Knowledge." IEE Proceedings in Software **150**(6): 359-366.

Koffi, A. D. (2005). "A Model for the Evolution of Software and Systems Engineering Project Cultures Throughout Their Life Cycles." Systems Engineering **8**(2): 151-163.

Kossiakoff, A. and W. N. Sweet (2003). Systems Engineering: Principles and Practice. Hoboken, New Jersey, USA, John Wiley & Sons.

Kotonya, G. and I. Sommerville (1998). Requirements Engineering: Processes and Techniques. Chichester, UK, John Wiley & Sons.

Kung, D. C., J. Gao, *et al.* (1994). Change Impact Identification in Object Oriented Software Maintenance. International Conference on Software Maintenance (ICSM 1994). Victoria, British Colombia, Canada.

Kusiak, A. (1993). Concurrent Engineering: Automation, Tools, and Techniques. New York, New York, USA, John Wiley & Sons.

Lacaze, X., P. Palanque, *et al.* (2006). From DREAM to Reality: Specificities of Interactive Systems Development With Respect to Rationale Management. Rationale Management in Software Engineering. A. H. Dutoit, R. McCall, I. Mistrík and B. Paech. Berlin, Germany, Springer**:** 155-172.

Lane, D. C. (1994). "With a Little Help from Our Friends: How System Dynamics and Soft OR Can Learn from Each Other." System Dynamics Review **10**(2-3): 101-134.

Law, J. and G. Rothermel (2003). Incremental Dynamic Impact Analysis for Evolving Software Systems. International Symposium on Software Reliability Engineering (ISSRE), Denver, Colorado, USA.

Lee, S., J. Shim, *et al.* (2002). A UML Approach for Software Change Modeling. Asia-Pacific Software Engineering Conference (APSEC 2002), Gold Coast, Queensland, Australia.

Leishman, T. R. and D. A. Cook (2003). "But I Only Changed One Line of Code!" CrossTalk: The Journal of Defense Software Engineering **16**(1): 20-23.

Lethbridge, T. C., J. Singer, *et al.* (2003). "How Software Engineers Use Documentation: The State of the Practice." IEEE Software **20**(6): 35-39.

Leveson, N. G. and C. S. Turner (1993). "An Investigation of the Therac-25 Accidents." Computer **26**(7): 18-41.

Leveson, N. G. and K. A. Weiss (2004). "Making Embedded Software Reuse Practical and Safe." ACM SIGSOFT Software Engineering Notes **29**(6): 171-178.

Lindvall, M. (1997a). An Empirical Study of Requirements-Driven Impact Analysis in Object-Oriented Software Evolution. PhD Thesis. Linköping University. Linköping, Sweden.

Lindvall, M. (1997b). "Evaluating Impact Analysis - A Case Study " Empirical Software Engineering **2**(2): 152-158.

Lindvall, M. and K. Sandahl (1998). "Traceability Aspects of Impact Analysis in Object-oriented Systems." Software Maintenance: Research and Practice **10**(1): 37-57.

Lock, S. and G. Kotonya (1999). "An Integrated, Probabilistic Framework for Requirement Change Impact Analysis." The Australian Journal of Information Systems **6**(2): 93-141.

Lyneis, J. M., K. G. Cooper, *et al.* (2001). "Strategic Management of Complex Projects: A Case Study Using System Dynamics." System Dynamics Review **17**(3): 237-260.

Lyneis, J. M. and D. N. Ford (2007). "System Dynamics Applied to Project Management: A Survey, Assessment, and Directions for Future Research." System Dynamics Review **23**(2/3): 157-189.

Madachy, R. J. (2007). "Software Process Dynamics." Retrieved 7 September 2007, from http://www-rcf.usc.edu/~madachy/spd.

Madachy, R. J. (2008). Software Process Dynamics. Hoboken, New Jersey, USA, John Wiley & Sons.

Maier, M. and E. Rechtin (2000). The Art of Systems Architecting. Boca Raton, Florida, USA, CRC Press LLC.

Marcus, A., J. I. Maletic, *et al.* (2005). "Recovery of Traceability Links Between Software Documentation and Source Code." International Journal of Software Engineering and Knowledge Engineering **15**(4): 811-836.

Martin, J. N. (1998). Overview of the EIA 632 Standard: Processes for Engineering a System. Digital Avionics Systems Conference (DASC 1998). Bellevue, Washington, USA.

Metallect. (2007). "IQ Server Product Brief." Retrieved 15 February 2008, from http://www.metallect.com/products.php.

Miller, K., T. Camp, *et al.* (2000). "Computing Cases." Retrieved 7 June 2007, from http://www.computingcases.org/case_materials/therac/analysis/SocioTechnical_Analysis.html.

Mrozek, Z. (2002). Design of the Mechatronic System with the Help of UML Diagrams. Workshop on Robot Motion and Control, Bukowy Dworek, Poland.

Neill, C. J. and P. A. Laplante (2003). "Requirements Engineering: The State of the Practice." IEEE Software **20**(6): 40-45.

Neumann, P. G. (1995). Computer Related Risks. Reading, Massachusetts, USA, Addison-Wesley.

Nuseibeh, B. and S. Easterbrook (2000). Requirements Engineering: A Roadmap. International Conference on Software Engineering (ICSM 2000), Limerick, Ireland.

Nuseibeh, B., S. Easterbrook, *et al.* (2000). "Leveraging Inconsistency in Software Development." IEEE Computer **33**(4): 24-29.

O'Neal, J. S. (2003). Analysing the Impact of Changing Software Requirements: A Traceability-Based Methodology. PhD Thesis. Louisiana State University. Baton Rouge, Louisiana, USA.

O'Neal, J. S. and D. L. Carver (2001). Analyzing the Impact of Changing Requirements. International Conference on Software Maintenance (ICSM 2001), Florence, Italy.

Olson, T. G. (2006). "Defining Short and Usable Processes." CrossTalk: The Journal of Defense Software Engineering **19**(6): 24-28.

Ott, L. M., A. Kinnula, *et al.* (1999). "The Role of Empirical Studies in Process Improvement." Empirical Software Engineering **4**(4): 381-386.

OxfordEnglishDictionary. (1989). "interface, n." Oxford English Dictionary Online Retrieved 6 June 2007, from http://dictionary.oed.com/cgi/entry/50119021.

Pahl, G., W. Beitz, *et al.* (1996). Engineering Design A Systematic Approach. London, UK, Springer.

Palmer, J. D. (2002). Traceability. Software Engineering: Volume 1: The Development Process. R. H. Thayer and M. Dorfman. Los Alamitos, California, USA, IEEE Computer Society Press.

Pap, Z., I. Majzik, *et al.* (2001). Completeness and Consistency Analysis of UML Statechart Specifications. Design and Diagnostics of Electronic Circuits and Systems Workshop (DDECS 2001), Győr, Hungary.

Park, M. and F. Peña-Mora (2003). "Dynamic Change Management for Construction: Introducing the Change Cycle into Model-Based Project Management." System Dynamics Review **19**(3): 213-242.

Parnas, D. L. and P. C. Clements (1986). "A Rational Design Process: How and Why to Fake It." IEEE Transactions on Software Engineering **12**(2): 251-256.

Patton, M. Q. (2002). Qualitative Research & Evaluation Methods. Thousand Oaks, California, USA, Sage Publications.

Perry, D. E., A. A. Porter, *et al.* (2000). Empirical Studies of Software Engineering: A Roadmap. International Conference on Software Engineering. Limerick, Ireland.

Perry, D. E., H. P. Siy, *et al.* (2001). "Parallel Changes in Large-Scale Software Development: An Observational Case Study." ACM Transactions on Software Engineering and Methodology **10**(3): 308-337.

Peterson, I. (1995). Fatal Defect: Chasing Killer Computer Bugs. New York, New York, USA, Random House.

Peterson, J. and J. Hiles (1976). "Conference Report 2nd International Conference on Software Engineering or a Systems Engineer by any Other Name..." IEEE Computer **9**(12): 67-71.

Pfahl, D., A. Al-Emran, *et al.* (2007). "A System Dynamics Simulation Model for Analyzing the Stability of Software Release Plans." Software Process Improvement and Practice **12**(5): 475-490.

Pfleeger, S. L. and J. M. Atlee (2006). Software Engineering: Theory and Practice. Upper Saddle River, New Jersey, USA, Prentice Hall.

Raffo, D. M. and M. I. Kellner (1999a). "Empirical Analysis in Software Process Simulation Modeling." Journal of Systems and Software **53**(1): 31-41.

Raffo, D. M. and M. I. Kellner (1999b). Modeling Software Processes Quantitatively and Evaluating the Performance of Process Alternatives. Elements of Software Process Assessment and Improvement. K. El Emam and N. H. Madhavji. Los Alamitos, California, USA, IEEE Computer Society Press: 297-341.

Raffo, D. M. and M. I. Kellner (2000). "Empirical Analysis in Software Process Simulation Modeling." The Journal of Systems and Software **53**(1): 31-41.

Rainer, A. and T. Hall (2003). "A Quantitative and Qualitative Analysis of Factors Affecting Software Processes." Journal of Systems and Software **66**(1): 7-21.

Rajlich, V. (1997). A Model for Change Propagation Based on Graph Rewriting. International Conference on Software Maintenance (ICSM 1997), Bari, Italy.

Rajlich, V. (2000). "A Model and a Tool for Change Propagation in Software." ACM SIGSOFT Software Engineering Notes **25**(1): 72.

Rajlich, V. and P. Gosavi (2004). "Incremental Change in Object-Oriented Programming." IEEE Software **21**(4): 62-69.

Ramesh, B. (1998). "Factors Influencing Requirements Traceability Practice." Communications of the ACM **41**(12): 37-44.

Ren, X., O. C. Chesley, *et al.* (2006). "Identifying Failure Causes in Java Programs: An Application of Change Impact Analysis." IEEE Transactions on Software Engineering **32**(9): 718-732.

Ren, X., F. Shah, *et al.* (2005). Chianti: A Tool for Change Impact Analysis of Java Programs. International Conference on Software Engineering (ICSE 2005), St. Louis, Missouri, USA.

Rierson, L. K. (2000). A Systematic Process for Changing Safety-Critical Software. Digital Avionics Systems Conference (DASC 2000), Philadelphia, Pennsylvania, USA.

Robertson, S. and J. Robertson (1999). Mastering the Requirements Process. Harlow, UK, Addison-Wesley.

Rosenberg, S. (2007). "Anything You Can Do, I Can Do Meta." Technology Review **110**(1): 36-48.

Rottman, J. W. (2006). "Successfully Outsourcing Embedded Software Development." IEEE Computer **39**(1): 55-61.

Ruiz, M., I. Ramos, *et al.* (2001). "A Simplified Model of Software Project Dynamics." The Journal of Systems and Software **59**(3): 299-309.

Sage, A. P. (1990). Software Systems Engineering. New York, New York, USA, John Wiley & Sons.

Sage, A. P. (2000). Introduction to Systems Engineering. New York, New York, USA, John Wiley & Sons.

Saliu, O. and G. Ruhe (2005). "Supporting Software Release Planning Decisions for Evolving Systems." Innovations in Systems and Software Engineering: A NASA Journal **1**(2): 189-204.

Seaman, C. B. (1999). "Qualitative Methods in Empirical Studies of Software Engineering." IEEE Transactions on Software Engineering **25**(4): 557-572.

Sengupta, K. and T. K. Abdel-Hamid (1996). "The Impact of Unreliable Information on the Management of Software Projects: A Dynamic Decision Perspective." IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans **26**(2): 177-189.

Shaaf, R. (2005). "What's All This about Systems?" Computer **38**(6): 102-104.

Sheard, S. A. (2001). "Evolution of the Frameworks Quagmire." IEEE Computer **34**(7): 96-98.

Sommerville, I. (1998). "Systems Engineering for Software Engineers." Annals of Software Engineering **6**(1-4): 111-129.

Sommerville, I. (2001). Software Engineering. Harlow, UK, Addison-Wesley.

Sommerville, I. (2007). Software Engineering. Harlow, UK, Addison-Wesley.

Sommerville, I. and P. Sawyer (1997). Requirements Engineering: A Good Practice Guide. Chichester, UK, John Wiley & Sons.

Sridharan, M., S. J. Fink, *et al.* (2007). Thin Slicing. Programming Language Design and Implementation, San Diego, California, USA.

Stephenson, Z. and J. McDermid (2005). "Deriving Architectural Flexibility Requirements in Safety-Critical Systems." IEE Proceedings in Software **152**(4): 143-152.

Sterman, J. D. (2000). Business Dynamics: Systems Thinking and Modeling for a Complex World. Boston, Massachusetts, USA, McGraw-Hill.

Strauss, A. L. and J. M. Corbin (1998). Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory. Thousand Oaks, California, USA, Sage Publications.

Strens, M. R. and R. C. Sugden (1996). Change Analysis: A Step towards Meeting the Challenge of Changing Requirements. International Symposium and Workshop on Engineering of Computer Based Systems (ECBS 1996), Friedrichshafen, Germany.

Swanson, E. B. (1976). The Dimensions of Maintenance. International Conference on Software Engineering (ICSE 1976). San Francisco, California, USA.

System Dynamics Society. (2007). Retrieved 05 September 2007, from http://www.systemdynamics.org.

Takahashi, M. and Y. Kamayachi (1989). "An Empirical Study of a Model for Program Error Prediction." IEEE Transactions on Software Engineering **15**(1): 82-86.

Taylor, T. and D. N. Ford (2006). "Tipping Point Failure and Robustness in Single Development Projects." System Dynamics Review **22**(1): 51-71.

Thelin, T., H. Petersson, *et al.* (2004). "Applying Sampling to Improve Software Inspections." Journal of Systems and Software **73**(2): 257-269.

Thomé, B., Ed. (1993). Systems Engineering: Principles and Practice of Computer-Based Systems Engineering. Chichester, UK, John Wiley & Sons.

Tsantalis, N., A. Chatzigeorgiou, *et al.* (2005). "Predicting the Probability of Change in Object-Oriented Systems." IEEE Transactions on Software Engineering **31**(7): 601-614.

Turner, R. (2007). "Toward Agile Systems Engineering Processes." CrossTalk: The Journal of Defense Software Engineering **20**(4): 11-15.

van Brussel, H. (1996). "Mechatronics - A Powerful Concurrent Engineering Framework." IEEE/ASME Transactions on Mechatronics **1**(2): 127-136.

van den Berg, K. (2006). Change Impact Analysis of Crosscutting in Software Architectural Design. Workshop on Architecture-Centric Evolution (ACE 2006), Nantes, France.

van Vliet, H. (2000). Software Engineering: Principles and Practice. New York, New York, USA, John Wiley & Sons.

von Knethen, A. (2002). Change-Oriented Requirements Traceability: Support for Evolution of Embedded Systems. International Conference on Software Maintenance (ICSM 2002), Montreal, Canada.

von Mayrhauser, A., A. M. Vans, *et al.* (1997). "Program Understanding Behaviour during Enhancement of Large-scale Software." Software Maintenance: Research and Practice **9**(5): 299-327.

Wakeland, W. W., R. H. Martin, *et al.* (2004). "Using Design of Experiments, Sensitivity Analysis, and Hybrid Simulation to Evaluate Changes to a Software Development Process: A Case Study." Software Process Improvement and Practice **9**(2): 107-119.

Wallace, D. R. and D. R. Kuhn (2001). "Medical Device Software: An Analysis of 15 Years of Recall Data." International Journal of Reliability, Quality, and Safety Engineering **8**(4): 351-371.

Weinberg, G. M. (2003). "Destroying Communication and Control in Software Development." CrossTalk: The Journal of Defense Software Engineering **16**(4): 4-8.

Weiser, M. (1984). "Program Slicing." IEEE Transactions on Software Engineering **10**(4): 352-357.

White, B. A. (2000). Software Configuration Management Strategies and Rational ClearCase: A Practical Introduction. Boston, Massachusetts, USA, Addison-Wesley.

White, S. M. (2005). Improving the System/Software Engineering Interface for Complex System Development. International Conference and Workshops on Engineering of Computer-Based Systems (ECBS 2005), Greenbelt, Maryland, USA.

Whitgift, D. (1991). Methods and Tools for Software Configuration Management. West Sussex, UK, John Wiley & Sons.

Wiegers, K. (1994). "Creating a Software Engineering Culture." Software Development **2**(7): 59-66.

Wynn, D. C., C. M. Eckert, *et al.* (2006). Applied Signposting: A Modeling Framework to Support Design Process Improvement. ASME International Design Engineering Technical Conferences (DETC 2006), Philadelphia, Pennsylvania, USA.

Yourdon, E. (1989). Modern Structured Analysis. Englewood Cliffs, New Jersey, USA, Yourdon Press.

Yourdon, E. (1999). Death March. Upper Saddle River, New Jersey, USA, Prentice Hall.

Zhang, W. J. and Q. Li (1999). Design for Control: A Proposed Methodology for Developing Mechatronic Systems. International Conference on Engineering Design (ICED 1999), Munich, Germany.