

Investigating Requirements Volatility during Software Development Projects:

An Empirical Study

Nurmuliani (B.AgrSc., M.App.St)

A Thesis Submitted for the Degree of

DOCTOR OF PHILOSOPHY

University of Technology Sydney
Faculty of Information Technology

March 2007

CERTIFICATE OF AUTHORSHIP/ORIGINALITY

I certify that the work in this thesis has not previously been submitted for a degree nor has it been submitted as part of requirements for a degree except as fully acknowledged within the text.

I also certify that the thesis has been written by me. Any help that I have received in my research work and the preparation of the thesis itself has been acknowledged. In addition, I certify that all information sources and literature used are indicated and referenced in the thesis.

Signature of Candidate

Production Note:
Signature removed prior to publication.

Acknowledgement

This thesis was completed with the support and guidance of a great number of people whose contribution in various ways to the research and the making of the thesis deserved special mention.

I would like to express my deep and sincere gratitude to my supervisor, A/Professor Didar Zowghi for her support and sharp sense of research direction. Her understanding, encouraging and personal guidance have provided a good basis to complete this thesis.

I am deeply grateful to my supervisor, A/Professor Susan P. William, for her detailed and constructive comments, and her valuable advice and supervision to improve the quality of this thesis.

I owe my sincere gratitude to Mr. Lakshminarayanan Vaidyanathasamy, who gave me the opportunity to conduct this long-term case study in his organisation. I warmly thank to Mr. Yogendra Pal for his valuable advice and friendly help. I am thankful to him for giving me insights into software development project practices. Many thanks go to all participants from this company for their support throughout this research.

I owe my deepest gratitude to my beloved husband and my partner in life, Greg, who supported me all the way. To my son Arie and my daughter Sofia, thank you for your infinite patience, love, and understanding during the lengthy process of this study. I am truly grateful to my mother, brothers and sisters for their best wishes, prayers and support.

I would like to thank my colleague, Ms Norazlin Yusop, Mr Chad Coulin, Mr Samiaji Sarosa, for their support and constructive discussions. I would like to also express my gratitude to Dr Bruce Howarth for proofreading this thesis.

Lastly, and most importantly, I would like to dedicate this thesis to my late father and my dearest mother who formed part of my vision and taught me to keep learning and taking on greater challenges.

Preface

Some of the work described in this thesis has been published previously in several conference proceedings:

- 1) (Nurmuliani et al., 2004a) describes case study analysis on requirements volatility (*Project Release_A*) and the initial development of requirements change classification, as discussed in Chapter 4.
- 2) (Nurmuliani et al., 2004b) describes the use of the card sorting technique to classify requirements change requests from software developers' perspectives, discussed in Chapter 4.
- 3) (Nurmuliani et al., 2005) describes case study analysis on requirements volatility (*Project Release_B*) and the use of requirements change classification, discussed in Chapter 5.
- 4) (Nurmuliani et al., 2006) discusses requirements volatility issues and its impact of change effort, challenges and lessons learned in analysing requirements volatility during software development project. This work is discussed in Chapter 4 and 5.

Table of Contents

Chapter 1:	Introduction.....	1
1.1	Background	1
1.2	Problem Definition and Motivation	3
1.3	Research Objectives and Research Questions.....	4
1.4	Thesis Contribution.....	6
1.5	Thesis Roadmap	7
Chapter 2:	Literature Review.....	9
2.1	Introduction	9
2.2	Software Engineering.....	10
2.2.1	Software Lifecycle Models	11
2.3	Requirements Engineering	13
2.3.1	The Concept of a Requirement	13
2.3.2	An Overview of Requirements Engineering	14
2.3.3	Challenges in Requirements Engineering	18
2.4	Requirements Volatility (RV).....	20
2.4.1	Definitions.....	20
2.4.2	Requirements Volatility Measures.....	23
2.4.3	Potential Causes of Requirements Volatility	24
2.4.4	Impacts of Requirements Volatility	27
2.5	Recommended Strategies for Handling RV.....	30
2.5.1	Incremental Software Development Process	31
2.5.2	Change Control Process.....	32
2.5.3	Requirements Engineering Practices.....	32
2.5.4	Effective Communication	35
2.5.5	Requirements Change Classification	36
2.6	Summary and Proposed Research Directions	37
2.6.1	Sources of Change	38
2.6.2	Change Impacts.....	39
2.6.3	Strategies.....	39

Chapter 3: Research Methodology	42
3.1 Introduction.....	42
3.2 Theoretical Perspective	44
3.3 Mapping Research Objectives, Questions, and Strategies	45
3.4 The Case Study Methodology	47
3.4.1 Case Study Protocol	48
3.4.2 Case Study Design	49
3.4.3 Types of case study design and site selection	50
3.4.4 Data Collection Procedures.....	52
3.4.5 Data Management and Analysis	56
3.5 Applying the Case Study Design	58
3.6 Research Validity, Reliability, and Generalisability.....	61
3.6.1 Reliability	62
3.6.2 Construct Validity	62
3.6.3 Internal Validity	62
3.6.4 External Validity (Generalisability).....	64
3.7 Ethical Consideration.....	64
3.8 Summary	65
Chapter 4: Case Study Findings – Part 1	66
4.1 Introduction.....	66
4.2 Phase_1: Organisational Context.....	67
4.2.1 Organisation Background	68
4.2.2 Software Application Overview.....	69
4.2.3 Requirements Engineering Practices.....	70
4.2.4 Software Project Characteristics	76
4.3 Phase_2: Change Analysis_1	83
4.3.1 Phase_2: Data Collection and Analysis Methods	83
4.3.2 Phase_2: Case Study Findings from Project Release_A.....	85
4.3.3 Phase_2: Empirical Analysis of Requirements Volatility using the Classification – Project Release_A	93
4.3.4 Phase_2: Additional Findings on the Change Control Process and the CR Form.....	104

4.3.5	Phase_2: Summary	105
4.4	Phase_3: Validation of the Classification	106
4.4.1	Phase_3: Card Sorts Overview.....	106
4.4.2	Phase_3: Data Collection and Analysis Framework.....	107
4.4.3	Card Sort Setting and Participants	108
4.4.4	Phase_3: Card Sort Findings.....	112
4.4.5	Phase_3: Summary.....	119
4.5	Summary	119
Chapter 5:	Case Study Findings – Part 2	121
5.1	Introduction.....	121
5.2	Phase_4: Change Process Assessment.....	122
5.2.1	Phase_4: Data Collection and Analysis Methods	123
5.2.2	Phase_4: Findings on the Assessment of Change Process Documents	124
5.2.3	Phase_4: Survey Analysis and Findings	132
5.2.4	Phase_4: Summary of the Change Process Assessment.....	140
5.3	Phase_5: Change Analysis_2.....	141
5.3.1	Phase_5: Data Collection and Analysis Framework.....	141
5.3.2	Phase_5: Empirical Findings.....	143
5.3.3	Phase_5: Empirical Analysis of Requirements Volatility using Classification – Project Release_B	152
5.3.4	Phase_5: Additional Findings on the Change Control Process and the CR Form.....	162
5.3.5	Phase_5: Summary of the Change Analysis_2	164
5.4	Phase_6: Evaluation of the New Change Control Process and CR Form....	164
5.4.1	Phase_6: Data Collection and Analysis	165
5.4.2	Phase_6: Findings	166
5.4.3	Phase_6: Survey Findings.....	172
5.4.4	Phase_6: Summary of the new CCP evaluation.....	180
5.5	Summary	181
Chapter 6:	Analysis and Extension of Theory and Practice	183
6.1	Introduction.....	183

6.2	Classification of Requirements Change	183
6.2.1	The development of Requirements Change Classification	185
6.2.2	The Benefits of Classification.....	187
6.2.3	Requirements Change Attributes	188
6.3	Measuring Requirements Volatility	195
6.4	Representing Requirements Volatility	196
6.5	The Cost of Requirements Volatility	198
6.6	Strategies for Handling Requirements Volatility.....	201
6.6.1	Recommended Strategies	201
6.6.2	Strategies in Current Practice.....	202
6.6.3	Strategies for handling RV.....	206
6.7	Summary	209
Chapter 7:	Conclusions and Future Work.....	211
7.1	Introduction	211
7.2	Research Objectives/Questions revisited.....	211
7.3	Thesis Contributions Revisited	218
7.4	Implications for Research	219
7.4.1	Requirements Change Classification	219
7.4.2	Measuring and Representing Requirements Volatility	220
7.4.3	Requirements Volatility and Change Effort.....	220
7.5	Implications for Practice	221
7.5.1	Change Analysis Approach.....	221
7.5.2	Requirements Change Classification	222
7.5.3	Measuring and Representing Requirements Volatility	223
7.5.4	Requirements Volatility and Change Effort.....	223
7.5.5	Change Process and Change Request Form.....	224
7.6	Strategies for Managing RV	225
7.6.1	Reactive Strategy	225
7.6.2	Proactive Strategy	227
7.7	Challenges, Lessons Learned and Future Work.....	229
7.7.1	Challenges.....	229
7.7.2	Lessons Learned.....	230

7.7.3	Future Research Directions	231
7.8	Reflections on the Research	232
7.9	Concluding Remarks	233
	Bibliography	234
	Appendix A: Ethics Committee Approval Document	244
	Appendix B: Consent Form	246
	Appendix C: Project Release_A Change Request Datasheet	248
	Appendix D: Card Sorting Feedback Sheet	253
	Appendix E: Participants' Card Sorting Result	255
	Appendix F: Participants' Feedback on Card Sort	258
	Appendix G: Pre- Survey Questionnaire on Change Control Process	265
	Appendix H: Pre-Survey Data Table	272
	Appendix I: Project Release_B Datasheet	279
	Appendix J: Survey Questionnaire on a new Change Request Process	283
	Appendix K: Post-Survey Data Table	289
	Appendix L: Detailed Summary of Post-Survey Results	299
	Appendix M: Summary of All Requirements Change Request Data	307

List of Figures

Figure 2.1 The Multiple Aspect of Requirements Volatility	38
Figure 3.1. Research process at different stages	42
Figure 3.2. The structure of the case study research.....	59
Figure 4.1 The first three phases of research activities.....	67
Figure 4.2 Information flow of New Feature Suggestion (NFS)	71
Figure 4.3 Requirements development and negotiation process at GDS.....	72
Figure 4.4 Software development lifecycle - <i>Project Release_A</i> (Source: adapted from GDS's internal document)	78
Figure 4.5 The 'diagram' used in the analysis of each feature and the Excel Form capturing the constructs of the sentence template (Source: Damian, et al., 2002)	79
Figure 4.6 Software development lifecycle adopted - <i>Project Release_B</i> (Source: Internal GDS document)	80
Figure 4.7 The 'diagram' used in the requirements analysis for <i>Project Release_B</i> (Source: Internal GDS document).....	82
Figure 4.8 <i>Phase_2: Data Collection and Analysis Framework - Project Release_A</i>	85
Figure 4.9 Change Requests Arrival Rate for Project Release_A	87
Figure 4.10 The Rate of Requirements Volatility – <i>Project Release_A</i> (Low-level requirements).....	88
Figure 4.11 An overview of requirements change classification – <i>Project Release_A</i>	92
Figure 4.12 Overview of RV and the total change effort throughout the development lifecycle of <i>Project Release_A</i>	93
Figure 4.13 Pareto analysis of Total Estimated Effort by <i>Reason Category</i> and <i>Change Types</i> – <i>Project Release_A</i>	95
Figure 4.14 Total requirements change classified by <i>Change Origin</i> and <i>Change Types</i> – <i>Project Release_A</i>	96

Figure 4.15 Total requirements change classified by <i>Reason Category</i> and <i>Change Types</i> – Project Release_A.....	97
Figure 4.16 Requirements Addition by Reason Category – Project Release A.....	98
Figure 4.17 Requirements Deletion by Reason Category - Project Release A.....	101
Figure 4.18 Requirements Modifications by <i>Reason Category</i> - <i>Project Release A</i>	102
Figure 4.19 <i>Phase 3: The validation process</i>	108
Figure 4.20 An example of the Cards	109
Figure 4.21 A participant's set of cards after sorting.....	111
Figure 4.22 Criteria generated by the participants at GDS	112
Figure 4.23 The distribution of categories according to the sorting criteria.....	113
Figure 5.1 The last three phases of research activities.....	122
Figure 5.2 Change Control Process Flow	128
Figure 5.3 The importance level of the CCP activities by the participants.....	133
Figure 5.4 Agreement with the CCP and CR form	134
Figure 5.5 Participants' Agreement on the aspects of CR review	136
Figure 5.6 Participants' Agreement on the change impact information	137
Figure 5.7 Participants' Agreement on the aspects of CR implementation	138
Figure 5.8 The easiness of tracking the implemented Change Requests	139
Figure 5.9 Data Collection and Analysis Framework - <i>Project Release_B</i>	142
Figure 5.10 Change Request Arrival Rate for <i>Project Release_B</i>	143
Figure 5.11 Requirements Volatility Rate - <i>Project Release_B</i> (High-level requirements).....	144
Figure 5.12 Requirements Volatility Rate - <i>Project Release_B</i> (Low-level requirements).....	145
Figure 5.13 An overview of requirements change classification – <i>Project Release_B</i>	149
Figure 5.14 An overview of RV and total change effort spent throughout the development lifecycle of Project Release_B.....	150
Figure 5.15 Pareto analysis of Total estimated effort by Reason Category and Change Types - Project Release_B	151
Figure 5.16 Total requirements change classified by <i>Change Origin</i> and <i>Change Type</i> - <i>Project Release_B</i>	153

Figure 5.17 Total requirements change classified by Reason Category and Change Type – Project Release_B	154
Figure 5.18 Requirements additions, reason categories, and change effort (<i>Project Release_B</i>)	156
Figure 5.19 Requirements deletions, reason categories, and change effort (<i>Project Release_B</i>)	158
Figure 5.20 Requirements modification, <i>reason categories</i> , and change effort (<i>Project Release_B</i>)	161
Figure 5.21 Type-A Change Request Process Flow (Source: GDS document).....	167
Figure 5.22 Type-B Change Request Process Flow (Source: GDS document).....	168
Figure 5.23 Type-C Change Request Process Flow (Source: GDS document).....	169
Figure 5.24 The importance level of some of the CCP activities	173
Figure 5.25 The Initiators' agreement on the CR form and reviewer's response....	174
Figure 5.26 The Initiators' perspective on CR types and change impact analysis .	175
Figure 5.27 The Initiators' perspective on CR types and appropriate reviews.....	176
Figure 5.28 The Reviewers' agreement on the aspect of new CCP	177
Figure 5.29 The Reviewers' agreement on the change impact information.....	178
Figure 5.30 The Implementers' agreement on aspects of CR implementation.....	179
Figure 5.31 Overall view of the new CCP (left) and its supporting tool (right).....	180
Figure 6.1. Requirements Change Classification Process.....	186
Figure 6.2. A summary of <i>Reason Category</i> attributes derived from the current research (literature) and the current practice (this case study).....	191
Figure 6.3. A summary of <i>Change Origins</i> derived from current research (literature) and current practice (this case study).....	194
Figure 7. 1. Change Analysis Approach	222

List of Tables

Table 3.1 Research Objectives, Questions, and Research Strategies adopted	46
Table 4.1 Descriptions of Change Types	90
Table 4.2 Descriptions of Change Origin	90
Table 4.3 Descriptions of Reason Categories	91
Table 4.4 Number of Categories Elicited in Superordinate Construct Groups.....	115
Table 4.5 The Reason Category attributes from two classifications.....	117
Table 5.1 Requirements change types.....	147
Table 5.2 Descriptions of Change Origin	147
Table 5.3 Descriptions of Reason Categories	148
Table 5.4 A summary of the existing CCP and the new CCP	170
Table 5.5 A summary of the existing CR form and the new CR form	171

Abstract

Changes to software requirements are inevitable during the development process. Despite advances in software engineering over the past three decades, requirements changes are a source of project risk in software development, particularly when businesses and technologies are evolving rapidly.

This so-called requirements volatility has attracted much attention, but its extent and consequences are not well understood. The research literature lacks empirical studies investigating requirements volatility, particularly its underlying causes and consequences, and there are no effective strategies to deal with the associated problems throughout software development. We address these issues with a long-term case study in an industrial software development setting to identify and characterise the causes of requirements volatility, its impacts on the software development process, and the strategies used by current system development practitioners to deal with requirements volatility problems.

We analysed requirements change request data from two software project releases, and investigated the organisation's handling of requirements changes. Our data include the change request database, project documents, interviews, observations, and regular discussions with the key informants from the project members. We used a combination of qualitative and quantitative research techniques.

We first present a critical review of the literature on requirements volatility issues, from which an analytic synthesis for a currently lacking comprehensive coverage of requirements volatility phenomena is derived. The review clarifies the terms used, the sources and adverse impacts of requirements volatility, and the strategies available to current software development teams. We also provide a detailed description of a repeatable research design that researchers and practitioners could use to conduct similar investigation of requirements volatility in any industry setting.

We developed requirements change classifications from the change request data. Project members also classified requirements change requests using a card sorting technique. The resulting categories play a vital role in the empirical analysis of

several aspects of requirements volatility. Its extent can be characterised by such classification attributes as the types of change (addition, deletion, and modification), reasons for change, and change origin. The classification is useful in analysing the cost of requirements change in terms of rework or effort required. Based on an empirical analysis using the proposed classification, effective strategies were defined to match organisational needs. The organisation was able to use these results to improve its change control process and its change request form, thereby improving management and reducing the impacts of requirements volatility.

Chapter 1: Introduction

1.1 Background

Software has become a driving force in everyday life and business decision making. Its application in the modern world is seen in small to large companies, and in all kinds of systems such as financial, telecommunications, military, governments, and medical. As information technology grows rapidly, software is everywhere, from withdrawing cash from an ATM, taking digital photos, making a phone call, to driving cars. It is apparent that software is becoming integral to our way of life (Charette, 2005).

Software is known as the key driver behind most new technologies; however, the engineering of software is becoming more complicated as systems become increasingly complex. As a relatively new discipline, software engineering is concerned with analysing, designing, constructing and maintaining reliable and cost-effective software for complex systems.

In recent years, software engineering research and practice has been continually challenged by rapid changes in the environments in which software is situated. The engineering of good software for large and complex systems must address the demands of distributed applications on heterogeneous platforms, since many organisations are now operating wide-area, heterogeneous distributed systems. It is apparent that this situation also requires software to be adaptable to dynamic changes in the environment.

Despite advances in software engineering over the past 35 years, a considerable number of software projects still fail to be delivered on time, within planned budget, and often exhibit poor quality (Pressman and Ince, 2000). A survey conducted by The Standish Group shows that 28% of projects were cancelled before completion, and 46% of projects were still over-time and over-budget (The Standish Group,

1998). This highlights a challenge for software engineers to continue improving their techniques and processes to improve productivity and quality.

A software system is developed through a set of phases: inception, initial development, implementation, deployment, and maintenance. This set of phases is referred to as the *software lifecycle model*. As a logical system element, software is engineered through its own lifecycle model (Pressman and Ince, 2000). Developing large software systems is considerably more difficult because it involves complex engineering tasks that may require iteration and rework before completion (Marciniak, 2002). Whatever lifecycle model is used to develop software, the software development project commences with an initial phase that is now widely referred to as *requirements engineering (RE)*.

There is growing recognition of RE as a relatively new and important area in software engineering research and practice. It is believed that a successful RE process plays a crucial role for software development companies producing high-quality, reliable software. There are still many problems faced by current software development organisations in improving quality, reducing cost and dealing with changes. These problems also challenge requirements engineering researchers to keep improving requirements engineering techniques.

It is well understood that RE is a critical phase in all software development lifecycle models. Most software developers would agree with Brooks' assessment in his well-known article (Brooks, 1987) that the hardest part of building a software system is determining what to build. No other part of software development is as difficult as establishing the detailed technical requirements. In the last four decades, substantial evidence of project failures due to requirements problems has emerged. A recent report by Charette (2005) cites 'badly defined system requirements' as one of the primary causes of software project failure. This indicates that defining requirements for a software system is still a real problem in the software development process, particularly in situations where software systems are complex and highly integrated. These systems have long expected lifetimes and during that time they evolve substantially to accommodate inevitable changes in the requirements and environment (Lehman, 1998).

1.2 Problem Definition and Motivation

Software development is a dynamic process and best characterised by continuous evolution. In recent years, dealing with changes while software is being developed has become the way of life for most software projects. Evolution of software is prompted by all kinds of changes. Requirements evolve because the requirements cannot be fully gathered at the beginning of a project: the customers always change their mind(s), and the knowledge of users/stakeholders/developers in engineering requirements increases over time. Rapid changes in technologies, business rules governing the utilisation of software, and the environment in which software is situated also lead to changes in software requirements (Zowghi and Offen, 1997). Changing requirements are indeed a central point of all software projects and such changes cannot be avoided; they are an intrinsic factor that must be addressed.

Managing changes to requirements throughout the software development lifecycle is one of the most significant issues in requirements engineering. Frequent changes to requirements, also known as requirements volatility (RV), requirements change, or requirements evolution, are considered an undesirable property of software projects. Uncontrolled requirements volatility causes major difficulties during the software development lifecycle (Curtis et al., 1988) and is regarded as one of the factors that cause a project to be over time and over budget (The Standish Group, 1998). Managing and controlling requirements volatility is important because of its ripple effects on the rest of the system (Lubar et al., 1993, Sutcliffe, 2002).

Various studies report requirements volatility as a source of project risk that project managers have to handle during the software development lifecycle (Hyatt and Rosenberg, 1996, Tiwana and Keil, 2004). Requirements volatility also causes significant project uncertainty and its impact results in increases in cost, schedule, and development effort (Hammer et al., 1998, Stark et al., 1999, Zowghi et al., 2000).

Despite RV being widely acknowledged as an important issue in the software engineering field, the problems related to requirements volatility are still not well understood. Studies to date indicate that there is a lack of empirical evidence on requirements volatility problems; in particular, its underlying causes and

consequences are still not well understood. There has not been a comprehensive study to investigate the multiple aspects of requirements volatility, including effective strategies to deal with the problem throughout the software development lifecycle. Many researchers have been engaged to handle RV problems and have proposed mechanisms or approaches that have been academically tested, but often they are not relevant to the problems in practical life. This is a challenge for researchers: to conduct more empirical research that will address problems in practical life and provide information to better understand those problems.

Conducting an empirical study, particularly undertaking a case study in a ‘real’ software project within an organisational setting is difficult because the case study method does not have a well-understood theoretical basis (Kitchenham and Pickard, 1998). However, the benefit of such case studies is that they represent the current state of requirements volatility problems that organisations are facing.

The paucity of comprehensive empirical evidence has prevented the clear identification of the role of requirements volatility on various issues underlying the requirements volatility problems. This research will address these shortfalls of the previous studies. This study will empirically identify and characterise requirements volatility problems and their impact on software development effort, and identify strategies used by current software development practitioners to handle requirements volatility problems.

1.3 Research Objectives and Research Questions

The goal of this study is to increase understanding about the characteristics of requirements volatility and to develop effective strategies to handle requirements volatility problems during the software development lifecycle. The research objectives deriving from research goals are:

[Obj1] To examine the characteristics of requirements volatility in the software development lifecycle.

[Obj2] To establish the status of current theory in managing requirements volatility and gaps in current research.

[Obj3] To understand the way an organisation manages requirements volatility throughout the software development lifecycle.

[Obj4] To identify limitations of requirements volatility management in practice.

[Obj5] To identify gaps between current theory and current practice in managing requirements volatility.

[Obj6] If necessary, to extend the current theory by proposing effective strategies for the management of requirements volatility.

Arising from the research objectives above, the research questions are defined as follows:

[RQ1] What are the characteristics of requirements volatility identified in the literature?

[RQ1a] What are the sources of, reasons for, and types of changes, that contribute to requirements volatility?

[RQ1b] What are the impacts of requirements volatility on software development projects?

[RQ2] What potential strategies are useful for managing requirements volatility?

[RQ3] How does an organisation manage requirements volatility?

[RQ3a] What sources of, reasons for, types of changes that contribute to requirements volatility? and: what are the impacts of requirements volatility on software development projects?

[RQ3b] What strategies does an organisation use to manage requirements volatility in practice?

[RQ3c] What are the current limitations of these strategies?

[RQ4] What gaps exist between current theory and current practice?

[RQ4a] Are there any differences between the sources of, reasons for, types of changes, and impacts of requirements volatility described in the current theory and those identified in the current practice?

[RQ4b] What are the differences between the strategies outlined in the current theory and the strategies the organisation apply in practice?

[RQ5] If necessary, can current theory be extended to accommodate the imperatives and needs of current practice?

1.4 Thesis Contribution

This thesis makes several important contributions to the discipline of software engineering in general, and the field of requirements engineering in particular. The focus is on requirements volatility management during software development. The following are the principal contributions drawn from the thesis findings:

- A critical review of the literature on requirement volatility issues is presented. This includes a survey of requirements volatility characteristics, a critical analysis of current strategies, and an analysis of the management of requirements volatility and its relationship with the software development project. This review provides an analytic synthesis of a currently lacking comprehensive understanding of the concept of requirements volatility. It clarifies the terms used, the sources of and adverse impacts of requirements volatility, and the potential strategies that can be adopted in current software development practice.
- A research design involving a longitudinal case study method in an industrial setting is described in detail. This research methodology provides valuable insight into conducting long-term case study research of this kind in industry.
- A requirements change classification based on change request data collected from two software project releases is presented. This classification can be used to identify and trace the problems of, reasons for, sources of, and impacts of requirements volatility. The classification also plays a pivotal role in requirements change management. This classification increases our understanding of the requirements change.
- The application of Card Sorting approach to classify requirement change attributes from software practitioners' perspectives is presented. The process of conducting a card sorting exercise is described and presented in detail, so that it can be utilised by researchers and practitioners. The results of the exercise are also used to validate the preliminary classification developed earlier.

- A novel representation and measurement of RV characteristics are presented. A graphical presentation is used as an important mechanism to understand requirements volatility characteristics. The results from data analysis demonstrate each aspect of the RV characteristics throughout the software development lifecycle including the total (estimated) cost of RV.
- Two main strategies (*Reactive* and *Proactive Strategies*) for managing requirements volatility are summarised from the literature and the case study findings. The applicable approaches within these strategies, which software practitioners can apply in their organisational settings, are highlighted.

In summary, this thesis provides a novel contribution to the investigation of requirements volatility problems, understanding its underlying causes and its associated impacts and cost.

1.5 Thesis Roadmap

There are six further chapters to this thesis. An overview of these chapters follows.

Chapter 2 provides theoretical perspectives motivating the investigation of requirements volatility problems during software development projects. The relevant literature from the fields of software engineering, information systems, and requirements engineering is reviewed and critically analysed. A comprehensive survey of requirements volatility issues, terms and concepts, potential causes and their adverse impacts is also presented. Current debates on strategies for managing requirements volatility problems are discussed. The chapter concludes by presenting the essential elements to be focused on in characterising requirements volatility problems and discusses critical research issues addressed in this study.

Chapter 3 describes the research methodology adopted in this thesis. The research design, including data collection and analysis methods are discussed, and the rationale behind choosing the research design is explained. The detailed structure of the research process is also presented. This process is iterative and consists of six main phases and spans through two projects, designated *Project Release_A* and *Project Release_B*.

In **Chapter 4**, research findings from the first three phases of the research activities are presented. The study was carried out at an organisation located in Sydney. A contextual overview of this company is presented. Detailed empirical findings of requirements volatility from *Project Release_A* data is also presented and discussed, including the development of a preliminary requirements change classification. A novel application of the card sorting technique is introduced to classify requirements change requests based on the software practitioners' perspective. Detailed procedures and comprehensive findings of the card sorting exercise are presented.

Chapter 5 presents the research findings from the last three phases of the research activities. Empirical findings of requirements volatility issues from *Project Release_B* data are described and the characteristics of RV are illustrated in detail. An initial process assessment is conducted to identify the limitations of the existing change control process and to define opportunities for process improvement initiatives. Document inspections, observation, and survey techniques are used to gather information about the usage of the change control process and change request form. Findings from this initial survey are discussed in this chapter. Another survey is conducted after the implementation of a new change control process and its supporting tool. An overview of the new process and survey findings are also discussed in this chapter.

Chapter 6 provides a synthesis of information on various aspects of requirements volatility identified in the current practice. These include characterising and representing the rate of requirements volatility, identifying effective approaches to handling RV problems, and understanding different aspects of RV that were identified in the literature and those reported in this case study.

Chapter 7 summarises the key findings of this study and relates them to the research objectives and questions. Then the major contributions of this thesis are revisited, outlining the use of the change classification to manage the requirements change process better. In addition, strategies for handling requirements volatility are discussed. The challenges and lessons learned from conducting this case study are stated and outstanding research issues for future research are presented.

Chapter 2: Literature Review

2.1 Introduction

This chapter provides the essential background material and rationale to assist in understanding the objectives of the research presented in this thesis. Given that the concept of requirements volatility is still ill-defined and approaches to handle its adverse impacts are still not well understood, this chapter explores in depth the issues behind the occurrence of requirements volatility during the software development lifecycle. The content of this chapter is structured to address the following objectives:

1. To define the concept of requirements volatility
2. To identify aspects of requirement volatility during the software development lifecycle, such as change drivers and the impacts of requirements volatility on various aspects of software development projects.
3. To identify strategies or approaches to handle requirements volatility problems, from the perspectives of both theory and practice.

The intention of this review is to locate and provide evidence from the literature for assessing the significance of the problems to be resolved by this research. This review also stimulates us to explore unresolved issues for future research. The structure of this chapter is as follows:

- Section 2.2 explores challenges in software engineering, particularly the relationship between software evolution, the role of requirements engineering, and its critical importance in the software development lifecycle
- In Section 2.3, a brief introduction to requirements engineering is presented and discussed, including various definitions and the main challenges faced in the current research.

- Section 2.4 provides a comprehensive survey of the requirements volatility issue, its terms and concepts, potential causes of requirements volatility and their adverse impacts.
- In Section 2.5 potential strategies for handling requirements volatility issues proposed by previous researchers are discussed.
- Section 2.6 summarises this chapter, outlining the essential elements that need to be focused on to assist in characterising requirements volatility problems and relevant research issues to be addressed in this study.

2.2 *Software Engineering*

IEEE Standard 610.12 defines software engineering as “*the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software*” (IEEE-610.12., 1991). As a relatively new discipline, software engineering is concerned with the theories, methods and tools that are needed to develop good quality software, particularly for large and complex systems (Sommerville, 2004).

It is apparent that software has become pervasive in commerce, culture, and everyday activities (Pressman and Ince, 2000). In recent years, software has become increasingly important in our society and culture. Therefore the software engineering community continually attempts to develop software systems that will create a better future for organisations, individuals, or communities.

Software is a logical system element (Pressman and Ince, 2000) that is developed or engineered. Therefore software engineering differs from other engineering disciplines and software is different from other engineered products. The intent of software engineering is to provide a framework for building software with higher quality. Thus, special techniques, tools, methods are needed for software development.

The software development process comprises a set of activities, methods and tools that software engineers use to design, develop and maintain software along with its associated products such as programs, documents and data. Many software

process models have been proposed and they are developed to assist in the control and coordination of a real software project (Pressman and Ince, 2000).

2.2.1 Software Lifecycle Models

One of the early models, Royce's model (Royce, 1970), sometimes called the "*classic lifecycle*" or the "*waterfall model*" is a linear, systematic and sequential approach to software development. In this model, development proceeds linearly from requirements analysis through design, code, unit testing, system testing and acceptance testing, with limited feedback on the outcome of the previous phases. This model has been criticised for representing an unrealistic view of software development. The underlying assumption of this model is that all software requirements can be fully gathered upfront and frozen, which is unrealistic. As Brooks pointed out (cited in Thayer and Dorfman (1997)), the customers/users often do not know beforehand what they expect from a software system, and the requirements are almost certain to change. Furthermore, the existence of various stakeholders results in conflicting and intrinsically decentralised requirements.

The **rapid prototyping** model was introduced to overcome the problem of lack of feedback and proper iteration in the *waterfall model*. The *prototyping* model provides a reduced functionality or limited performance version of a software system in its development (Marciniak, 2002). This approach uses an operational system to define software requirements and allows developers to rapidly construct early versions of software systems that users can validate. The user validation can then be used as feedback to refine the emerging system specifications and designs. However, prototyping can also be problematic for two main reasons: 1) the users/customers often see a prototype model as a working version of software, when in fact it is not; and 2) prototyping has a tendency to encourage informal design methods which may introduce more problems (Pressman and Ince, 2000). Despite these problems, the *prototyping model* is an effective paradigm for software engineering to assist customers (and developers) in understanding requirements. Management of changing requirements is essential in this model.

Software evolves throughout its life to address demands for distributed applications and complex systems. There seems to be growing confidence in software engineering practices to move away from the traditional waterfall lifecycle model toward iterative, evolutionary software process models.

The **Incremental** model was introduced to overcome the shortfalls of the waterfall model. It combines elements of the waterfall model with the iterative process of prototyping (Pressman and Ince, 2000). This model is meant to develop a software system in stages (incrementally), which allows the developers to learn from each versions of the software. Each linear sequence (requirements analysis, design, code, and test) produces a deliverable, referred to as an '*increment*' of software. Similar to the *prototyping model* and other evolutionary models, the incremental process model is inherently iterative. However, the *incremental model* actually delivers part of the final product with each increment. Managing changes to requirements is also essential in this model, to address changes derived by user feedback following each increment that may lead to increased customer demands or even confusion.

The **Spiral** model was designed to include the best features of the waterfall model, prototyping, and incremental models, and includes the element of *risk analysis*. The *spiral model*, originally introduced by Boehm (1988), consists of consecutive risk-driven iterations of incremental phases. Similar to the *prototyping model*, an initial version of the software is developed, and repetitively refined in response to customer feedback. However, the development of each version of the software is carefully designed and a source of risk is continuously identified. The *spiral model* is a realistic approach to the development of large scale, complex systems and software, because software evolves over time. As the process progresses, the developers and customers gain better understanding and respond to risks at each evolutionary level. The inclusion of risk analysis is intended to evaluate each version of the software to determine whether or not development should continue.

It has been recognised that software must evolve to address continuous changes in the real world or become less useful (Lehman and Belady, 1985). The evolution of software is driven by several factors including changes in software/system

requirements, business rules, market demand, work environment, and government regulation. In recent years, the engineering of good software for large, complex systems must address the demands for distributed applications on heterogeneous platforms, because many organisations are now wide area, heterogeneous, and geographically distributed. It is apparent that this situation requires software to be adaptable to dynamic changes of the environment.

In the software development context, whatever lifecycle/process model organisations use, requirements will always change and should be managed.

2.3 Requirements Engineering

Requirements Engineering (RE) is now acknowledged as a novel discipline within software engineering. As a relative new discipline, requirements engineering is regarded as one of the crucial steps in the software development process that identifies, validates and documents an important aspect of software engineering: *requirements*.

2.3.1 The Concept of a Requirement

A requirement is a capability that stakeholders want or a statement of a system service or constraint. It is a description of how the system should behave or a document that describes the customer's problem and needs to be addressed in software development process (Sommerville and Sawyer, 1997, Kotonya and Sommerville, 2002).

The IEEE Std. EEE-610.12 (IEEE-610.12., 1991) defines the term 'requirement' as:

“(1) a condition or capability needed by a user to solve a problem or achieve an objective,

(2) a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed document,

(3) a documented representation of a condition or capability as in (1) or (2)”

This definition reflects that a requirement represents a mixture of problem information or real-world needs to be resolved, statements describing the behaviour of a system, and design constraints. The definition also indicates requirements as functions that the software or system must perform, but not how they must be implemented. Requirements are the basis for a sign-off contract for every project, planning, risk management, testing readiness, and change control.

It is widely acknowledged that requirements are defined in the early stage of a project development lifecycle. However, the analysis of requirements spans across the entire development lifecycle because requirements evolve and new requirements or changed requirements can be introduced at any point of the project lifecycle.

There are two main types of requirements that can be elicited from the stakeholders: *functional* and *non-functional* requirements. Functional requirements are *requirements that specify a function that a system or system component must be able to perform* [IEEE-610.12, 1991]. These functional requirements specify what the software has to do. They are traceable to a specific source, often to use cases or business rules. Non-functional requirements are *requirements which are not specifically concerned with the functionality of a system but place restrictions of the product being developed* (Kotonya and Sommerville 2002). Non-functional requirements are mostly quality requirements that stipulate how well the software should do what it has to do.

In a simpler definition, Davis (2005) defines a requirement as “*an externally observable characteristic of a desired system*”. This definition is attractive because of its simplicity and conciseness in highlighting the ‘extended’ observability of a requirement and that it is a characteristic of a system that someone ‘desires’.

2.3.2 An Overview of Requirements Engineering

Despite being an immature discipline, requirements engineering (RE) has attracted much attention from academia and software engineering practitioners. The terminology used to describe what constitutes requirements engineering has evolved substantially over the years. However, there seems to be a lack of agreement on the

definition of requirements engineering. In the classic software development lifecycle, which is based on the *waterfall* model for software development [Royce, 1970], RE is considered as a single phase of the early software development lifecycle in which a specification is produced to represent what customers want. However, this restricted view of the requirements engineering role has evolved.

Nowadays, requirements engineering plays an important role throughout the development of software systems (Pohl, 1997, Hull et al., 2002). This reflects the evolving role of requirements engineering from its traditional role as a front end towards the essential activity of change management within the software lifecycle in which a variety of participants or stakeholders involved.

Several definitions of requirements engineering have been given in the literature. The existing definitions differ in terms of the focus being discussed or investigated. In terms of outcomes, RE is defined as a relatively new field that concentrates on all of activities related to discovering, documenting, and maintaining a systems requirements document (Sommerville and Sawyer, 1997). They further state that a requirements document “*should tell a designer everything he needs to know to satisfy all stakeholders – but nothing more*”. By this definition, RE deals with a specification produced to address stakeholders’ needs and is a document for designers to work from. In contrast, with respect to the representational, social and cognitive aspects involved in the requirements specification process, Loucopoulos and Champion (1989) define RE as “*a systematic way of developing requirements through an iterative process of analysing a problem, documenting the resulting observations in a variety of representation formats, and checking the accuracy of the understanding gained*”. In a broader view of the context, Jarke and Pohl (1993) define RE as a process of “*establishing visions in context*” in order to improve a common understanding of RE. Based on an information system perspective, they divide the *context* into four worlds: the *system*, *usage*, *subject*, and *development* worlds. This four-worlds framework indicates that the stakeholders involved in the RE process will have different backgrounds, skills, and knowledge. It is claimed to be a very useful mechanism in managing change.

In a status report, Hsia et al. (1993) describe RE as one of the most crucial steps in the process of creating a high-quality software product. They further characterise RE as

“all activities relating to:

- ♦ *identification and documentation of customer and user needs;*
- ♦ *creation of a document that describes the external behaviour and the associated constraints that will satisfy those needs;*
- ♦ *analysis and validation of the requirements document to ensure consistency, completeness and feasibility, and satisfy evolution needs.”*

This definition covers aspects of the identification or elicitation, specification, and validation of requirements.

Similarly, Zave (1997) defines a more comprehensive definition of RE for software systems:

“Requirements engineering is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behaviour, and to their evolution over time and across software families.”

This definition also covers many aspects of requirements engineering: i.e. ‘real-world goals’ or the motivations of ‘why’ and ‘for what’ a software system is developed. This represents gathering information from all relevant business goals and needs (Jarke and Pohl, 1993). Zave’s definition also highlights the importance of ‘precise specification’ that represents essential activities for analysing and validating requirements. This definition also covers one of the central problem of requirements engineering, i.e. “*evolution over time...*”.

An interesting definition of RE from an information system perspective characterised by Bubenko (1993) is that RE can be seen as “*the area of knowledge concerned with communicating with organisational actors with respect to their visions, intentions, and activities regarding their need for computer support, and developing and maintaining an adequate requirements specification of an information system*”. This definition emphasises managerial, organisational, social, and technical issues and problems of RE.

A summary of the above definitions indicates that although they differ in focus, recent definitions cover more aspects of requirements engineering than older ones. They share some common aspects, such as defining what the customer wants. It is also widely accepted that only *what* a system should do should be determined in a requirements specification.

Although most requirements engineering researchers focus on different aspects of requirements engineering, there is no general agreement as to which subset of activities constitutes requirements engineering. The ranges of activities that are reflected in the above definitions and reported in the literature include: identification, elicitation, feasibility study, modelling, prototyping, analysis, representation, specification, documentation, verification, validation, inspection, negotiation, and management. Some authors distinguish the four most important steps in requirements engineering process as requirements analysis, requirements modelling, requirements negotiation, requirements validation, and/or requirements management, while other authors combine some of those steps and only acknowledge two main steps: requirements development and requirements management.

Requirements engineering is a multi-disciplinary, communication-rich and human-centred process (Zowghi and Paryani, 2003). This involves a variety of stakeholders, multiple viewpoints, organisational issues, and technical complexity, which make requirements engineering an inherently broad and open-ended research area.

Requirements engineering is thus a process of identifying, analysing and validating customer needs for a system. The primary product of the RE process is a requirements specification, which is a formal document referred to as the *Software Requirements Specification* (SRS). This document should clearly and precisely state *what* a system should do, but not *how* it should do it.

In summary, the definition of RE defined by Sommerville and Sawyer (1997) is most closely aligned with this thesis. Gathering, documenting, and maintaining a systems requirements document are important activities in today's complex and distributed environment, particularly managing changes to agreed requirements.

2.3.3 Challenges in Requirements Engineering

There is growing recognition of RE as a relatively new field and an important area in software engineering research and practice. It is believed that a successful RE process plays a crucial role for software development companies producing high quality, reliable software. There are still many problems faced by current software development organisations in improving quality, reducing cost and dealing with changes. Thus, these problems challenge requirements engineering researchers to keep improving requirements engineering techniques.

The results of this review also emphasise problems and challenges that still exist in the current requirements engineering field. Over the last three decades software requirements have been recognised as a real problem. In their investigation on software requirements problems across many different projects, Bell and Thayer (1976) confirmed that software requirements are indeed a real problem. Based on the assessment of requirements problems reports, they showed that incorrect, incomplete, and unclear requirements are the most common types of requirements problems detected. In the context of safety-critical systems, requirements problems such as missing, misunderstood, and erroneous requirements are significantly associated with safety-related software faults (Lutz, 1993). The popular large scale survey report conducted by The Standish Group International (The Standish Group, 1994) also confirmed that lack of user involvement and incomplete and changing requirements specifications are the major factors that cause software project delays and cost overruns. The challenge we have here is to improve RE techniques for developing and stating software requirements, and to develop an effective mechanism to challenge safety-related software errors (Bell and Thayer, 1976, Lutz, 1993).

Another major problem that has been discussed in the literature is that software requirements always change. It has been reported that changing requirements cause considerable problems during the software development process (Curtis et al., 1988). It is assumed that software can be seen as a representation of parts of the world. However, the world keeps changing and user needs are also constantly changing, therefore software system must evolve and software requirements must change too.

Controlling requirements change while software is under development or during system testing is an important and difficult task because of the constant changes in work environment and stakeholder/user needs (Curtis et al., 1988, Lutz, 1993, Hull et al., 2002). The challenge here is that effective change management is needed and should be integrated with requirements change management.

Other important RE problems include lack of communication between software developers and users/customers, i.e. communicating complex changes (Harker and Eason, 1993), communication within development teams, i.e. team misunderstanding of requirements or a fact about the system causes many safety-related software errors (Lutz, 1993).

Many researchers have attempted to solve RE problems and have proposed mechanisms or approaches that have been academically tested, but often are not relevant to the problems in practical settings (Bubenko, 1993). This is a challenge for researchers, to conduct more empirical research that will produce solutions to address problems in the practice of software development.

In summary, some major problems in RE include: (1) problems with the 'requirements' themselves, i.e. the quality of requirements specification; (2) problems with evolving customers'/users' needs; (3) communication; and (4) lack of empirical research in practice. The existence of these problems in the real world challenges the requirements engineering community to address and resolve them and to narrow the gap between what is needed in practice and what is produced in research.

Although other requirements engineering problems may be equally as important, in current situations where competitive strategies and technologies are evolving rapidly, requirements change is a major problem and organisations have to deal with it. Thus, effective strategies to handle this problem are urgently needed. This thesis is primarily concerned with requirements change problems or requirements volatility. Previous major research in requirements volatility issues is surveyed and critically reviewed. Detailed descriptions of requirements volatility, its concepts and measures, and its multiple aspects are presented in the next sections.

2.4 Requirements Volatility (RV)

2.4.1 Definitions

According to The Macquarie Dictionary, ‘volatility’ is defined as *the quality or condition of being volatile in various senses, lack of steadiness, and light or changeable in mind* (Macquarie Dictionary, 2002). In the context of software engineering, volatility is acknowledged as a measure of the likelihood that parts of an application will change during the software lifecycle (Pressman and Ince, 2000). With respect to software requirements, volatility can refer to lack of stability or frequent changes in requirements during the software development lifecycle.

The notion of requirements volatility has been widely used in the software engineering literature and has been considered an undesirable property in software development projects. It is used to represent changes to requirements. Other similar terms have also been used, including *scope creep*, *requirements creep*, *requirements instability*, and *requirements evolution*. Although the term requirements volatility has many aliases, they share a common understanding about the underlying concept of requirements volatility that is associated with frequently or rapidly changing requirements. They only disagree in the attributes of the requirements volatility measures.

Costello and Liu (1995) in their characterisation of requirements engineering metrics, identify requirements volatility as ‘changes to requirements’ together with the reasons for changes. Other researchers refer to a similar concept in their investigations, describing requirements volatility as the count of requirements changes, such as addition of new requirements, deletion of existing requirements, and/or modification made to existing requirements (Costello and Liu, 1995, Malaiya and Denton, 1999, Pfahl and Lebsanft, 2000, Armour et al., 2001, Bowen et al., 2002). In addition, Hyatt et al. (1996) define requirements volatility as the rate of requirements change over time.

Requirements change also occurs during software maintenance. Stark et al. (1999) define requirements volatility as changes to an agreed set of requirements.

They identify three types of changes: additions to delivery content, deletions from delivery content, and changes in the scope of agreed requirements.

Another concept of requirements volatility is associated with the evolution of requirements in the context of software product lines (Savolainen and Kuusela, 2001). The authors emphasise two major aspects of requirements volatility: *the probability of future change in requirements* and *the vagueness of requirements*. This definition only formulates requirements volatility aspects based on the importance of the requirements for different products.

Change attributes have been mentioned in most of the definitions described above, which cover addition, deletion, and modification of requirements. 'Changes' to requirements can be represented as any updates to a given requirements specification. There are two types of updates, *atomic* updates and *composite* updates. Atomic updates are defined in two forms: *additions* and *deletions*. A composite update is defined in terms of any sequence of additions and deletions (Zowghi and Nurmuliani, 1998).

Since software requirements are derived from constantly evolving customers' needs and business environments, the task involved in defining, analysing, and validating software requirements is characterised by uncertainty. Requirement uncertainty¹ is often used in relation to requirements volatility from the information system viewpoint. Requirements uncertainty comprises three attributes (Nidumolu, 1995):

- Requirements instability, the extent of changes in user requirements over the course of the project;
- Requirements diversity, the extent to which users differ among themselves in their requirements;
- Requirements analysability, the extent to which the process for converting user needs to a set of requirements specifications can be reduced to mechanical steps or objective procedures.

This definition covers the important aspects of requirements volatility such as attributes of the users, the applications, and the analysts/developers.

¹ In this study requirements uncertainty and requirements volatility are used synonymously

Based on Nidumolu's concept of requirements uncertainty, Lane and Cavaye (1998) extend the term of requirements volatility as the potential for change and the uncertainty of software requirements. This definition comprises four dimensions of requirements volatility: the potential for change, requirements instability, requirements diversity, and requirements analysability.

Zowghi et al. (1998, , 2000, , 2002) in their long-term study of requirements volatility, adopt Nidumolu's concept to characterise the detailed impacts of requirements volatility. These definitions treat requirements volatility as a multi-dimensional construct, representing the total project requirements volatility as the sum of the project's scores of those dimensions.

Furthermore, Zowghi et al. (1998, 2000) distinguished two types of requirements volatility:

- *pre-SRS RV*, which refers to the volatility of requirements during the early phases of requirements engineering (i.e. elicitation, elaboration, analysis, negotiations, modelling of requirements) and before the Software Requirements Specification (SRS) has been completed and signed off,
- *post-SRS RV* which occurs during the later phases of software development (i.e. design, coding, testing, and deployment) and after the SRS has been completed.

The first type of RV is necessarily constructive because in these stages it is expected that the domain and scope of the system are being fully explored and examined, while the second type is possibly destructive because of its adverse impacts on the productivity of the software development process, scope creep and the quality of final product.

In summary, the problems of requirements volatility will not be well understood or resolved if each researcher has a different concern in the attributes of requirements volatility, particularly in the 'change' attributes. Does RV consist of additions and deletions? Or does RV only consist of either additions, deletions, or modifications? Alternatively, does RV mean a combination of all these change types? The implications of these attributes are relevant in any software development process. In summary, we can distinguish two streams of RV concepts. First, requirements volatility can be expressed in technical terms as the rate of requirements change (i.e.

additions, deletions, and/or modifications), or the total number of requirements changes over the period of the project's lifecycle. The rate of change in fact is driven by actual data on requirements changes introduced at particular periods. Second, requirements volatility can be viewed as a multi-dimensional construct with respect to the characteristics of the software development process or the 'high' or 'low' level of the frequency of changes. The use of these operational concepts of RV is dependent on the type of research operation conducted.

Similar to Costello and Liu's definition, in this thesis, Requirements Volatility (RV) is defined as *the tendency of requirements to change over time in response to evolving needs, such as user needs, business goals, organisation, or environment.*

2.4.2 Requirements Volatility Measures

Costello and Liu (1995) introduced the concept of requirements volatility metrics. Their metric is the **total counts of requirements change** (i.e. additions, deletions, and modifications of requirements) in each specification over a given time interval (e.g. per month). Costello's metric for RV can be used as an indicator for the stability of requirements in the systems over a given time interval.

Hyatt and Rosenberg (1996) define their RV measure as a count of the number of changes divided by the number of total requirements. Similarly, Stark et al. (1999) also define a comprehensive metric of RV during the maintenance phase:

$$TRV = 100 \frac{Added + Deleted + Changed}{\# of Requirement(baseline)}$$

Stark's RV value can be greater than 100% if more changes occurred than the number of requirements originally planned.

With respect to software development characteristics, Lane and Cavaye (1998) and Zowghi et al. (2000, 2002), identified two significant dimensions of RV based on principle component analysis: requirements instability and requirements diversity. This measure is based on the project managers' opinion on the fluctuation of requirements change during development lifecycles in their companies. However, it

does not cover fine-grain requirements change data, as it only indicates the potential level of requirements volatility rate.

The level of requirements volatility can be expressed as a percentage. The definition can be instantiated in two ways (Butcher et al., 1999):

With respect to a time period. The level of volatility can be calculated for a release, for a time interval (weekly, monthly, or quarterly), or for a development lifecycle.

With respect to the change type. The classification of requirements change should be developed in order to best measure the causes and impacts of requirements volatility. The requirements volatility measures can meet the definition of a ratio scale, which has a proper zero value, 0/total requirements = no volatility.

In this study, the operational definition of RV is related to the metric as follows:

$$RV(\%) = \frac{100}{N\Delta t} \sum requirementsChange(Addition + Deletion + Modification)$$

where:

N is the total number of requirements at a period of time Δt ,

Σ is the summation of requirements changes, i.e. addition, deletion, and modification of requirements over a period of time Δt .

In summary, the operational definition of requirements volatility can be represented as the ratio of requirements change (addition, deletion, and modification) to the total number of requirements for a given period of time. The rate of requirements volatility can be either represented as a percentage of the number of requirements or a total number of requirements change over time.

2.4.3 Potential Causes of Requirements Volatility

In this section of our review, we survey and discuss existing literature that investigates potential sources of RV. We categorise the sources of RV into the aspects of: (a) user/customer needs (b) domain understanding, (c) Communication problems, (d) Organisational complexity (e) Business Environment and (f)

Technology changes. This list is by no means exhaustive, there may be other sources of RV that are not captured in this thesis.

2.4.3.1 User/Customer Understanding

Users' needs change when their knowledge increases as development progresses. Most customers or stakeholders have difficulty in expressing what they want from a system. They rarely understand the complexity of the development process, so they frequently request changes in the requirements (Curtis et al., 1988, Christel and Kang, 1992). This phenomenon has also been identified by Bowen et al. (2002), who state that end-users' poor understanding of system requirements contributes to the volatility of requirements.

Since requirements are the product of the contributions of many individuals who have conflicting needs and goals, there will be difficulties in reaching consensus about requirements (Curtis et al., 1988, Christel and Kang, 1992, Nidumolu, 1995, Nidumolu, 1996a, Nidumolu, 1996b). Specifically, when there are different needs among multiple customers or changing needs of single customer, disagreement among stakeholders on agreed requirements results.

2.4.3.2 Domain Understanding

Misunderstanding the application domain can also trigger fluctuations in requirements. A study by Curtis et al. (1988) found that the development team's limitation in understanding the application domain has resulted in performing incomplete analysis of the requirements.

Jones (1996) suggested that the causes of requirements change may vary among different types of domains. For example: changing requirements for systems software and commercial software may be due to market needs or competitive pressures; while changes to requirements for military software may be driven by changes in mission requirements or hardware platforms.

2.4.3.3 Communication

Poor communication between users, customers, stakeholders, and developers can result in software that does not meet users' expectation. Curtis, et al (1988) indicated that communication and coordination problems occur across project levels and development phases in large software projects. This is not surprising for a large and complex environment. An empirical study by Chudge et al. (1996) also confirmed that poor communication is a primary problem during the requirements change process. In their survey-based research, Zowghi and Nurmuliani (2002) showed that effective communication could reduce the extent of requirements volatility and its impact on software development.

2.4.3.4 Organisation Complexity

Organisational complexity contributes to requirements volatility problems. Christel and Kang (1992) suggested that organisational goals, policies, and structures may change during system development. These changes may cause more changes to the requirements of systems being developed. For example, changing the goals or priority of an organisation resulted in changing the scope of the product, therefore requirements changes.

2.4.3.5 Business Environment

Business environment changes cannot be controlled by software developer teams and clients, and trigger requirements volatility. For example, changes in government regulation, i.e. tax law, changes that are derived from business process reengineering studies, or sometimes driven by competitive pressure. These changes may result in the need to change software requirements, so that they reflect the new environment the system will operate in.

2.4.3.6 Technology changes

Another source of RV is technology changes. A new technology affects existing customers and developers. Rapid changes in technology also contribute significantly to changing requirements, particularly in developing software for multiple platforms. Software needs to be changed to address these rapid changes

2.4.4 Impacts of Requirements Volatility

Researchers in the past have empirically investigated the impact of requirements volatility on software development process. Those empirical studies span several disciplines, including Software Engineering, Information Systems Management and Business Studies. In this section, we present a brief overview of the literature review on the impacts of RV.

2.4.4.1 Project Risk Driver

Requirements volatility is considered as a main source of project risk (Hammer et al., 1998, Rosenberg et al., 1998). The impact of requirements volatility increases when the changes to software requirements occur in the later stages of the development lifecycle (Malaiya and Denton, 1999). These changes cause a ripple effect and cause additional changes to other associated areas such as design, code components, or test cases. Based on the observation and analysis of the NASA Goddard Space Flight Centre (GSFC) data, Rosenberg et al. (1998) indicate that requirements volatility may affect a verification program when modifications to requirements take place in the later stages of development. This risk has been confirmed by Hammer's finding that the verification effort cannot be planned effectively, particularly during requirements tests or design, when requirements are unstable.

A recent study conducted by Tiwana and Keil (2004) confirmed that requirements volatility is identified as one of the six key project risk drivers. Based on the evaluation of 720 projects, this study reveals that requirements volatility and project complexity are ranked as the lowest among the six drivers of project risk. It is

understandable that requirements volatility relative risk is low, because a certain level of requirements volatility can be expected to occur in software projects.

2.4.4.2 Schedule

Although it is not the only factor causing project schedule overrun, a high level of requirements volatility has a significant impact on schedule slippage. An empirical study by Stark et al (1999) analysed the historical data of 20 software releases from one organisation in order to evaluate the effects of requirements change during software maintenance releases. Stark et al. examined change data from one of the releases and found that adding, deleting, and changing features during the development phase caused a 78% increase in the schedule duration. This indicates that increased requirements volatility is associated with increase in schedule releases (overruns).

An empirical study on the impact of requirements volatility on software project performance conducted by Zowghi and Nurmuliani (2000) confirmed that requirements volatility affects the probability that the project will not meet its planned schedule and budget. This study was a survey-based on software development organisations across Australia. The survey results suggest that increased requirements volatility resulted in a decreased probability that the project could meet the planned schedule and the planned budget.

2.4.4.3 Development Effort

Requirements volatility also has an impact on software development effort. A simulation model developed by Pfahl and Lebsanft (2000) demonstrates the impact of requirements volatility on software project duration and effort during the development lifecycle. The model was developed using the Systems Dynamics (SD) method, which describes, models and simulates the behaviour of socio-economic systems (Forrester, 1971). The model was tested in a telecommunications firm and the simulation outputs demonstrated that an increased number of new and modified software requirements leads to an increased effort and duration of software project. Hence, the software project cost increased. This finding suggests that requirements

volatility has a substantial impact on the overall cost or effort for software development organisations.

With a similar purpose to the model developed by Pfahl and Lebsanft (2000), Ferreira et al. (2003) expanded the simulation model using survey data derived from 300 software project managers and requirements engineering factors were also included in the model. This simulation results show a very large potential risk of requirements volatility for software project cost and project duration.

2.4.4.4 Software Defect Density and Reliability

Requirements volatility is considered to be a cause of software defect density. Malaiya and Denton (1999) analytically examined the impact of requirements volatility on software defect density. The defect density is defined as the number of defects detected per unit size of the product or module, i.e. lines of code (Fenton and Pfleeger, 1996). They applied a mathematical formula to characterise the impact of requirements change on source code (i.e. code additions, deletions, and modification). This mathematical examination showed that changes in software requirements have significant effect on software defect density, because some of the software components need to be redesigned, deleted, or added. When the changes occur in the earlier phase of development, their impact on code is minimal. However, when a new requirement is added at a later stage (e.g. the testing stage), which implies adding new code, the impact is said to be significant because it is likely to add defects and make tests ineffective.

This study is arguably credible evidence that demonstrates the indirect impact of RV on software defect density. However, the theory needs to be validated in real software environments where the testing duration is not fixed and other factors such as defects associated with the user interface may be involved as well.

In terms of the factors that contribute to software reliability, researchers have hypothesised that requirements volatility is strongly associated with the reliability of software. Bowen et al. (2002) indicated that requirements volatility has a negative impact on the reliability of software. The reliability of any program depends on how well the software meets user requirements. The relationship between software

reliability and RV is that *ceteris paribus*² an increase in the volatility of systems requirements will result in a decrease in the reliability of software. It is claimed further that systems that exhibit high levels of volatility during the development phase are likely to have lower reliability during the operational phase. Frequent changes in user requirements decrease the reliability of systems analysis and design because specifications no longer reflect user requirements accurately (Bowen et al., 2002). This is an interesting finding that indicates that requirements volatility problems may decrease the reliability of software.

2.4.4.5 Development Productivity

Requirements volatility is hypothesized to have impacts on software development productivity. Lane and Cavaye (1998) investigated the impact of RV on effectiveness and efficiency of software development productivity and concluded that there was no direct impact of RV on these two factors. Lane's finding further suggested that factors such as project size and organization size strongly influence the impact of RV on software development productivity. In their previous work, Zowghi et al. (2000) also found no strong evidence to suggest that requirements volatility has direct impact on software development productivity (such as code quality, quality of project management and developers capability).

2.5 *Recommended Strategies for Handling RV*

Requirements change management is a vital activity for the success of a software project since software requirements evolve throughout the development process. Change management is regarded as a difficult task, particularly in an environment where many factors are involved, such as organisation policies, socio-technical issues, and/or application complexity. Managing the impacts of requirements change has attracted much attention both in academia and practice. However, effective change management is still not well understood and there is still a need for improvement, particularly for development of large, complex systems.

² Latin expression for "all other factor being equal or unchanged."

In this review, articles and research papers that focus on strategies for managing requirements change in general and requirements volatility in particular were located and assessed. The review revealed a lack of empirical evidence to show how particular strategies or approaches were developed to address or solve particular problems related to the sources of or the impact of requirements volatility. Existing strategies vary in terms of proposing a solution for a particular problem being investigated. The following sub-sections describe and discuss the existing strategies reported in the literature for handling requirements volatility problems.

2.5.1 Incremental Software Development Process

Boehm and Papaccio (1988), in an article about understanding and controlling software cost, have identified requirements volatility as an important and neglected factor that can cause software cost overruns. The *incremental software development lifecycle* is recommended as an approach to control requirements volatility. In this development lifecycle model, requirements and capabilities are allocated to a series of increments that are distinct but may overlap each other (Thayer and Dorfman, 1997). Incremental development is said to be useful when users/stakeholders request new features or requirements that cannot be implemented in the current product development lifecycle, but can be left for the next increment (Boehm and Papaccio, 1988). It has been speculated that this approach allows each increment to operate as a stable environment, and would significantly decrease the extent of requirements volatility. Other researchers seem to agree with this approach (Harris et al., 1991, Harker and Eason, 1993, Moynihan, 2000a). They suggest that an incremental delivery process should be applied to cope with changes coming from all sources, i.e. user and organisational requirements.

It is believed that with an iterative requirements engineering process, problems can be detected earlier, for instance misunderstandings between software developers/analysts and users on agreed requirements can be resolved, and immediate feedback from developers about requirements can be obtained (Harris et al., 1991, Christel and Kang, 1992, Moynihan, 2000a). In practice, however, there are not many software organisations or projects implementing this approach because

every project or organisation is different and the implementation of the system development method is dependent on the policy the organisation applies and the characteristics of the project itself.

2.5.2 Change Control Process

Customers or stakeholders will raise a change request while a system is being developed and a requirement may change at any point during the life of a software project. Out-of-control requirements change is recognised as a sign of project failure or delay (Hull et al., 2002). Large system development organisations may establish a *change control process*, intended as a means for software engineering managers to plan, monitor and control requested changes to any project deliverables including changes to requirements specification.

Some organisations may have a separate process that only handles requirements change, or it may be embedded in the change management organisation. Managing change is considered to be critical in requirements development (Hull et al., 2002, Kotonya and Sommerville, 2002). Requirements change management is the core of change management. With a change control process in place, proposed changes that may come from different sources can be effectively assessed for their impacts on project cost or schedule. Reasonable decisions to reject or allow the changes to go ahead and a careful plan to implement the change can be generated using the process. However, this approach will only work effectively when the organisation implements the change control process activities comprehensively.

2.5.3 Requirements Engineering Practices

Several good practices in the requirements engineering field have been reported, including the use of requirements engineering methods, requirements review, and requirements traceability. In recent years, a framework for effectively managing the software process has been widely used in many organisations. For example, the Capability Maturity Model Integration (CMMI) framework is a useful guide for requirements management, which is a key process area to achieve maturity Level 2

(Chrissis et al., 2003). Within this maturity level, organisations are expected to manage all changes to the requirements, maintain relationships and identify inconsistencies among the requirements, project plans, and work products, and taking corrective actions. This framework, however, does not outline *how* to manage requirements change. The following sections discuss various requirements engineering practices in the essence of requirements change management.

2.5.3.1 Requirements Analysis and Modelling

The use of defined methodologies for requirements engineering activities may reduce the extent of requirements volatility. A survey-based study of RV suggests that the usage of a defined methodology for *analysing and modelling requirements* would decrease the extent of requirements volatility (Zowghi and Nurmuliani, 2002). By following rigorous procedures in analysing and modelling requirements during requirements elicitation and validation, clear and concise software requirements can be produced to reflect users' needs. Furthermore, misunderstanding of requirements among stakeholders can be avoided. Based on his experiences in the software industry, Jones (1996) suggested that Joint Application Design (JAD), a requirements development method in which user and developer representatives work together with a facilitator to produce a joint requirements specification (Macaulay, 1996), can reduce requirements volatility by almost half. This approach assumes that users and developers will have the same understanding of requirements specification and agree to the results at the end of each meeting. However, this method is aimed at solving the differences or conflicts between software developers and customers/users; it is not directly designed to handle requirements change.

2.5.3.2 Requirements Review/Inspection

Another RE practice that would reduce the extent of requirements volatility is believed to be *requirements review or inspection*. This RE activity is the most common means for validating the requirements specification document. Performing formal reviews is very useful for identifying requirements errors or defects in the

early stage of the development lifecycle. Therefore changing requirements in the later stages can be anticipated and thus can reduce the adverse impact of RV.

2.5.3.3 Requirements Traceability

In respect to the technical perspective, particularly relating to the software requirements specification (SRS), *traceability* of requirements is acknowledged as an important approach to support the management of requirements volatility. Traceability refers to *the ability to describe and follow the life of a requirement, in both the forwards and backwards directions* (Gotel and Finkelstein, 1994). For example, 'forward-traceability' can provide the link between requirements specification and the design and implementation of components. This makes it easier to refer to the complete set of requirements that are affected by code or design changes (Kotonya and Sommerville, 2002).

Traceability is becoming an essential support mechanism to manage requirement change (Jarke, 1998), particularly during the impact analysis of a proposed change. The changes to requirements can be traced back to their origin to determine their criticality or traced forward to identify their impacts on other related software project deliverables, i.e. detailed design and test specifications. This mechanism allows software developers or managers to count and estimate the impact of changes on the project schedule or effort.

With respect to the requirements volatility problem, the requirements tracing mechanism will support the effective management of change in terms of an effective estimate of the impacted areas and scheduling work or effort. However, in practice the application of this mechanism is strongly associated with the organisation policies where it is adopted. The organisation tends to apply this approach according to their working culture. In other words, the implementation of requirements tracing mechanism in the organisation setting will be tailored to the organisation's needs. It is more common for some organisations to apply only a backward traceability rather than a forward traceability, which could also be due to the cost.

2.5.3.4 Prototypes

Building early prototypes is intended to encourage users or stakeholders to detect and move some changes to the front of the development lifecycle (Jones, 1996). It is also believed to be one of the most useful approaches to cope with a wide range of project drivers, e.g. changes to customer needs, customer's high expectations, and mission-critical systems (Christel and Kang, 1992). During requirements elicitation or validation, prototypes are useful for clarifying unclear requirements and narrowing misunderstanding of requirements between users and developers/analysts. Although prototyping is a useful approach to bridge the gap between customers' and developers' understanding, changes to requirements will always emerge in the middle of the development process and even later, during system testing. In addition, prototypes only represent the high level of software system to be built. Therefore, prototyping would only be an effective strategy for managing requirements volatility in the earlier stage of development and for a particular stable domain.

2.5.4 Effective Communication

Frequent communication between users/customers and developers/analysts during requirements elicitation and development process may help to resolve conflicts, misunderstanding, and reduce unexpected changes. In their empirical study findings, Zowghi and Nurmuliani (2002) indicate that the more frequently developers and customers communicate with each other during RE, the less volatile their requirements will be.

Communication is an essential issue during software development and change management. Communication is important, not only between customers and developers, but also between development teams. Communicating proposed changes throughout the change process is crucial to avoid change requests backlog and to effectively track the completion of approved changes. In practice, there is no specific mechanism that would effectively capture the communication problem. With respect to managing requirements volatility, communication among stakeholders involved in

change process is very important in the essence of impact assessment and the effectiveness of the change process.

2.5.5 Requirements Change Classification

Managing requirements volatility is one of the key process areas that organisations with low maturity levels have to focus on when engaged in improving their software processes. Identifying and characterising the nature of requirements changes could lead to more effective management of changing requirements (Harker and Eason, 1993, Lam et al., 1999).

Classifying requirements changes has been identified as one of the ways to improve our understanding of the nature of the changes (Harker and Eason, 1993, Lam and Shankararaman, 1998). Harker and Eason (1993) (1993) distinguish between stable and volatile requirements. They classify volatile requirements into five categories: *emergent*, *consequential*, *mutable*, *adaptive*, and *migration requirements*. These volatile types of requirements are mostly driven by changes in user needs, environmental factors, technology, developers' knowledge, and policies. Categorizing changes will help software developers analyse each type of change according to its origin and assess its impact on software development product and process.

Lam and Shankharaman (1998)(1998, , 1998) adopted a process improvement approach in developing a framework to manage changing requirements during software development. They define 'classifying change' as one of the five best practices in managing change. The classification they propose is domain specific (i.e. *Screen change*, *Report change*, and *Data change*). Other classifications of requirements changes have been defined in the literature during both software development and maintenance (Sommerville, 1996). (Lam et al., 1998)

The classifications described in most of the previous studies are in the form of high-level abstraction of changes, except for the classification by Lam and Shankharaman(1998), which is more specific and domain oriented. The classifications defined by other researchers (Harker and Eason, 1993, Sommerville, 1996, Pressman and Ince, 2000) are very useful when information about the nature of

changes (such as reasons for change and its origin) are included in the change request forms. However, problems arise if this kind of information is not available or only partly recorded in the change request forms (Nurmuliani et al., 2004a).

Managing requirements change is a multifaceted problem. One solution may be to consider adopting a more multidimensional approach. For example, Harker and Eason (1993) (1993) suggest that we should consider the nature of changes, the characteristics of the participants, and the design context as these are all important factors in the strategy for managing requirements change.

In a more specific approach to a particular project, Costello (1995) characterised requirements volatility based on *reasons for change*. The reasons attributes include *external interfaces, customer-requested enhancement, reallocation from another specification, oversight in earlier specification, change in scope, and incorrect/untrue requirement*. Based on changes that occurred at each development lifecycle, Bohner (2002) identified the causes of requirements change: *changes in customer needs, enhanced functionality requirements, changes in operational environment, increased performance requirements, deleting obsolete requirements, clarifying requirements, action items from reviews/walkthrough, design restriction, feedback from prototypes, correcting errors in requirements*.

All these classification attributes emerge to broaden our knowledge in understanding requirements volatility problems. Different classification attributes have been identified for different types of organisations or projects.

2.6 Summary and Proposed Research Directions

As a result of the literature review, a comprehensive overview of multiple aspects of requirements volatility is illustrated in Figure 2.1. As the foregoing research suggests, requirements volatility causes difficulty during the software development lifecycle. In this thesis we aim to improve our understanding and knowledge by identifying the characteristics of requirements volatility in terms of the sources of change, the impacts of change, and strategies to manage requirements change.

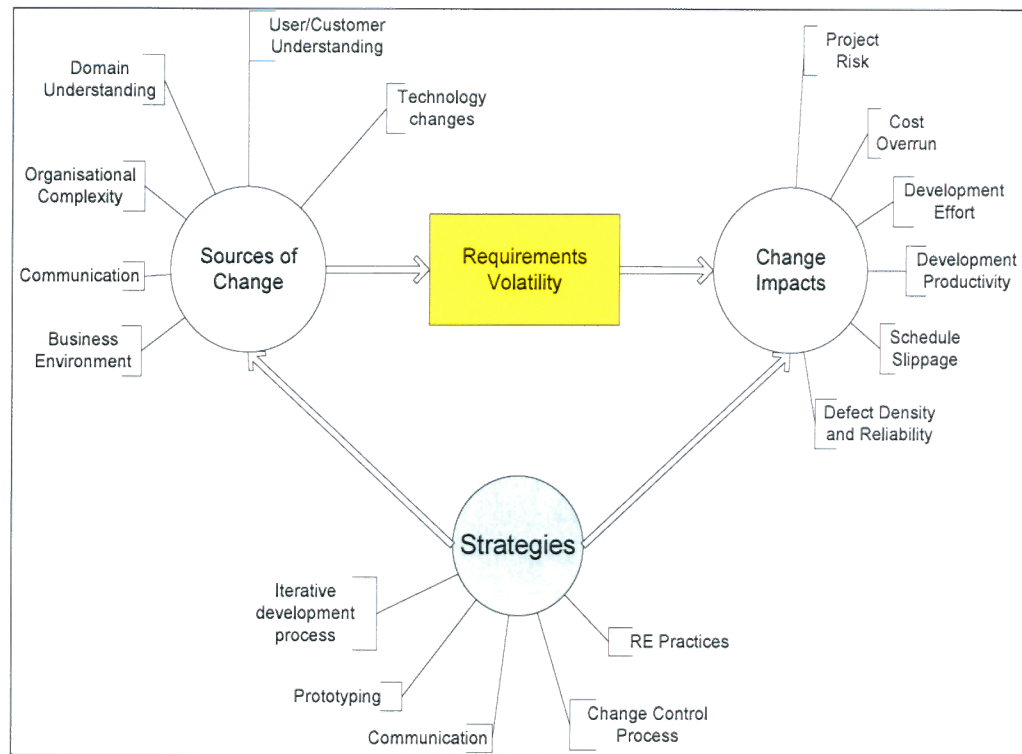


Figure 2.1 The Multiple Aspect of Requirements Volatility

2.6.1 Sources of Change

The literature review highlights the possible sources of change that contribute to the existence of requirements volatility during the software development lifecycle. These sources include evolving customers/analysts/developers' understanding of the domain, government regulation changes, technology changes and organisation complexity. However, an in-depth examination of the sources that trigger requirements volatility in a project's lifecycle is still lacking; in particular, no rigorous research has been found which shows the magnitude of each source of change. Due to the nature of software requirements that are driven by a constantly changing environment, software requirements will change no matter how well they are defined in the early stages of a software project.

Although the causes of changes in software requirements are widely known, identifying the factors that contribute to requirements volatility is still crucial in

practice. Recognising the sources of requirements volatility and investigating their levels of influence would lead us to better understand the underlying factors that trigger the problem. However, exploring these sources cannot be performed without paying attention to the impacts caused by RV. In most cases, sources and impacts of RV could be considered as mutually constituted. Hence, investigating the impacts of requirements volatility is also an essential aspect for studying this complex phenomenon.

2.6.2 Change Impacts

Our analysis of research to date shows that most previous studies focus on the impacts of requirements volatility in isolation. There are several adverse impacts of requirements volatility on software project attributes that have been identified, including schedule slippage, cost overrun, ineffective planning for verification effort, and software defect density in the later stage of development. These adverse impacts found in the literature motivate us to pay more attention to the requirements volatility problems and to find solutions or approaches that can assist organisations to manage this problem effectively.

With respect to the research into requirements volatility impacts, there is a lack of rigorous research that provides effective strategies to manage these impacts. This research is important as it guides the software developers or managers to understand the impacts while analysing proposed change requests during the software development process. Furthermore, few studies have proposed solutions to address the underlying factors of requirements volatility and the management of its impacts. A comprehensive approach that provides a mechanism for identifying the factors and addressing the impacts of requirements volatility may become an effective solution.

2.6.3 Strategies

There is no particular strategy for managing RV found in the literature that has been specifically developed in a holistic way. This kind of strategy should be developed towards understanding the needs for change and the way to manage the

impact of RV for a particular type of project. Most of the strategies described in the literature have not been tested in practice.

With respect to the software development lifecycle, the literature advises software organisations to move away from the traditional lifecycle model, such as the waterfall model (Royce, 1970), and move towards 'evolutionary' methods, such as Incremental development, Prototyping, and Agile methods. However, there is a lack of evidence to show these methods actually reduce the extent of requirements volatility.

Other proposed strategies include improved communication channels, effective change control processes, and effective requirements engineering practices, i.e. requirements review, requirements tracing, requirements analysis and modelling. A few studies have indicated that effective requirements engineering activities could reduce requirements volatility. While communication is a common problem found in most software projects, there is still a need to develop an effective communication mechanism that can help stakeholders to communicate changes. With respect to the change control process, the majority of software organisations have established such processes. However, the effectiveness of these processes in moderating change request or requirements change request in particular is still unanswered.

In conclusion, our review of the literature has enabled us to conclude that requirements volatility is an undesirable property for software project. Requirements volatility is still a problem although there is little agreement on its characteristics in the literature. Previous studies have drawn attention to the main aspects of requirements volatility, but there are still issues to be addressed. A number of research issues have emerged as a result of this literature review.

- Changes in software requirements are inevitable and perceived as a problem in software engineering.
- The concept of requirements volatility is still not well defined in relation to the software development process, thus its impacts are still not managed well.
- There is a lack of empirical studies in the literature that capture a holistic view of the requirements volatility problem during the software development lifecycle. This view includes the extent of requirements volatility at every stage of software development process.

- An in-depth investigation into the multiple aspects of requirements volatility in practice (i.e. organisation context) is still not well researched.

The research issues listed above will form the basis of the research reported in this thesis.

Chapter 3: Research Methodology

3.1 Introduction

This chapter describes the research methodology and the data collection and analysis methods used to investigate requirements volatility issues in practice, allowing us to analyse and characterise the nature of requirements volatility, and to understand its consequences. In this chapter, the research questions are revisited and the theoretical perspective and the rationale behind the research strategy are presented. The process that describes how this research was designed and implemented is illustrated in Figure 3.1 (hourglass shape).

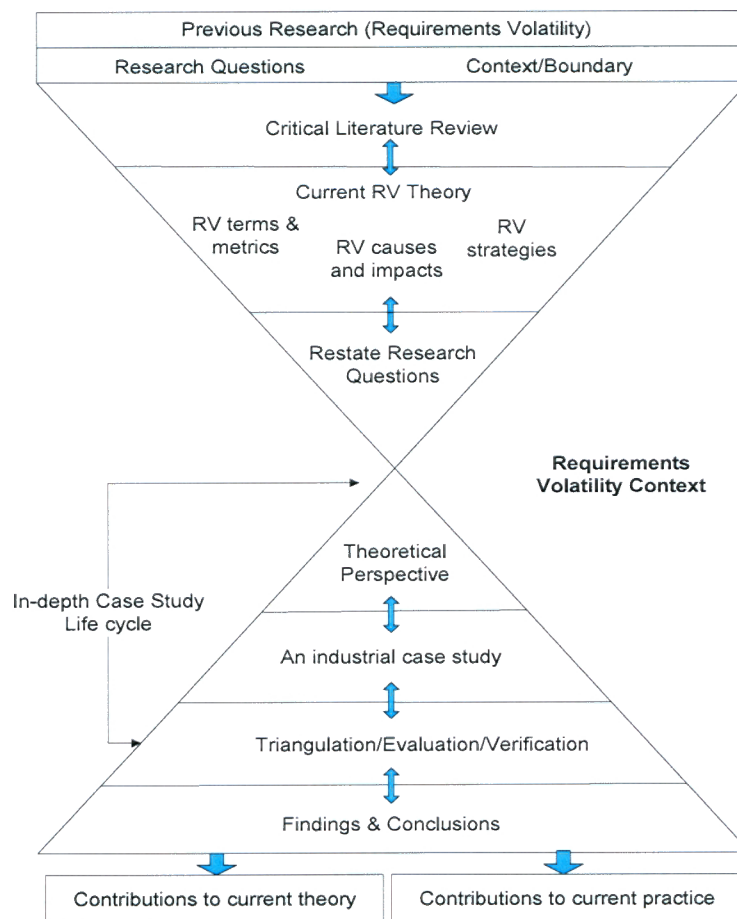


Figure 3.1. Research process at different stages

Figure 3.1 represents the sequence of research activities at different stages. The top part of the hourglass represents the initial phase of this research, in which a critical literature review was conducted in the context of requirements volatility issues during the software development lifecycle. Based on the gap analysis of the previous studies in the literature review stage, the focus of this study was narrowed and the research questions are refined and restated.

The lower part of the hourglass represents the development of a theoretical perspective, from which the study of requirements volatility is approached. Based on this perspective, the case study methodology was adopted. The rationale, strengths and weaknesses of this methodology are discussed. An industrial case study was designed and developed with various data collection methods and analyses considered. During the case study period, a triangulation of the results was carried out. Finally, the broad bottom of the hourglass represents the findings of this research that contribute towards advancing current theory and practice.

This chapter is organised as follows:

- Section 3.2 provides the underpinning theoretical perspective for this research
- Section 3.3 presents the research objectives/questions and maps them to the research strategies.
- In Section 3.4 the selection of the case study methodology is discussed.
- In Section 3.5 we present the case study design for conducting industrial case study research .
- Section 3.6 addresses the issues of reliability and validity.
- Section 3.7 considers the ethical issues encountered and the University of Technology Sydney Human Research Ethics Committee requirements are discussed.
- Section 3.8 concludes the chapter.

3.2 *Theoretical Perspective*

The review of the literature presented in Chapter 2 reveals that requirements volatility is considered to be problematic during the development of software. Its underlying causes and consequences are still not well understood. In addition, there are few empirical studies addressing the problem due to ill-defined terms and concepts of requirements volatility as well as the complexity of conducting such field studies across the entire software development lifecycle. The adverse impacts of requirements volatility on software processes and products have received a great deal of attention in the literature while its underlying causes have been discussed briefly. In order to minimise its impacts, the requirements volatility problem should be managed effectively and strategies should be developed in a way that recognises and captures the change drivers and their potential impacts.

Following an extensive review of the literature, a conceptual framework was developed using the three aspects that have been associated with the key factors for success or failure of a software project (see Chapter 2, Figure 2.1). These aspects are: 1) the sources of requirements volatility, 2) the impacts of requirements volatility, and 3) strategies to handle requirements volatility problems. The first two aspects lead to understanding the characteristics of requirements volatility problems. Requirements volatility may be triggered by internal and external factors, such as technical changes, market trends and customer needs change. The impacts of requirements volatility can include increased rework, project cost and schedule overrun. The third factor is an essential aspect related to how we handle this problem, particularly in a complex software development environment. These strategies for handling requirements volatility should be developed further to manage either the underlying factors that cause requirements to be changed or the impacts caused by the changes. However, findings from the literature review reveal that none of the previous studies have investigated the interrelationship of those three aspects in depth.

Most of the previous studies on requirements volatility (Nidumolu, 1996b, Stark et al., 1999, Pfahl and Lebsanft, 2000, Zowghi et al., 2000, Zowghi and Nurmuliani, 2002) have adopted a 'positivist' perspective. These positivist studies generally

attempt to test theory and hypotheses, quantify measures of variables, and draw inferences about the phenomenon from the sample to a stated population (Orlikowski and Baroudi, 1991). The intention of this study is not to follow the ‘positivist’ approach, i.e. hypothesis testing. Instead, we want to investigate requirements volatility issues in practice, and to explore the detailed characteristics of requirements volatility. Changes to requirements while software is being developed can occur at any time or any phase of the software development lifecycle. Thus, to improve our understanding of this phenomenon, it is necessary to conduct an in-depth investigation through continuous observations of requirements volatility as it occurs. This study was set up within software projects in an organisation and in-depth investigation was conducted throughout the project lifecycle. In order to ensure a logical and consistent research design, this research has adopted the *case study methodology* as the most appropriate methodology to address the research goal and objectives. The justification of the chosen methodology is presented in Section 3.4.

3.3 Mapping Research Objectives, Questions, and Strategies

Two main research strategies are adopted to address the research questions and objectives: a review of the literature and a longitudinal case study. The research strategies and their corresponding research questions and objectives are presented in Table 3.1 (on the next page).

The research objectives and questions drive the research process and shape the choice and use of case study methodology and methods. As Crotty (1998) points out, it is important to define the theoretical perspective, the purpose of the research and the questions that the research will be addressing to ensure the trustworthiness of the research and its findings.

Table 3.1 Research Objectives, Questions, and Research Strategies adopted

Research Objectives	Research Questions	Research Strategies
<p><i>Obj1</i> - To examine the characteristics of software requirements volatility</p>	<p><i>RQ1</i> What are the characteristics of requirements volatility?</p> <p><i>RQ1a</i> - What are the sources of, reasons for, and types of changes that contribute to requirements volatility?</p> <p><i>RQ1b</i> - What are the impacts of requirements volatility on software projects?</p>	Review of Literature and Current theory
<p><i>Obj2</i> - To establish the status of current theory in the management of requirements volatility and gaps in current research</p>	<p><i>RQ2</i> - What potential strategies are useful for managing requirements volatility?</p>	
<p><i>Obj3</i> - To understand the way an organisation manages requirements volatility throughout the software development lifecycle.</p>	<p><i>RQ3</i> - How does an organisation manage requirements volatility?</p> <p><i>RQ3a</i> - What sources of, reasons for, and types of changes that contribute to requirements volatility? and: what are the impacts of requirements volatility on software development projects?</p> <p><i>RQ3b</i> - What strategies does an organisation use to manage requirements volatility in practice?</p>	Longitudinal Case Study
<p><i>Obj4</i> - To identify limitations of requirements volatility management in practice.</p>	<p><i>RQ3c</i> - What are the current limitations of the strategies?</p>	
<p><i>Obj5</i> - To establish gaps between the current theory and current practice on requirements volatility management</p>	<p><i>RQ4</i> - What gaps exist between current theory and current practice?</p> <p><i>RQ4a</i> - Are there any differences between the characteristics of requirements volatility described in the current theory and those identified in the current practice?</p> <p><i>RQ4b</i> - What are the differences between the strategies outlined in the current theory and the strategies that organisations apply in practice?</p>	Literature Review and Longitudinal Case Study
<p><i>Obj6</i> - If necessary, to extend the current theory by proposing effective strategies for the management of requirements volatility.</p>	<p><i>RQ5</i> - Can current theory be extended to accommodate the imperatives and needs of current practice?</p>	

3.4 The Case Study Methodology

The case study methodology is one of the research strategies strongly associated with qualitative research, in which a researcher explores in-depth an activity, a process, or a phenomenon; collects detailed information using a variety of data collection procedures over a prolonged period of time, from multiple sources, within its real-life setting (Stake, 1995, Yin, 2002, Creswell, 2003). The case study focuses on detailed contextual analysis of a limited number of events and their relationships.

Case studies have been used for many years across various disciplines, such as medicine, education, psychology, sociology, law, and anthropology (Leedy and Ormond, 2001). Recently, case study research has been increasingly used in the fields of Software Engineering and Information Systems. Although there are numerous case study definitions, this research refers to Yin's definition:

"A case study is an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident" (Yin, 2002).

This means that understanding a complex or poorly understood phenomenon is determined by its context. Leedy and Ormond (2001) assert that the case study methodology is suitable for studying poorly understood phenomena. In this research, the case study is designed as a means of identifying, describing, and interpreting the problems of requirements volatility within the context of a software development project, which is considered as a complex environment.

Case studies are an appropriate methodology in situations where the researcher has little control over the events and where there is a contemporary focus within a real-life context. The other characteristic is that the case study requires a problem that seeks a holistic understanding of the events under investigation using inductive logic reasoning to make sense of the findings (Stake, 1995, Gillham, 2000, Yin, 2002, Creswell, 2003). Our requirements volatility study was conducted throughout the lifecycle of software development projects within a particular organisation, in which the boundaries between the requirements volatility phenomenon and organisation complexity are not clearly evident. Thus, continuous observation and

multiple sources of evidence are used with inductive reasoning to obtain a holistic understanding of requirements volatility problems and strategies to handle it.

Case studies are widely acknowledged for their applicability to the real-life context or an industrial setting and their flexibility, in which the researcher uses multiple sources and employs a variety of methods for data gathering. However, they are often criticised for their dependence on a small number of cases. It is also claimed that the lack of a scientifically rigorous method might provide no grounds for establishing reliability or generality of findings (Kitchenham and Pickard, 1995, Leedy and Ormond, 2001, Yin, 2002). Critics of the case study consider that the intense exposure to study of the case and the influence of the researcher's skill and interpretation may bias the findings (inherent subjectivity). Because case studies are typically based on qualitative subjective data, their results may not be generalisable or may only be applicable for a particular context. These issues may limit the validity of the case study findings. Another issue is the ethical considerations in association with the researcher's personal integrity, sensitivity, or personal biases while conducting the case study. Detailed issues of validity and reliability in relation to this study will be discussed in Section 3.7.

3.4.1 Case Study Protocol

In order to counter some of the limitations of the case study discussed in the literature and to ensure the rigour of the case study as a research methodology, researchers are encouraged to develop a case study *protocol* containing the procedures and rules that guide researcher to conduct the research successfully. Yin (2002) recommends the use of a case study protocol that includes:

1. An overview of the case study project (research objectives and case study topics being investigated);
2. Field procedures (credentials and access to sites and sources of information);
3. Case study research questions (the questions that the researcher must keep in mind during data collection);
4. A guide for the case study report (outline and format for the report).

The case study protocol is important and acts as a major component in asserting the reliability of the research. It also keeps the researcher's focus on the research goals and objectives. In this thesis, the first element of the case study protocol is related to an overview of the requirements volatility study, which includes the purpose of the case study and issues being investigated, as described in Sections 3.1 and 3.2 in this chapter. The field procedures are crucial during the data collection process. They should be properly designed since the researcher does not control the data collection environment (Yin, 2002). The field procedures developed for this study will be fully described in the Section 3.5. This procedure includes case study design, data collection and analysis methods.

The case study research questions determine the types of data to be collected, their possible sources, and the methods to be used in this study. The research questions for this study stated in Section 3.3 are important in defining the appropriate data sources and data collection methods. There are two major stages of data collection: the review of the literature and the case study.

The final element of the case study protocol is the guide for the case study report. Since by their nature, case study findings often portray a complex problem, it is important to plan the report as the case develops. In this study, the report was written in the form of research papers at every major milestone achieved and communicated through research conferences and eventually in this thesis.

3.4.2 Case Study Design

This section describes the research procedures for designing and conducting a case study in an industrial setting, and covers the type of case study, site selection, data collection procedures, and data analysis process.

Before conducting case study research, it is essential to plan and design the case study. According to Yin, there are four main components of research design that are essential for case study research design (Yin, 2002):

1. Initial case study questions – research questions framed as *what*, *who*, *where*, *when*, and *how* provide the focus of the study and determine the relevant method to be used. The research questions posed in this case study are *what* and *how*.

2. Study proposition, if any – directs attention to the issues that should be examined within the scope of the study. This study does not have any propositions, instead it highlights the issues related to requirements volatility in software projects and presents the current problems that still need to be addressed. The problem statement of this study was described earlier in Chapter 1.
3. Unit of analysis – is concerned with defining what the ‘case’ is and is a critical factor in the case study design. It could be groups, organisations or countries. The unit of analysis used in this study is software project releases.
4. Linking data to propositions and criteria for interpreting findings – this component is related to the data analysis process and the report on the case study data. In this study, the data are analysed and linked back to the theoretical framework, the research questions, and the research objectives.

This study adopts the four components of research design recommended by Yin (2002) to ensure that the research objectives are met.

3.4.3 Types of case study design and site selection

Selecting sites for conducting the research is one of the key points in designing the case study. The selection of case study sites or organisations is primarily based on the following criteria:

1. It should be a leading edge company that develops software.
2. It is committed to software process improvement.
3. The size of the company should be medium to large, i.e. more than 100 staff.

These criteria are essential to ensure choosing the relevant software organisation as a case study site for several reasons:

Firstly, this study covers software requirements volatility issues that are directly associated with a software development project. Thus, a company that develops software is required.

Secondly, this study also investigates the strategies for the management of requirements change in practice. This issue is related to the change management

process, where changes made to software project deliverables are controlled. Therefore, a company that has adopted process improvement initiatives is desirable as a case study site because the company must have established software processes to follow, i.e. a change control process and a requirements development process. In addition, requirements management is one of the key process areas covered in the software process improvement approaches (Chrissis et al., 2003).

Finally, the medium to large size of the company is essential since the size of the company could determine the way the company implements a process improvement program.

In this study, a Global Development Software (GDS³) organisation was chosen as the case study site. The characteristics of GDS satisfied the selection criteria defined above, i.e. the organisation develops software and it has committed to software process improvement (SEI CMM level 2 and ISO90001-2000). The detailed description of the case study context will be presented in Chapter 4.

In designing this case study, it is also important to select a case study site that will enable the researcher to examine in depth the problems of requirements volatility. Case studies can be either single-case or multiple-case designs (Yin, 2002). Furthermore, Yin points out that a single case study is appropriate when it represents a critical case in testing a well formulated theory, when it represents an extreme or unique case, when it is a representative or typical case, when it is a revelatory case, or when it is longitudinal (Yin, 2002).

It is understood that in current situations where software products and customer/stakeholders needs are changing rapidly, the rationale for conducting this research is not only to study the characteristics of requirements volatility, but also to investigate the way software engineers manage the requirements volatility problems in the software project. These needs are satisfied by a research design that involves a *longitudinal, embedded, single case* setting (Yin, 2002).

This study adopts a *single-case design* to study the requirements volatility problems in the GDS organisation. The rationale for this is that GDS represents a particular software development organisation and can inform the case of requirements volatility problems during its software development lifecycle. This case

³ The name is fictitious to preserve anonymity

study is a *longitudinal case study*; it involves a single case study of requirements volatility over three years (November 2002–August 2005), throughout the software project duration or from the early stage until the end of software project.

Yin (2002) points out that a case study design can be holistic or embedded. The embedded case study refers to one that involves more than one unit of analysis. Since this case study covers more than one unit of analysis, i.e. two software project releases, thus this case study is classified as an *embedded case study*.

In addition, this case study can be a *revelatory case* since it was conducted at GDS where the researcher had a unique opportunity to study the dynamic process of software projects through continuous observation. During the case study period, the researcher had full access to the GDS site, i.e. documents inspections, accessed the GDS computer and project files, opportunities to talk to software developers and managers, invited to weekly project meetings and other support process meetings.

3.4.4 Data Collection Procedures

In order to address the research questions and accomplish the research objectives, this study adopts various data collection methods, as well as both qualitative and quantitative data interpretation methods. The use of multiple research methods increases the robustness of the results and strengthens the validity and reliability of the study through triangulation.

The main focus of this investigation is on examining the problems of requirements volatility in the context of a large-scale software development project and to determine what factors may contribute to overcome these problems. The case study method does not claim any particular methods for data collection or analysis. In fact the key strength of the case study method involves the use of multiple sources of evidence for data gathering and analysis (Yin, 2002). Evidence is the primary concern of the case study researcher. There are four sources of evidence used in case study research, although not all sources are essential in every case study. However, the importance of these multiple sources of data to the reliability and validity of the case study is well established (Stake, 1995, Burns, 1997, Gillham, 2000, Yin, 2002):

3.4.4.1 Documents

A variety of documents can be used in case study research, including letters, agendas, minutes, reports, files, or any documents that are essential to the investigation. These documents may not be accurate and they may have been written for a specific audience and specific purposes. However, these documents are important as another way to corroborate evidence from other sources.

A variety of organisational and project documents were used in this study, including: *change request documents, project meeting minutes, weekly project reports, software quality assurance reports, and several software process documents, i.e. software requirements specification, functional specification documents, change control process documents.*

3.4.4.2 Interviews

Interviews are one of the most important sources of case study information. They are a useful technique for gaining insights about particular issues from participants involved in the case study. There are several forms of interviews to be used: open-ended, focused and structured interviews. The open-ended interview is used when the researcher wants to gather detailed information on a particular issue from respondents, who act as informants rather than respondents. The focused interview can be used when a respondent is interviewed for a short period of time on a very specific topic. This form of interview is often used to corroborate data gathered from other sources. The structured interview is similar to a formal survey. It may involve sampling procedures and survey instruments.

In this study, interview data were gathered through *open-ended interviews*, which were conducted together with the *card sorting* technique, *informal interviews* or *short discussions* with the project members, and *formal surveys* in relation to the project members' opinion on the use of change control processes. The card sorting exercise is part of the interview technique use in this study and a detailed description will be presented in Chapter 4

3.4.4.3 Participant and non-participant observation

Participant observation provides additional opportunities for collecting data. The main concern associated with participant observation is the potential bias of the researcher as an active participant. On the other hand, a non-participant observation or 'detached' observation is very different from 'participant' observation. The non-participant observer stays detached from the activity being investigated, where the observer observes the activity from 'outside' in a specific way (Gillham, 2000, Yin, 2002).

The case study researcher can use both participant and non-participant observation. A low-key participant observation should be used in the beginning of investigation to get to know the environment of the case study site. Non-participant observation can be used later when the research issues are well in focus (Gillham, 2000). Both types of observation were used in this study, which involved observations in weekly project meetings, requirements capture and analysis meetings, and change control improvement meetings as well as informal observations made during site visits to gather data.

3.4.4.4 Archival records

Archival records could also be useful for some studies since they consist of organisational records, charts, list of names, survey data, and other related project records. This source of evidence is also considered in this study, such as email records associated with change request approvals, project development schedules, project issues and action items.

Those sources of evidence were all relevant to this case study and were carefully recorded and stored in case notes, memos, and the case study database. The case study evidence should be maintained throughout the project because it is a crucial factor for the process of describing the context and outcome of the study. Gillham (2000) recommends that *a research log* should be used during the case study investigation. There are two main things that should be included in the research log (Gillham, 2000).

- *Evidence* – it could be a discussion topic heard in a meeting room or a staff room, something the researcher has observed, a comment made to the researcher, or a meeting resolution. These things could be fragmentary, but they occur during the case study period.
- *Personal notes* – these are related to insights, ideas, the name of someone the researcher needs to contact, or activities to be done.

In this study, the *manual* kind of research log recommended by Gillham was used during the case study research. This research log facilitates the writing of the case study report.

Since extensive data are collected during the case study period, the data should be organised in a way that can be accessed easily at any point during or after the study. There are three main principles of case study data collection that researcher needs to consider for the relevance and completeness of case study data (Gillham, 2000, Yin, 2002):

1. Use multiple sources of data – it is highly recommended to use multiple sources of evidence for a good case study. The use of multiple sources is the major strength of case study research. Multiple sources allow the triangulation of evidence that improves the reliability and validity of the data and findings. Hence multiple sources of data were used in this case study.
2. Create a case study database – all types of case study data need to be organised and documented as well as case study notes (from interviews, documents, observations, etc.), tabular materials, narratives, and relevant case study documents. Data collected in this study were organised according to the project releases in the forms of spreadsheets, words file documents, diagrams, and case study notes.
3. Maintain a chain of evidence – all case study evidence needs to be linked together in the form of a chain of evidence. The links should be traceable from the initial research questions, data collected, to conclusion drawn, or from conclusions back to initial research questions. Hence, this case study also adopts the chain of evidence approach and this will be presented at the end of this chapter and in Chapter 4.

3.4.5 Data Management and Analysis

The purpose of our analysis is to identify and characterise the nature of requirements volatility and related aspects. The case study data analysis method should be consistent with the theoretical framework of the research. A further analysis of the case study results is also intended to develop effective strategies for the management of requirements volatility.

Both quantitative and qualitative data analysis methods were applied in this study with emphasis on qualitative methods. Data is analysed iteratively and cyclically enabling us to link quantitative and qualitative data. Miles and Huberman point out that making linkages between qualitative and quantitative data is essential during the study (Miles and Huberman, 1994). For example, during *data analysis* qualitative data can help a researcher to validate, interpret, clarify, and illustrate quantitative findings.

Data analysis in case study research typically consists of examining, categorising, tabulating, interpreting, and combining quantitative and qualitative evidence to address the initial research questions (Yin, 2002, Creswell, 2003). Analysing the amount and variety of case study data collected is one aspect of the case study methodology that has not been well developed in the literature (Yin, 2002). The researcher needs to develop a general data analysis strategy as part of the case study design (Yin, 2002). The analytic strategy should indicate what to analyse and for what reason (why), so as to ensure that data collection is appropriate as well as to support the evidence to be analysed. Miles and Huberman (1994) suggest three components of qualitative data analysis that can be used in such a situation:

1. *Data reduction* – refers to the process of selecting, focusing, simplifying, abstracting and transforming data that appear in case study notes or interview transcripts. Data reduction is part of analysis in a form of sorting, focusing, discarding, and organising data that will be used in drawing conclusion.
2. *Data display* – refers to the organisation of information into forms such as matrices, graphs, charts, or networks that facilitate conclusion drawing and action.

3. *Conclusion (drawing and verifying)* – involves drawing meaning from data, noting regularities, patterns, explanations, connections to existing literature, integration of concepts and building up a logical chain of evidence (Miles and Huberman, 1994).

In a more specific way, Leedy and Ormond (2001) recommend five steps of case study data analysis:

1. *Organisation of details about the case* – the case study evidence is arranged in chronological order.
2. *Categorisation of data* – clustering data into meaningful groups.
3. *Interpretation of single instances* – examining specific documents and other relevant data to identify specific meanings.
4. *Identification of patterns* – analysing the data and their interpretation for underlying themes and other patterns.
5. *Synthesis and generalisations* – constructing the overall picture of the case and drawing conclusions.

These steps of data analysis can be done during the data collection process, where initial conclusions are likely to be drawn that could lead to the collection of other data in later stages of the study (Leedy and Ormond, 2001). Since this case study is longitudinal and the researcher has little control over the events being investigated, data collection and analysis processes may evolve. For example, when new evidence unfolds, the researcher should be capable of handling it. The research design may also change but it should at all times be referenced back to the primary research questions and objectives.

The analysis protocol used in this case study was a combination of the strategies recommended by Miles and Huberman (1994) and Leedy and Ormond (2001). In addition, *Content Analysis* was also used as part of the data analysis process. Content analysis is widely used in various research fields, and can be applied to examine any kind of written or recorded communication. Content analysis is preferred as a

research tool that can be used to determine the presence of certain words or concepts within texts (Krippendorf, 2004, Busch et al., 2005).

Data analysis and data collection were overlapped to allow for flexibility in data collection procedures and to anticipate new patterns that may emerge. The sequence of the case study data collection and the analysis process are described in the next section.

3.5 *Applying the Case Study Design*

The focus of interest in conducting this study was to investigate the requirements volatility problems that potentially occur during the development of software at a particular organisation. The initial intention was to empirically measure the occurrences of requirements volatility at every stage of the software development lifecycle and to determine the sources or causes of requirements change and their impacts on software project.

As described earlier, this study adopted a longitudinal case study methodology mainly to accomplish the research objectives *RObj3* and *RObj4* and to address the research questions *RQ3* or *RQ3a* and *RQ3b*. This longitudinal case study was set up in the GDS organisation (to be described in Chapter 4) where there were two software project releases being developed, *Project Release_A* (PRA) and *Project Release_B* (PRB). In the beginning of the case study, the researcher was given full access to PRA where the project was in the development phase (i.e. design phase). GDS management recommended engaging with PRA. The case study was carried out to the end of PRA's lifecycle. During that period, the researcher built up a good relationship with the GDS personnel and collected substantial evidence, even though the investigation started in the middle of the development phase. The GDS management gave further access for the researcher to continue the case study investigation on PRB. This was a unique opportunity to study requirements volatility on two project releases with full access to most of their data.

In the period of over 36 months, the research activities of this longitudinal case study fall into **six phases** where some of the phases are interrelated. For example, findings from the activity of *Phase 2* led to *Phase 3* and *Phase 4*. The overall view of the case study research is illustrated in Figure 3.2. This figure describes each phase

of the case study research including the corresponding sources of evidence, data collection and analysis methods used.

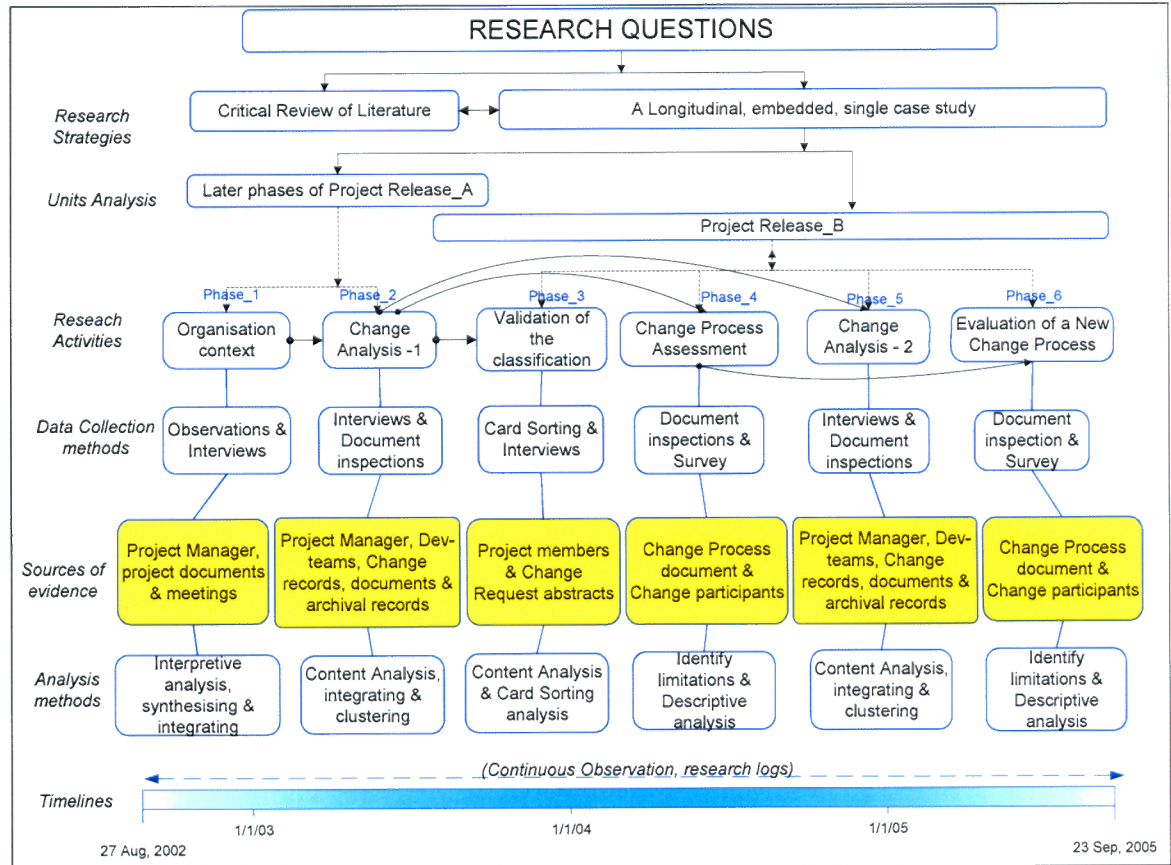


Figure 3.2. The structure of the case study research

Phase 1 – Organisational context, is the initial phase of this case study where the first research activity was carried out in an industrial setting. The purpose of this activity was to explore and gather initial information regarding the case study site, the characteristics of the software projects, and to get to know the key players in the organisation that could be contacted or could provide key sources of specific information.

The primary data collection used in this phase was informal interviews. The interviews were conducted with the key players who were involved in the project, using open-ended questions in an informal way without audio recording. The questions were mainly related to the organisational context, the characteristics of

software project releases, such as the software development process adopted, other organisation processes, software features or components being developed, the size of the project, and similar issues related to project activities. Regular meetings with the project manager were set up at least once every month to discuss and clarify relevant issues.

The source of evidence involved in this phase was project managers (PM), senior managers (SM), quality assurance lead (SQA), and several senior software developers (SD). Other sources were also included, such as organisational/project documents and process documents. The evidence gathered from discussions, interviews, meetings and document inspections was recorded in the case study notes or research log. Information gathered from several discussions with the Project Manager (PM) and other software developers increased the researcher's knowledge of the case study setting and the characteristics of the software being developed.

Phase 2 – Change Analysis-1, is the activity related to analysis of change request (CR) data for PRA. The purpose of this activity was to address the research question, *RQ3a: what sources of, reasons for, types of changes and impacts of requirements volatility are most significant in practice?* The main focus of this phase was to characterise requirements volatility problems that occurred during development of software in the PRA setting. A classification of requirements change was developed from the change request data.

Phase 3 – Validation of requirements change classification. This phase was triggered by the need to verify and validate the classification of requirements change requests developed in *Phase 2*. The other reason to conduct this activity was due to lack of information stated in the CR forms, which was also identified in *Phase 2*. The card sort technique was used to identify the software practitioners' classification of the reasons for changing requirements. Detailed procedures of the card sorting activity are described in Chapter 4.

Phase 4 – Change Control Process Assessment. During *Phase 2*, change request data were analysed. This brief assessment of the CR form led to the inspection of the Change Control Process (CCP) document established and followed by the GDS organisation. The purposes of the Phase 4 activity were to assess the change control process document and the CR form, to identify the weaknesses and

limitations of the CCP and the CR form, and to gather more information regarding the usage of the CCP based on the change participants' opinions. The findings of this assessment have been used as inputs for process improvement initiatives at GDS.

Phase 5 – Change Analysis-2, is the activity related to change request analysis on the PRB data. Data collection and analysis methods used are similar to the methods used in *Phase 2* as well as the sources of evidence. In addition to the evidence, the previously developed classifications from *Phases 2* and *3* were also included as sources of evidence.

Phase 6 – Evaluation of a new Change Control Process/Change Request Form. Drawing from the results of *Phases 3* and *4*, a new CCP was developed and introduced at GDS. Prior to the implementation of the new process, the researcher took an active role in the development of the new CCP and CR form in conjunction with the project manager. The purpose of this activity was to continue gathering information regarding the change participants' opinion on the usage of the new established change control process and CR form.

In summary, this longitudinal, embedded, single-case study was designed to explore requirements volatility issues during the software development lifecycle in depth. All evidence collected and analysed from the six phases of the case study are synthesised, integrated, verified, and built up as chain of evidence to be used in drawing conclusions of the case study. Other evidence was also extracted from the case study note and the research log.

3.6 Research Validity, Reliability, and Generalisability

Case study research deals with a complex phenomenon and generally involves multiple sources of data, which may include multiple cases, and produce large amounts of data for analysis. The reliability and validity of this research is a primary concern.

Some authors claim case studies provide little basis for scientific generalisation of findings to every possible situation and lack a scientifically rigorous method to ensure reliability and validity of research (Kitchenham and Pickard, 1995, Leedy and

Ormond, 2001, Yin, 2002). The potential for investigator subjectivity is also a concern, i.e. the skills and background of the researcher substantially influence his or her interpretation of the events.

Whilst these limitations may be true, much has been written about how to conduct case study research in a rigorous manner to produce authoritative findings. The following sections show how this study handles the issues of reliability and validity of the research.

3.6.1 Reliability

The issue of reliability in case study research is associated with the consistency of the case study results (Burns, 1997, Creswell, 2003). This thesis addresses this issue by adopting the case study protocol (Yin, 2002), particularly during the data collection and analysis processes. The researcher has engaged in a prolonged data gathering process at the case study site (2 – 3 days per week between July 2002 and September 2005). The length of this case study period helps to ensure the accuracy of the findings. Furthermore, the case study procedures are documented in detail and the sequence of case study activities and steps are clearly described in this research.

3.6.2 Construct Validity

Construct validity is concerned with the use of an insufficient operational set of measures and potential researcher subjectivity (Burns, 1997). In order to address the construct validity issue, this case study research adopted two important ways to counteract potential problems (Yin, 2002): using multiple sources of evidence (as described in Section 3.4.4) and establishing a chain of evidence that links parts together as shown in Figure 3.2.

3.6.3 Internal Validity

Internal validity is concerned with the credibility of the findings or how well the findings match reality. This study addresses the internal validity issue by considering

a number of strategies, including triangulation, re-checking with participants, and long-term observation.

1. Triangulation is a means of combining multiple sources of data, theories, methods, and observers to improve the validity of the research or for greater accuracy. Triangulation is also regarded as a means of looking at a single complex phenomenon under investigation in order to better understand the context in its multifaceted form.

Patton (2002) distinguished four types of triangulation for verification and validation of qualitative analysis, that is, the triangulation: *of methods* (methodological triangulation), *of data sources* (data triangulation), *among different analysts/evaluators* (investigators triangulation), and *of perspectives on the same data set* (theory triangulation).

With regard to the Patton's triangulation, the methods used in this case study are as follows.

- a) *Triangulation of methods*, which is related to the use of different methods for data exploration. Both qualitative and quantitative methods were utilised to examine and explore and the problems of requirements volatility.
 - b) *Triangulation of data sources*, in which different data sources are utilised within the same method. This case study used multiple data sources, i.e. documents, interviews, observations, archival records, research log, and case study notes, to improve the accuracy of findings.
 - c) *Triangulation of different investigators*, in which research findings were checked with fellow researchers and in discussions with supervisors. The case study findings were regularly checked and discussed with the researcher's supervisors and fellow researchers, i.e. through conference publications or informal discussion.
2. During the case study visits, the researcher often initiated and maintained an active corroboration on the interpretation of data with the case study participants who provided the data.

3. This case study was conducted using long term-observations, which helps the researcher to improve the accuracy of the findings and more concrete information can be obtained.

3.6.4 External Validity (Generalisability)

External validity is related to the researcher's concern whether the case study findings can be generalised or other researchers can apply the findings to their own research. Criticisms are often made that single-case studies provide little evidence for scientific generalisation. This case study is classified as a single-case study and the investigation was carried out in a particular organisation, so the findings may not be applicable to other organisation. However, the requirements volatility issues that were examined here can be found to some extent and degree in most large and complex software organisations. Although it is a single-case study, the methods and procedures used are described in sufficient and adequate detail and clearly documented, therefore they can be adopted and repeated by other researchers and practitioners in their organisations.

These findings will not be generalised to populations, i.e. all software development organisation, but they will contribute significantly to the current practice and theory. The findings can be claimed to be applicable to organisations with similar demographics.

3.7 Ethical Consideration

Before conducting this research, written permission from the University of Technology, Sydney Human Research Ethics Committee (HREC) as well as a written recommendation from GDS management were obtained (see Appendix A). This industrial case study research has complied with the HREC guidelines. Before commencing data collection, participants were invited to participate in this research. They were informed verbally about the research and the purposes of the investigation were explained. A consent form was used to obtain the participants' written consent (see Appendix B). In the consent form it was clearly stated that participants were assured of the confidentiality of their responses either from interviews or discussions,

and of their freedom to withdraw at any stage of the research without giving a reason. The participants also agreed that the data collected would only be used for research purposes, the results would be published in the thesis, conference, and journal articles without revealing the confidential issues for individuals and organisation. The researcher signed a non-disclosure agreement with the GDS to reinforce ethical consideration and to comply into the organisational requirements.

3.8 *Summary*

This chapter has described the rationale for using the case study methodology to investigate in depth the issues of requirements volatility in practice. The case study design, data collection and analysis methods have been developed according to the research strategies proposed in the early stage of this research. The multiple sources of the case study data and the techniques used to analyse them have been discussed. The application of the case study design in a real-life organisational setting requires a careful plan that is associated with the research objectives. In this study, the sequence of the case study activities has been well structured and developed throughout the case study duration.

Since this case study is classified as a longitudinal single case study design, the issues related to the validity and reliability have been considered and the techniques to address these issues have also been examined. The case study activities, which cover the investigation of two software project releases and provide fruitful findings on the requirements volatility issues , are described in detail in the next chapters.

Chapter 4: Case Study Findings – Part 1

4.1 Introduction

Chapters 4 and 5 present and describe the empirical findings of the case study to investigate requirements volatility during software development lifecycle. The study was carried out across two software project releases developed by Global Development Systems (GDS), an organisation located in Sydney. The objective of this study is to investigate the problems of requirements volatility in practice in order to better understand the characteristics of requirements volatility problems and the way the organisation deals with these problems. The empirical findings presented here provide detailed information regarding the extent of requirements volatility during the development lifecycle and factors that contribute to it.

The following research questions are addressed:

[RQ3] How does an organisation manage requirements volatility?

[RQ3a] What sources of, reasons for, types of changes, and impacts of requirements volatility are most significant in practice?

[RQ3b] What strategies does an organisation use to manage requirements volatility in practice?

[RQ3c] What are the current limitations of these strategies?

The contents of this chapter cover the first three phases of the research activities highlighted in Figure 4.1. This chapter is organised as follows:

1. Section 4.2 describes the organisational context of the case study site (*Phase_1*)
2. Section 4.3 presents the initial findings of requirements volatility study from *Project Release_A* (*Phase_2*)

3. Section 4.4 presents the validation of the requirements change classification using the card sort technique (*Phase_3*)
4. Section 4.5 provides a summary of the findings from *Phase_1* to *Phase_3*

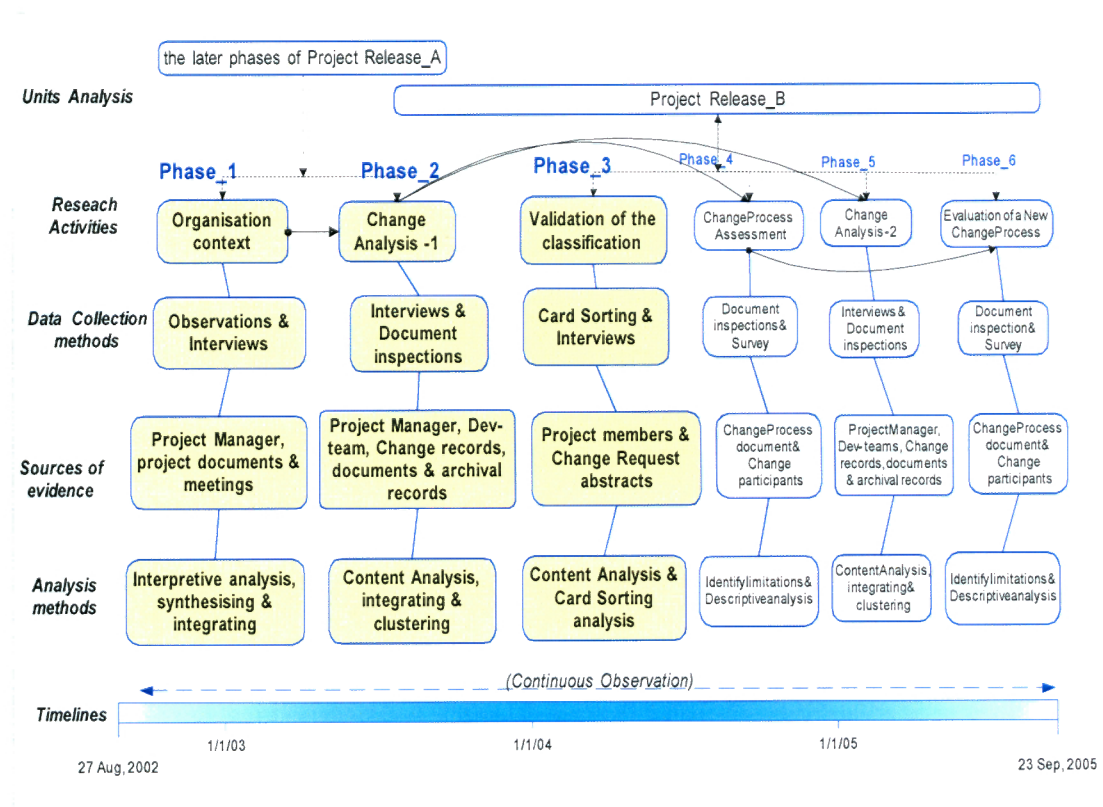


Figure 4.1 The first three phases of research activities

4.2 Phase_1: Organisational Context

This phase is the initial phase of the case study carried out in a software development setting. The purpose of this phase is to gather information about the organisational context with respect to software projects and includes the characteristics of the projects, software processes, requirements change management practices, and other related software development issues.

The case study ran from August 2002 until September 2005. Initial informal interviews and meetings were conducted with senior management in the first few months of the case study to learn about the organisational environment. During the

interviews and weekly project meetings, the researcher took extensive notes and inspected relevant project documents. These inspections led to further follow-up questions which were discussed further in subsequent meetings. The researcher also regularly attended weekly project meetings where project management and the development team discussed the progress of the project.

The following sub-sections present detailed information about the company background, requirements engineering process, project characteristics, and change management.

4.2.1 Organisation Background

This case study was carried out at Global Development Systems (GDS)⁴ an organisation located in Sydney, Australia. The GDS organisation is an ISO 9001-2000 certified and SEI CMM[®] Level 2 software development organisation that belongs to an international multi-site organization with headquarters and marketing divisions located in the United State of America (USA). The product strategy is directed from the marketing division, where the development group is located, in three Australian and one New Zealand sites. More recently, as part of global outsourcing its development group has expanded to India.

The key stakeholder groups or customers of the GDS product are scattered across the world. Direct communications between GDS and the customers is minimal. Customer needs are captured by the Business Management (BM) group or Marketing group. These customer needs are then communicated to GDS via Marketing Requirements statements (MRs). The communication between GDS and the customer groups is mainly done through BM. The detailed flow of software requirements at GDS is described in the Section 4.2.3

⁴ The company and product names are fictitious to preserve confidentiality.

[®] CMMI and Capability Maturity Model are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University. SM Personal Software Process, Team Software Process, PSP, TSP are services marks of Carnegie Mellon University

4.2.2 Software Application Overview

GDS is an engineering laboratory that develops product-line software. The product is an enterprise software application generator for mission-critical transaction applications, known as *CILKN*⁵, which is a set of computer-based tools and a supporting methodology that helps organisations build and deploy primarily transaction-intensive information systems. These application tools support industry operating systems (MCP, OS2200, Windows and UNIX). Each release may cover one or more platforms depending on the release agreement between the BM/Marketing group and GDS. The targeted end users of the software product are application designers, analysts and programmers who will use the enterprise software application to build and maintain transaction oriented business systems.

The *CILKN* tools and supporting methodology are categorised as follows:

1. **Application Developer:** – used by the end user to design, build, and develop transaction-intensive information systems that focus on business needs. The users become part of the development team, sharing ownership and responsibility for the system.
2. **Application Builder:** – helps the user during the system generation process. Using the Developer's system specification, the application builder creates all source files, transfers files between the workstation and host, and controls the compile and deployment phases of the application system.
3. **Application Runtime:** – designed for business-critical, transaction-intensive solutions. With this application, information systems take full advantage of the performance, underlying integrity, and resilient capabilities of supporting computing platforms.
4. **Enterprise Application Component Enabler:** – allows applications created with the Enterprise Application Environment to be accessed through the Microsoft standard component architecture or as a Java Bean.

The GDS organisation develops enterprise software application tools that are characterised by the delivery of a series of releases. The development cycle of each

⁵ The product names are fictitious to preserve confidentiality

release is between 12 and 24 months, in which each release is considered a standalone project in its own right. The size of each release is around 8000 KLOC, with approximately 125+ full time developers involved. During the case study, there were two project releases being developed which overlap at different release levels, for example, before one of the project releases is completed, a new project release may have started. The features for the new release are generally bug fixes and enhancements of the functionalities provided in the previous release.

4.2.3 Requirements Engineering Practices

Software requirements for new releases are instigated by changes in market conditions and specific customer requests for enhancements to the product. The requirements for a new release are gathered from multiple sources:

1. Marketing group – representing current customer needs and the potential for future customers;
2. Product strategy team – representing technology and underlying operating systems;
3. Pure engineering changes or engineering directions of the product aligned with the organisational strategy.

Communications between GDS management and their customers are done through a Business Management representative who acts as the customer representative. In addition, requirements for the current release may also arrive directly as customers' or end-users' requests through New Feature Suggestions (NFSs), which represents 'nice to have' functionalities via a User Contact Form (UCF). The UCF is the form used by the customer/end-user to report defects or bugs after the software products have been released. Figure 4.2 illustrates the NFS workflow from the customers to GDS.

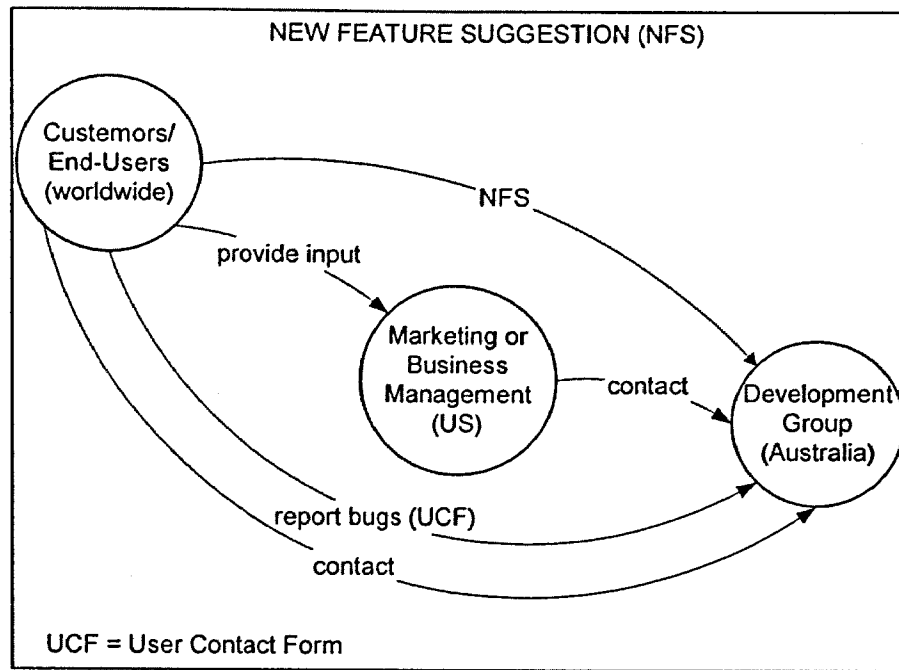


Figure 4.2 Information flow of New Feature Suggestion (NFS)

4.2.3.1 Requirements Development and Negotiation

At GDS, the requirements engineering process comprises essential activities including requirements analysis and reviews, requirements negotiation, and requirements management. There are four levels of requirements generated and managed for each release. They are structured and described as follows:

1. **Business Requirements (*level 1*)**, requirements that come from the Business Management and Marketing division. These requirements represent the market-focused objectives, the current customer needs, and new feature suggestions.
2. **Marketing Requirements (MKR) (*level 2*)**, requirements that represent product and/or service requirements to achieve the business-level requirements. These requirements statements describe the tasks the product/service needs to accomplish and non-functional characteristics of the product/service to be well accepted in the marketplace.
3. **Technical Requirements (TCR) (*level 3*)**, requirements that are developed at GDS to provide technical responses and implement the Marketing Requirements. These requirement statements are the decomposition of the submitted MKR proposed by the BM group. The set of requirements (TCR and MKR) are the

agreed set between the GDS and BM. The development and negotiation flow of the requirements is described in Figure 4.3.

4. **Internal Requirements (ITR)** (*level 4*), requirements that are developed at GDS and derived from the agreed set of TCR and MKR. These requirements are the input into the design activities.

The top two levels, Business Requirements and Marketing Requirements, are owned by the BM group, and are proposed and negotiated to GDS in the requirements definition of a new release. The other two levels of requirements (TCR and ITR) are owned by GDS. Each level is traceable to subordinate levels. These two requirement types are also classified as High and Low level requirement at GDS.

The researcher's representation of the requirements negotiation and development process at GDS for a particular release is illustrated in Figure 4.3.

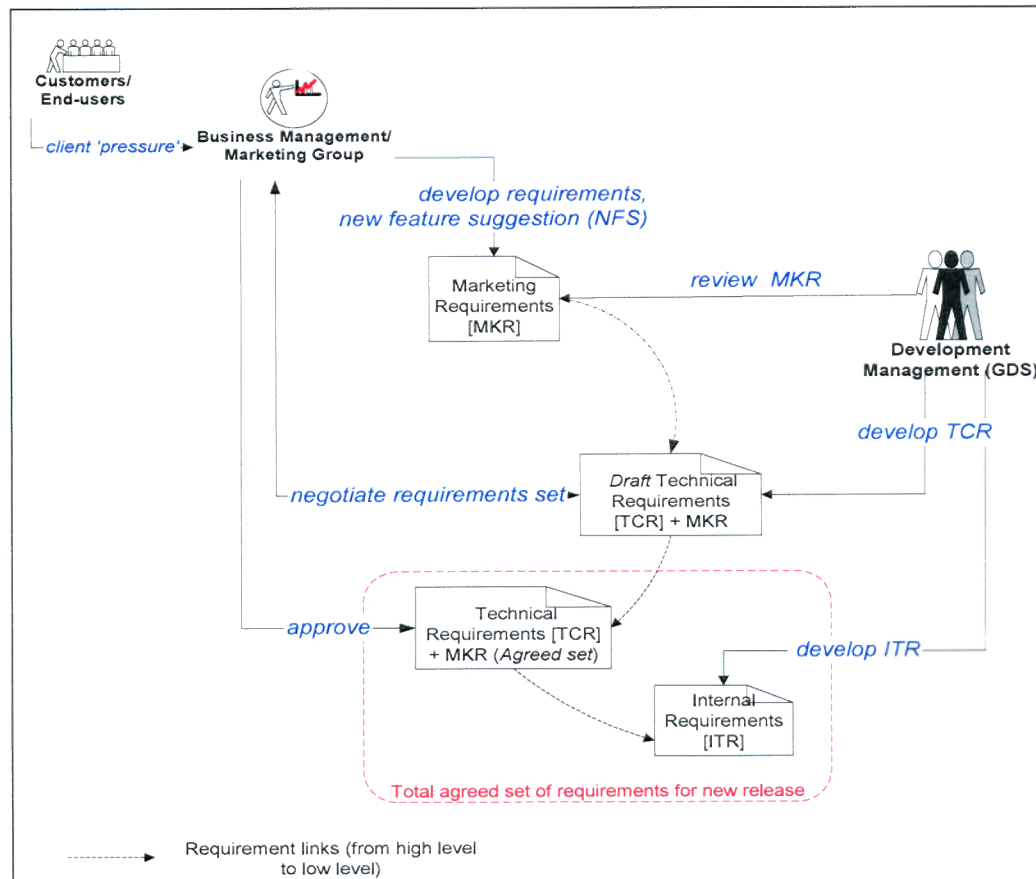


Figure 4.3 Requirements development and negotiation process at GDS

As shown in Figure 4.3, the process starts after the BM or Marketing Group has identified and gathered requirements from multiple sources including targeted customer/end-user groups that are globally distributed, business or product strategies, and the engineering team. BM develops a set of requirements and selects new features (NFS) for a new release. These requirements are then compiled into the Marketing Requirements (MKR) document and submitted to GDS. Engineering Managers at GDS then review and verify these marketing requirements. At this stage, the Engineering Managers review the submitted MKR and provide technical responses to the MKR. These technical responses are the input into the development of technical requirements (TCR). During this activity, GDS senior management initiates discussions and joint reviews between GDS and BM to capture the TCR and achieve a common understanding and prioritise the requirements. After the technical requirements are developed, GDS and BM go through a series of negotiations to reach an agreed-set of requirements (MKR and TCR). These formal meeting are conducted via telephone conferences. During the negotiation process, marketing and technical requirements may be added, deleted, or modified.

The next stage involves the decomposition of the agreed set of technical requirements at GDS. Development teams, which include Project Managers, Engineering Managers, and Team Leads at GDS conduct a series of 'requirements capture' sessions to analyse the TCR and develop low-level engineering requirements (called Internal Requirements (ITR)). Telephone conferences and Internet technologies (NetMeeting and Web video broadcasting) are used on a weekly basis during the sessions and in the communications with the remote developers in Australian sites (Sydney and Melbourne) and New Zealand. The requirements capture and analysis is described later in this chapter. All agreed Business Requirements, Marketing and Engineering Requirements (TCR and ITR) are documented and stored using Rational RequisitePro®.

® registered trademark of IBM

4.2.3.2 Requirements Management

Since 2001 the project activities at GDS have been conducted within the CMM level 2 framework. GDS has established a Requirements Management (RM) process that describes two main requirements engineering activities during the project lifecycle. The activities are Requirements Analysis and Decomposition, and Requirements Testing (validation and verification).

A. Requirements Analysis and Decomposition

This activity is aimed at capturing, analysing and verifying the customer's requirements, and decompose them into lower-level requirements, i.e. technical requirements and internal requirements, in order to establish the requirements baseline for a particular release.

The project manager is responsible for conducting this activity. Group sessions are held to analyse the requirements for each feature or component. The sessions involve project members with relevant knowledge in the particular feature or component. They are Engineering Managers, Technical Leads, Quality Assurance (QA) Lead, and representatives from the Product Information (PI) and Testing teams.

The method used during the analysis of each requirement involves drawing a diagram using Microsoft Power Point (see Figure 4.5), to identify the interrelationships that may exist in the system design as a result of implementing the customers' requirements. Once these relationships are identified, technical requirements or internal requirements are formulated and a Microsoft Excel form is used to capture newly developed requirements.

The newly developed requirements are structured according to:

1. requirement description,
2. requirement rationale,
3. test scenario.

A traceability link is created to trace each requirement to its test scenario. The requirements are then imported into the RequisitePro repository. The Requirements Specification (RS) document is created using Microsoft Word. The technical requirements are grouped into 'features' for a current release and 'components' for

the new release. Each *feature* or *component* generally (but not necessarily) addresses an agreed Marketing Requirement (MKR).

The Requirements Specification document is then subjected to team level peer review. This internal review mostly deals with the newly developed technical and internal requirements. In some circumstances the customer group may be involved in the internal (GDS) review. Once the reviews are completed and agreement with BM is reached, the RS is baselined.

B. Requirements Testing

This activity is intended to verify and validate the agreed requirements, which involves assessing the compliance of the work products with their allocated requirements. This activity is conducted by the Test Engineer who develops test plans and test scripts, and performs the tests as specified for the requirements.

C. Change Control Process

Managing changes to requirements is part of ongoing RM activities. All changes to baseline requirements are managed via the Change Control Process (CCP). This process enables project members at GDS to raise a change request to requirements and other project documents, to track and monitor the change requests to completion. This process was established by GDS to ensure that changes to requirements are approved and implemented, the impacts of changes are analysed, and that change implementations are tracked and verified to completion. The project manager is responsible for tracking the progress of the change request. A paper-based Change Request Form (CRF) is used to propose a change during software development. Electronic-mail, Microsoft Word and Excel are the tools used to support this process. Detailed findings from the assessment of this process are described in Chapter 5.

D. Peer Review Process

GDS has also established and implemented a peer review process, which is a standard process for carrying out peer reviews on any work product. The work product could be any artefact produced by a process, such as files, documents, processes, procedures, and manuals/guidelines. Relating to the software projects at GDS, the work products may include project deliverables that are under

configuration management, such as the following documents: requirements specifications, design specifications, test specifications, code sources, project plans, and processes. The peer review process is intended to help the author improve the work products by identifying issues which may be defects, weaknesses or improvements.

There are two review types: Team Peer Review (formal) and Desktop Peer Review (informal). Team peer reviews are used as a precursor to the approval of an important phase of a work product. Reviewers rely on a 'Checklist' of defects found in different types of work products. Review meetings are conducted within the team. The identified issues or defects are recorded in an 'Issue Capture Form'. Participants in the team peer review meeting include the author of the work product, project manager, feature management leads, and development team members (required and optional reviewers). At the end of the meeting, the results of the review are recorded in a 'review summary form'.

Desktop peer reviews are less formal; they only require one person besides the author to review the work product. A 'checklist' and 'issue capture form' are also used in this review type.

4.2.4 Software Project Characteristics

When the case study investigation started, there were two software project releases being developed. The first project, *Project Release_A*, was in the later stages of the development lifecycle (i.e. implementation and testing stage), and the second project, *Project Release_B* had just commenced. Although the first project had already completed the requirements capture and analysis process, we were still able to conduct the study for the following reasons:

1. All changes to requirements and other project deliverables were documented and recorded in a project repository;
2. A formal change control process was established and followed to manage all changes to project deliverables, i.e. requirements specification, functional specification, design specification, code and test specification, or project plans;
3. Most of the staff who involved in the project were still working at GDS.

Thus, complete documentation was available and provided essential sources of case study data.

Each project release has different characteristics, particularly on the product applications being developed. The following sections describe the characteristics of the two project releases studied at GDS.

4.2.4.1 Project Release_A

The aim of *Project Release_A* was to maintain the enterprise application customer base by keeping the product competitive and providing the performance and scalability needed to support the mission critical, high-volume transaction processing environment. The product of this release is based on the functionality provided by the previous release. In this release, obsolete functionalities are removed, and the interoperability of the application with the supported platforms is enhanced.

The scope of this release was to build software applications for the following operating environments (software platforms): MCP, OS2200, Windows, and UNIX. Twenty five *features* (a collection of functional requirements) were developed through the project development lifecycle, from October 2001 to March 2003. The *waterfall lifecycle* model was adopted to develop the applications with the assumption that the requirements would remain stable throughout the product development lifecycle.

As illustrated in Figure 4.4, *Project Release_A* is structured into four main phases of the development lifecycle. The phases are Feasibility – requirements analysis and negotiation, Design, Development, and Integration.

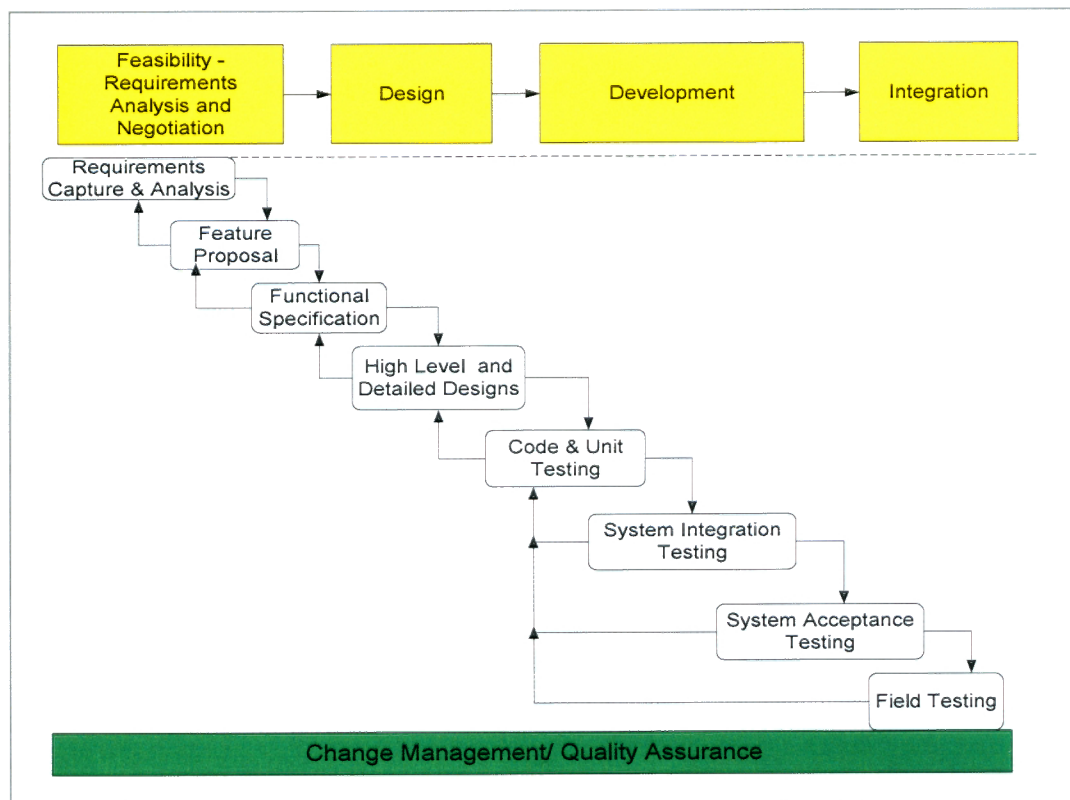


Figure 4.4 Software development lifecycle - *Project Release_A* (Source: adapted from GDS's internal document)

During the feasibility phase, product requirements (MKR, TCR, and ITR) for this release are analysed, validated, and negotiated with the customer representative. Product estimation and resource/staffing are also conducted in this phase. As shown in Figure 4.4, change management or change control and quality assurance are ongoing activities during the development lifecycle.

The project activities were conducted within the CMM framework. Improvement initiatives at GDS have been established through activities such as project planning, tracking, estimating, scheduling, and requirements management process. *Project Release_A* was regarded as a pilot project for the CMM-level 2 assessment.

Regarding requirements analysis activities (as broadly described in Section 4.2.3), requirements analysis methods and tools were used to capture and verify the customers' requirements for this release. The supporting tools in this phase are MS_PowerPoint, Microsoft Excel form, and Rational RequisitePro®. Group sessions were held to analyse and capture requirements for each feature. The 'diagram' is

used (see Figure 4.5) to facilitate the identification of all affected functionalities (green bubbles) and interrelationships that may exist in the system design.

Once the interrelationships are identified, requirements are formulated and described to address each relationship using a sentence template developed by Halligan (cited in Damian, et al. (2002)). An Excel Form (see Figure 4.5) is used to capture the elements of the sentence template. When it is completed, the requirements statements are then imported to the requirements database using RequisitePro®. The benefits of requirements engineering process improvement using the method and tools mentioned above have been reported by Damian et al. (Damian et al., 2002).

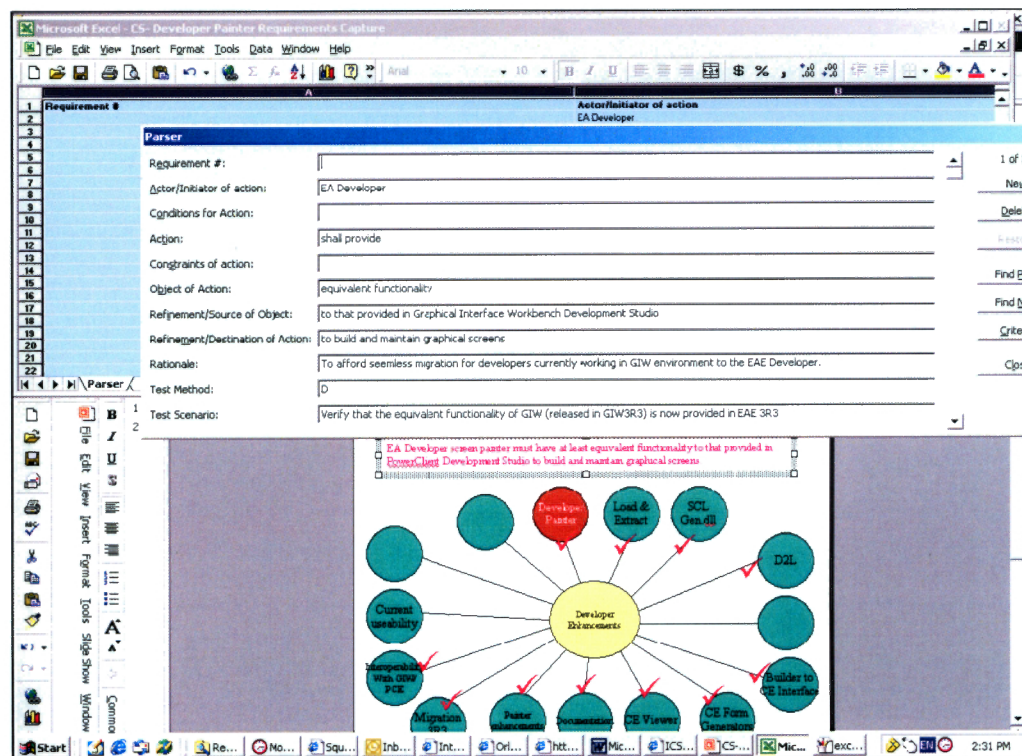


Figure 4.5 The ‘diagram’ used in the analysis of each feature and the Excel Form capturing the constructs of the sentence template (Source: Damian, et al., 2002)

4.2.4.2 Project Release_B

Project Release_B is regarded as a new product using architecture-based models. The product is based on the functionality provided by the previous release (*Release_A*). The aim of *Project Release_B* is to overhaul the application

environments of the previous releases using component technology while maintaining or improving the benefits of the current application environment systems. The reason for this change, besides the strategic direction for the product, is to improve the interoperability of the application environment systems with other systems.

The scope of this release is the enterprise software applications that can be applied for Windows platform(s) only with 12 system components developed throughout the development lifecycle. The development of this release is dependent on emerging technology, i.e. Windows.NET and Visual Studio Integration Package (VSIP).

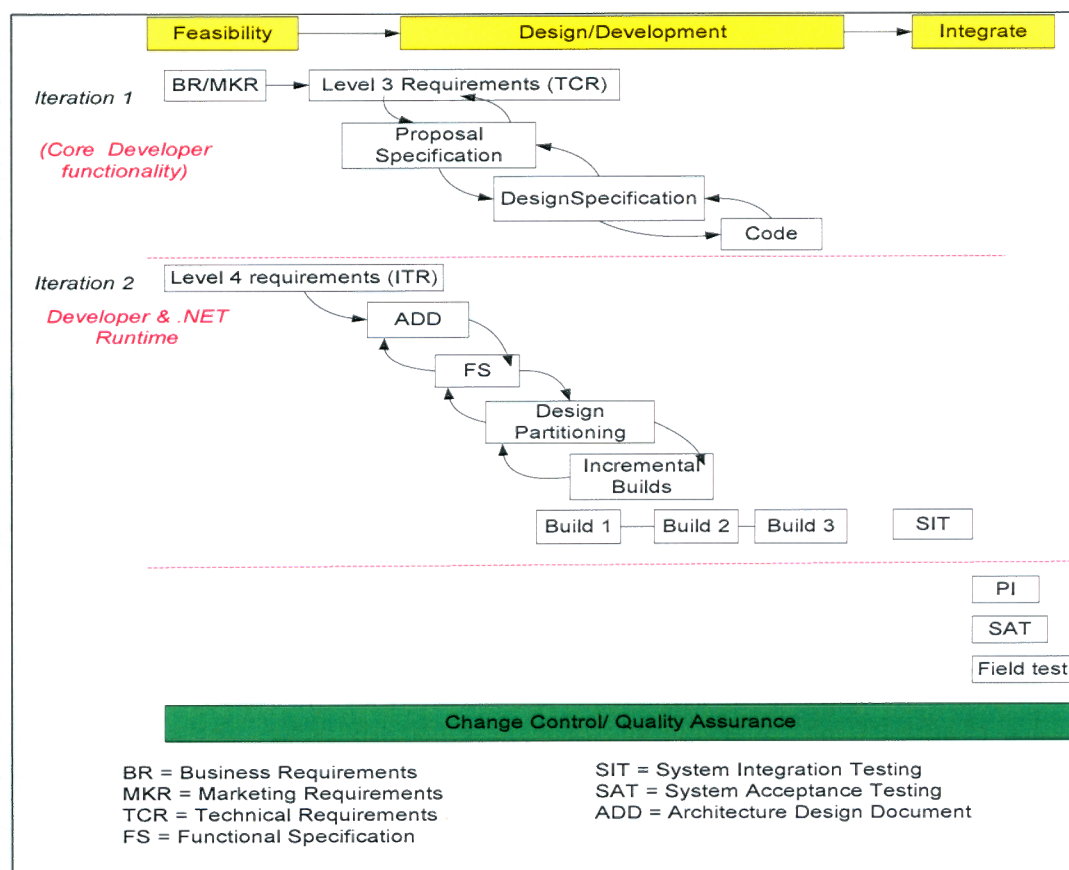


Figure 4.6 Software development lifecycle adopted - *Project Release_B* (Source: Internal GDS document)

Project Release_B provides a new Architecture Design Document (ADD) and safe passage for users of GDS's earlier product releases. A new software development methodology – *Iterative design and development* – was adopted. This methodology simply allows developing software designs, coding and unit test and

conducting systems integration through a series of iterations, each iteration being more detailed. The main difference between this project and the *Project Release_A* is in the sequence of development activities. *Project Release_A* adopted a development methodology that allowed the developers initially to produce high-level design specifications, followed by detailed design specifications, code and unit tests, and systems integration testing. According to the GDS management, the reason for using the iterative model is to allow the iteration releases to be produced and shipped to external parties and to allow software to evolve. As the iterative development approach was introduced for the first time, *Project Release_B* was chosen as a pilot project for the formalised *Iterative Design & Development Process* adopted by GDS. An indicative illustration of the *Project Release_B* lifecycle model is shown in Figure 4.6.

Initially, this release adopted the *Agile* methodology for software development (Highsmith and Cockburn, 2001) during the early phase of Iteration 1. However, based on management considerations, the *Agile* development methodology was discontinued and structured analysis was applied in the second iteration. During our regular meetings with one of the project managers, further information was elicited related to Iteration 1. He provided the explanation that the *Agile* methodology was not a suitable methodology for the projects at GDS. He further outlined the specific characteristics of GDS projects that contributed to the discontinuation of the adoption of the *Agile* methodology, including the following.

1. The GDS project team size is approximately 100 developers and is geographically distributed. Meanwhile, *Agile development has been acknowledged to only work well for smaller and collocated teams* (Boehm, 2002)
2. At GDS, communication between developers and customers is minimal and mostly conducted through customer representatives. The *Agile* methodology emphasises real time communication or face-to-face communication with customer (Boehm, 2002).

3. The GDS project develops working software, which is delivered at the end of the project duration. Agile methods emphasise working software as the primary value of progress and deliver working software frequently, with short timescale⁶.

The early phases of Iteration 2 were mostly associated with the breakdown of the high-level requirements (TCR) into more detailed low-level requirements (ITR). A 'Context diagram' (see Figure 4.7) was used to identify the interrelationships of components in the system. The 'context diagram' describes the level 4 actors (components) and sub-actors for each component. Requirements statements were formulated to address the interrelationship and were recorded in a Microsoft Excel spreadsheet. Development teams for *Project Release_B* did not use the sentence template in Excel form (as shown in Figure 4.5) to capture and develop requirements statements for reasons that were not disclosed. Based on the researcher's observations, the context diagram was designed as a web-based document that can be easily used by the teams during the requirements capture meetings. The project manager of the *Project Release_B* took part in the meetings as a moderator. The rest of the requirements analysis activities are similar to the activities described in GDS's requirements analysis process (see Section 4.2.3).

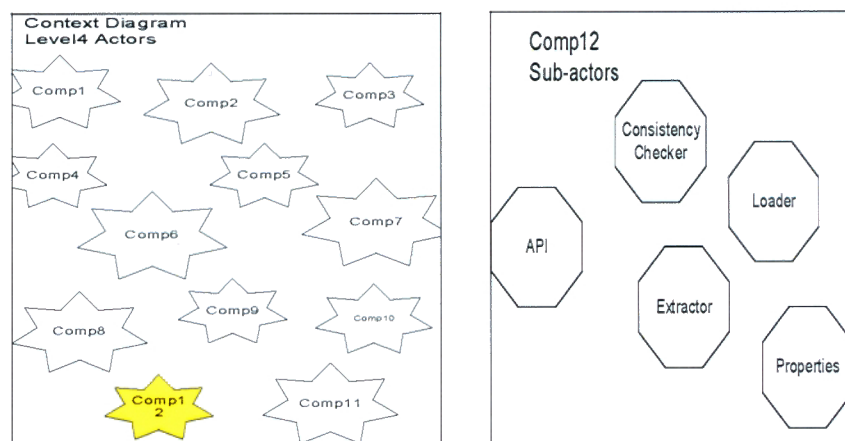


Figure 4.7 The 'diagram' used in the requirements analysis for *Project Release_B*
(Source: Internal GDS document)

⁶ www.agilemanifesto.org

4.3 Phase_2: Change Analysis_1

This section describes the research activities conducted during *Phase_2* of Figure 4.1. These activities are related to the analysis of change requests for the *Project Release_A* database. The objective of *Phase_2* is to address the following research questions: *RQ3a: what are the sources of, reasons for, types of changes that contribute to requirements volatility?* and: *what are the impacts of requirements volatility on software development projects?*

In this section, the methods used for data collection, sources of evidence and data analysis methods are described in more detail. Requirements change requests are examined, and a classification is developed to characterise requirements volatility and understand its causes and consequences. Empirical findings resulted from *Phase_2* activities are then presented.

4.3.1 Phase_2: Data Collection and Analysis Methods

Both qualitative and quantitative data are collected in this case study. The data include requirements change information, such as rationale for requirements changes ('*why-factor*'), types of requirements changes or updates (*addition, deletion, or modification to requirements*), sources of requirements changes ('*who*' requests the change or '*where*' the change was identified), and the impact of requirements changes, i.e. development effort. The other data collected are data for quantifying the requirements volatility rate, as follows:

1. total number of requirements in the project,
2. total number of requirements change (addition, deletion, and modification) over period of time (per month),
3. other change-related data, such as elapsed time to process each change request, and total estimated effort required to implement the change.

4.3.1.1 Data Collection

The sources of the evidence used in this phase were the key project members such as Project Manager (PM), Quality Assurance (QA) Lead, change participants (initiators, approvers, and implementers), CR documents and archival records, and other relevant process documents. The *Project Release_A*'s Change Requests are stored in the project repository using Microsoft Word and the summary of each CR is recorded in a Microsoft Excel spreadsheet.

The techniques used for data collection include document inspections, interviews or discussions, and non-participatory observations. Extensive inspections on the CR documents and archival records were conducted iteratively. The interviews or informal discussions were used to gather more detailed information relating to change request records. Non-participatory observation is a useful technique applied during attendance at regular weekly project meetings, requirements capture and analysis sessions, and requirements and test specification reviews. Half-hour interviews or informal discussions with the project manager and software developers were conducted at least once every week during the last six months of the project to discuss change request contents and to obtain clarification about the missing information. On some occasions after the weekly project meeting or CR document inspections, the researcher identified issues or questions that needed to be answered. The researcher then initiated informal interviews to discuss those issues. The researcher, academic supervisor, and the project manager have also conducted monthly meetings regarding the progress of this research project. The interviews were not recorded; instead, the researcher took extensive notes or wrote the results of the interviews in the case study notes immediately after the meetings were completed.

4.3.1.2 Data Analysis

Data collected during this phase were analysed using both qualitative and quantitative techniques. *Content analysis* is used as the main data analysis technique, by going through each submitted change request form, identifying substantive statements related to the requirements attributes (*the sources of, reason for, and types*

of change), clustering the data from the CR records according to those attributes, and creating lists of categories (see Appendix C). Then, a classification of requirements was constructed composing the three main attributes described earlier. This classification is a useful mechanism for further analysing the characteristics of requirements volatility. Detailed steps of data collection and analysis framework applied in *Phase 2* are illustrated in Figure 4.8.

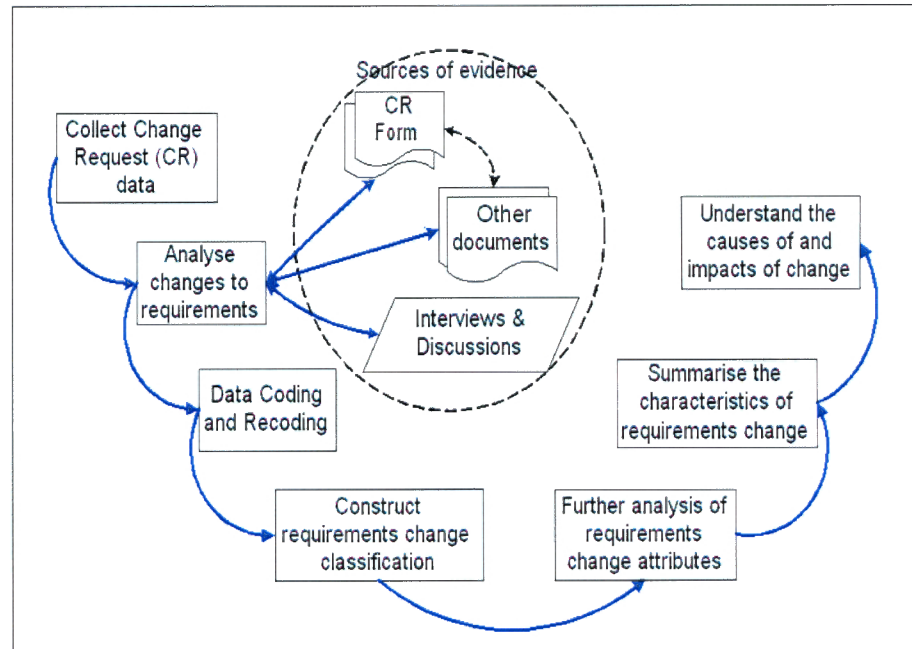


Figure 4.8 *Phase_2: Data Collection and Analysis Framework - Project Release_A*

Quantitative analysis was applied in this phase. The focus of the analysis is the measurement of requirements change, such as the change request arrival rate, the rate of requirements volatility over a period of time, a decomposition of the number of requirements changes through the change classification, and the calculation of estimated effort required to implement the changes.

4.3.2 Phase_2: Case Study Findings from Project Release_A

The empirical findings resulting from *Phase_2: Change Analysis-1* activities provide rich information on requirements volatility issues during the software development project. In the following sections the main findings of this phase such

as change request arrival rate, characteristics of requirements volatility, and a visual illustration of requirements volatility.

4.3.2.1 Change Request Arrival Rate

A change request form is used to raise and submit a change in project deliverables. The CR form template is a paper-based form. A total of 78 change requests were collected during the last six months of the *Project Release_A* (total duration 20 months). Most of these requests (86%) were related to changes to the requirements specification, functional specification, and other specifications. The rest of the change requests (14%) were related to project documents, i.e. project development plan, test plan.

The arrival rate of overall change requests during the project development lifecycle is illustrated in Figure 4.9. The two legends included in Figure 4.9 are 'Overall change requests' and 'Requirements change requests'. The average arrival rate of change requests is relatively low: approximately two to three change requests were submitted per week.

As shown in Figure 4.9, the number of change requests increased sharply from March 2002 to April 2002 when the requirements analysis phase and review process on work products, such as requirements specification, feature proposal, and functional specification documents, were being completed. During this period, most of the requests resulted in additions and deletions of requirements. This is not surprising, since the developers or engineers received feedback from requirements and functional specification reviews. The arrival of change requests decreased as the project approached the end of its lifecycle. The majority of change requests during this period related to functional and design specification changes, as the developers gained more knowledge about the product.

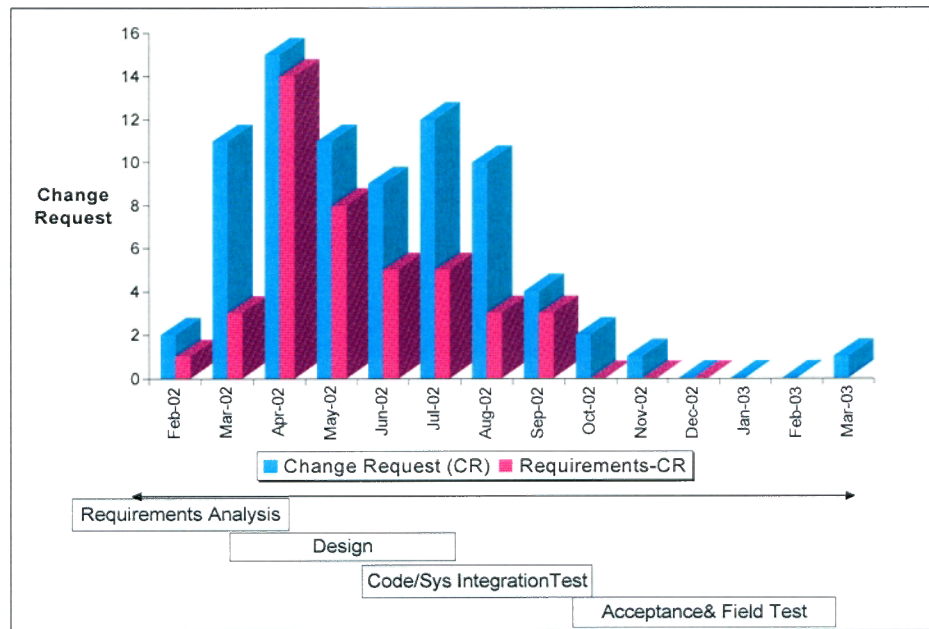


Figure 4.9 Change Requests Arrival Rate for Project Release_A

4.3.2.2 Measuring Requirements Volatility

Based on the CR records, the rate of requirements volatility throughout a development lifecycle can be quantified. As described in Chapter 2, the requirements volatility measure is defined as the ratio of the total number of requirements changes (i.e. addition, deletion, and modification) to the total number of requirements for a particular period of time (e.g. month).

Of 78 CRs, 42 were related to changing requirements. These change requests were carefully examined and evaluated. A total of 118 requirements changes were identified, in which six **high-level requirements** and 112 **low-level requirements** changed throughout the development lifecycle. Among the six high-level requirements that were subject to change, one was modified for clarity, three new requirements were added due to “functionality enhancement” and “missing requirements”, while two of the existing requirements were deleted because they were considered obsolete. Changes to the high-level requirements mostly originated from Marketing group or Business Management (in US) and GDS project management and were made in the early phases of the project’s life.

In contrast, a substantial number of low-level requirements changes occurred throughout the development process. Out of the 112 low-level requirements, 58 new requirements were added to the project, 24 existing requirements were deleted, and 30 existing requirements were modified or amended. The findings presented in the next sections are related to the low-level requirement changes. A visual view of the rate of requirements volatility for the low-level requirements is illustrated in Figure 4.10. The X-axis represents the month when the change requests were approved and integrated into the project. The legend at the bottom of the graph indicates the development phases. The early phases were overlapped because of differences in the schedules of the 25 features being developed.

The rates varied across the stages of the development lifecycle and are consistent with the arrival rate of change requests. The only high peak (16.85%) was at the end of the requirements analysis stage. This requirements volatility rate can be viewed as an indicator for the instability of requirements during the product development process.

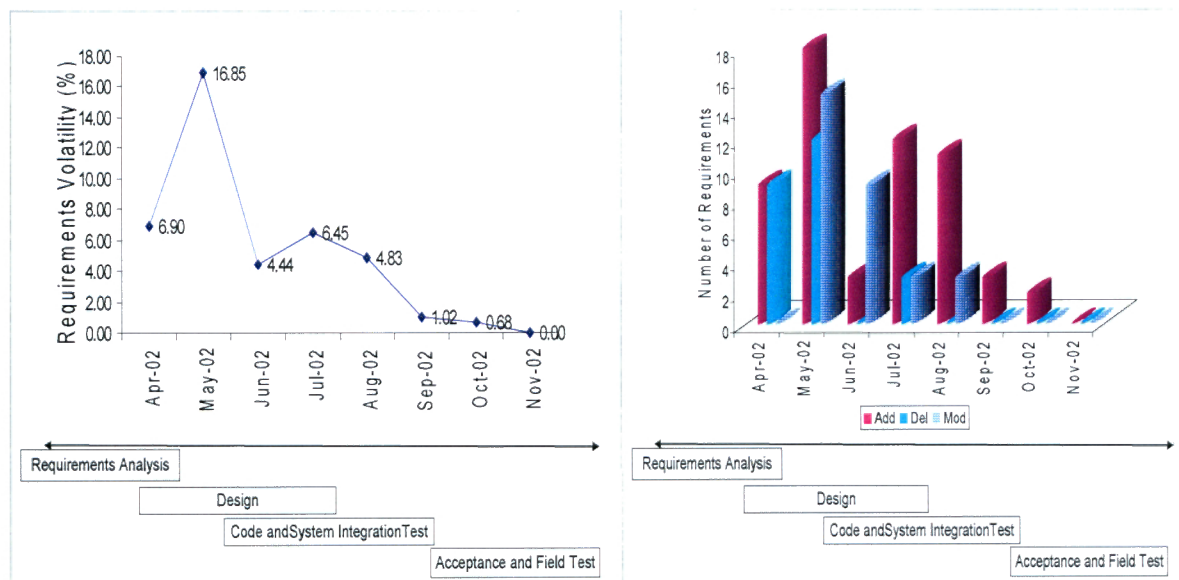


Figure 4.10 The Rate of Requirements Volatility – *Project Release_A* (Low-level requirements)

As shown in Figure 4.10 (left hand side) the rate of requirements volatility decreased substantially in June 2002, then increased slightly in July 2002. At the end of the system integration testing, the rate continues to decrease towards the end of the

development lifecycle. Overall, it is clearly shown in Figure 4.10 that requirements fluctuated in the early phases of the software development lifecycle and tend to be stable near project completion (after the field test). However, a graphical presentation of the volatility rate does not indicate the magnitude of the requirements change at a particular point of the development lifecycle, for example, the types of requirements change occurring in the later stages of development or the reason for the changes occurring at particular stages of development.

4.3.2.3 A Classification of Requirements Change

As understanding requirements volatility problems is an essential element of effectively managing the volatility impacts, we need to identify and define the characteristics of requirements volatility. Classification or categorisation is viewed as an approach to improving our understanding about phenomena of interest (Harker and Eason, 1993). In this study a classification of requirements change is developed to empirically analyse the nature of requirements change. This classification provides a useful mechanism to categorise the requirements change requests, characterise the nature of the changes, and to be used for further analysis of the change requests.

In this phase, a preliminary classification of requirements change is developed. This classification is aimed at empirically characterising requirements volatility with its three components: requirements change types (*Change Types*), the rationale for changes (*Reason Category*), and the source of change (*Change Origin*).

The classification of requirements change developed through several iterations. Change request (CR) documents for *Project Release_A* were inspected and the contents were examined. During the inspection of each CR, the three components of the classification were elicited. For instance, the change types were manually identified using keywords, such as, 'add new', 'delete', 'modify', 'amend', or 'requirements'. Then, the CR description was carefully analysed and summarised according to the purpose or the rationale of change. The origin of the CR (person or activity) is also extracted. These change attributes are then clustered into the three components of the classification.

- **Change Types**

Table 4.1 Descriptions of Change Types

Type	Description
1) Addition	Adding new requirements into the software or system being developed
2) Deletion	Deleting or removing existing requirements from the system,
3) Modification	Modifying or rewording requirements statements

- **Source of Change (Change Origin)**

A proposed CR may originate from various sources, such as people who are involved in the project (e.g. engineers, project managers, etc), as action items from the review process, developer analysis, technical discussions, and the project document. In this classification, various sources are treated the same to reflect the origin of the change. Details of *Change Origin* elicited from the CR data are as follows:

Table 4.2 Descriptions of Change Origin

1) Defect Report	A report that registers all defects discovered from previous release
2) Engineering Analysis	An analysis conducted by software engineer(s) during the development phase or the analysis that triggered by issues/problems identified
3) Project Management	Management considerations that are generally associated with the business goals and policy
4) Marketing Group	Representing market trend or current customer needs
5) Detailed Design Analysis	The analysis is not part of the design review, conducted by designers/ engineers/ analysts to improve or resolve design issues
6) Design Review	A formal peer review process on design specification documents
7) Technical Team Discussion	Irregular discussions initiated by the technical team to resolve issues or problems faced
8) Functional Specification Review	A formal peer review process on feature proposal documents
9) Feature Proposal Review	A formal peer review process on requirements specification document during requirements analysis phase
10) Customer-Support Team Discussion	Discussions between customers and support team after the release of the products

- **Reason Category**

This component is related to the categorisation of requirements change in term of reason or rationale behind the proposed CRs. The following are the list of categories elicited from the change request data:

Table 4.3 Descriptions of Reason Categories

1) Defect Fix	Correcting defects discovered from previous releases
2) Missing Requirement	Requirements that were not captured during the initial product definition or discovered after design refinement
3) Functionality Enhancement	Maintaining or managing functionality or capabilities for the product releases, e.g. technical upgrade, functionality upgrade, security upgrade
4) Product Strategy	Changes to the product packaging, installation, or licensing
5) Design Improvement	Changes to improve software design or capabilities
6) Scope Reduction (change)	Removing functionality or reducing the amount of work due to lack of resources or functionality beyond the current scope
7) Redundant Functionality	Unnecessary functionality or functionality that already exists or can be replaced by other existing functionalities
8) Obsolete Functionality	Functionality that is no longer required for the current release or has no value for the potential users
9) Erroneous Requirements	Incorrect or wrong requirements
10) Resolving Conflict	Functionality conflict or sub-functionality interferes with the integrity of other functionalities that exist in the system
11) Clarifying Requirement	Changes to requirements text or rewording requirements text, i.e. remove unnecessary words or were discovered after design refinement
12) Requirement Redundancy ⁷	Requirement overlaps or requirements are already covered by other requirements
13) Third-party software/tool influences	Requirements that cannot be fulfilled due to the availability/unavailability of a third-party software/tool, configuration/platform changes, or conformance to industry trend

Three components of the classification can be represented in a graphical view as shown in Figure 4.11. This overall view of the classification attributes clearly indicates the reasons for changing requirements as a result of a particular development activity, customer requests, or management decision. While the mapping of these attributes is a useful way to understand the big picture of why

⁷ The attribute 12 (Requirements Redundancy) is similar to attribute 7 (Redundant functionality). However, the different is in the root-cause of the changes, e.g. 'requirements' and 'functionality'

requirements changed during software development project, another way to utilise this classification is in the empirical analysis of change. The findings of applying this classification are presented in Section 4.3.3

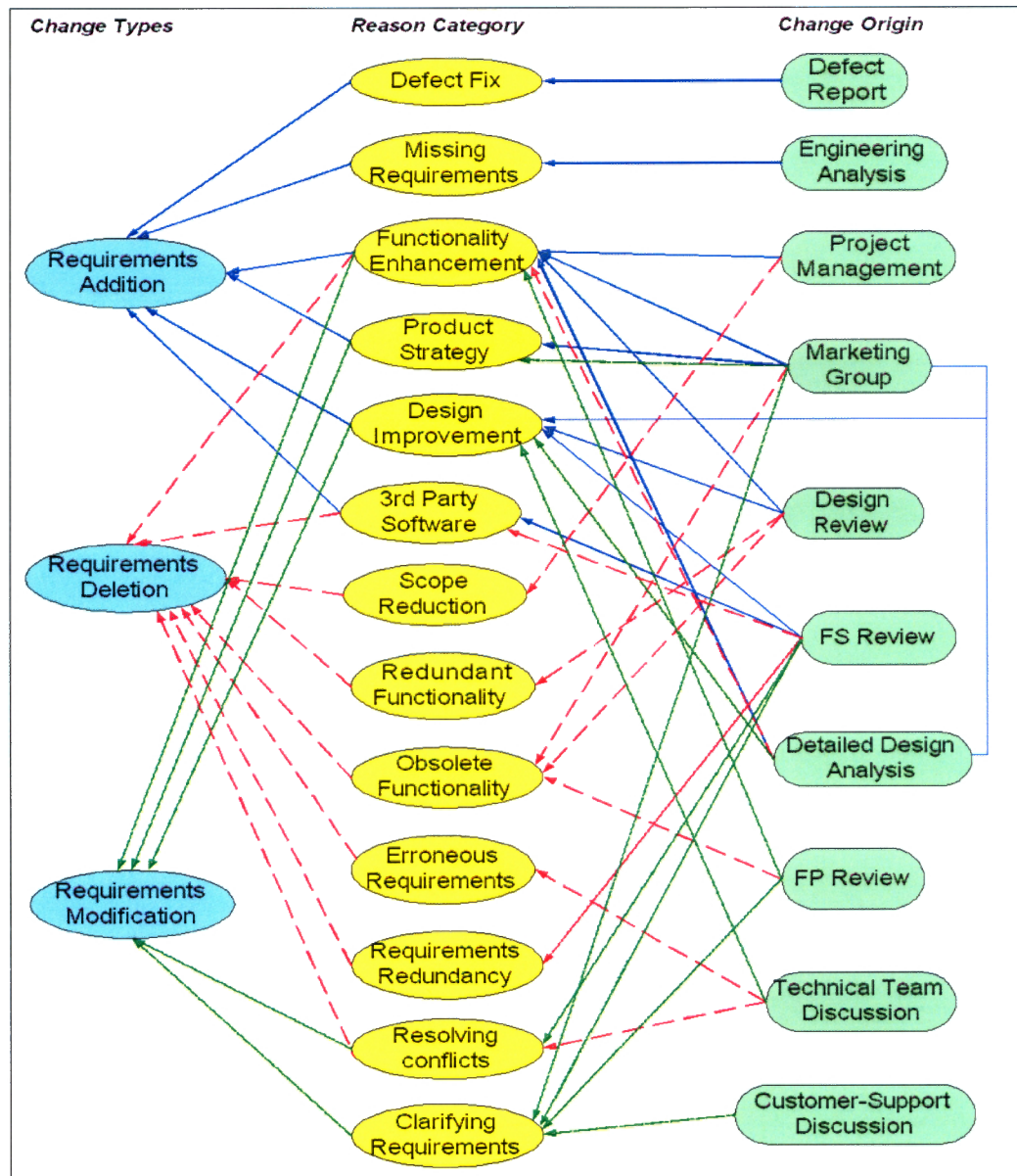


Figure 4.11 An overview of requirements change classification – *Project Release_A*

4.3.3 Phase_2: Empirical Analysis of Requirements Volatility using the Classification – Project Release_A

A classification of requirements change can be utilised further to analyse requirements volatility issues in more detail. In this section, examples of the utilisation of the change classification to understand the underlying causes of requirements change and its consequences are presented.

4.3.3.1 The Cost of Requirements Volatility

It has been reported in the literature that requirements volatility has adverse impacts on software project attributes, such as schedule and cost (Stark et al., 1999), performance (Zowghi and Nurmuliani, 2002), or development effort (Pfahl and Lebsanft, 2000, Ferreira et al., 2003). In this study, the association between RV and development effort (change effort) is part of the case study analysis and the result of the analysis is described in this section.

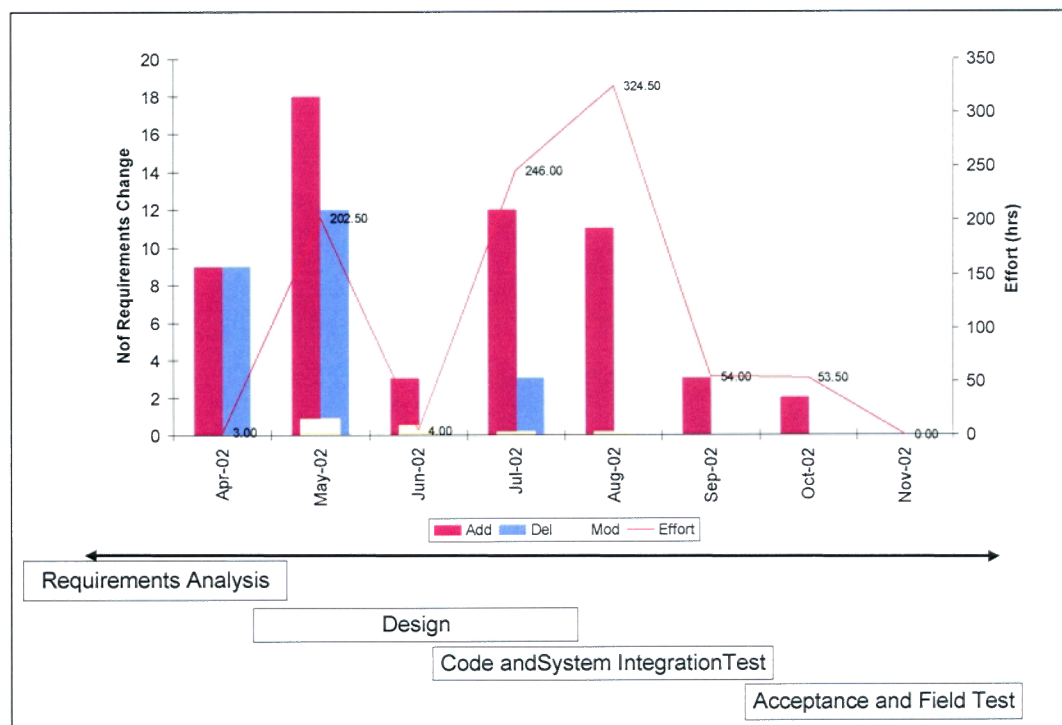


Figure 4.12 Overview of RV and the total change effort throughout the development lifecycle of *Project Release_A*

The RV metrics collected using the classification of *Change Types* (add, del, and mod) to calculate the rate of RV can be used to assess the amount of effort required to implement an individual change. The mapping between the number of requirements changes (based on change types) and total change effort spent to implement the changes every month is shown in Figure 4.12. This diagram describes the volatility of software requirements throughout the development lifecycle and the total cost to implement the changes.

As shown in Figure 4.12, the case study data reveals that the large effort shown in this figure mostly represents the effort needed for additions of new requirements. The finding suggests that adding new requirements later in the development lifecycle increases the rework to be done or new documents to be produced; thus, more effort is needed. As indicated in Figure 4.12 the total effort required to implement the changes at the later phases of the project's life seems high. This finding also clearly confirms the classic phenomenon of the cost of requirements change during software testing (Boehm and Papaccio, 1988, Malaiya and Denton, 1999). These empirical findings demonstrate that changing requirements, particularly adding new requirements, in the later phases is considered a high risk. These findings are consistent with the RV phenomenon.

Further analysis reveals that most of the effort needed between July 2002 and November 2002 related to additions of new requirements. While the rate of RV was high in May 2002, the total development effort was lower than that in July–August 2002. This indicates that although a substantial number of requirements were changed, the changes occurred in the early phases of the lifecycle.

A further use of the change classification is to analyse the cost of changes. This helps the project manager during the change implementation plan and estimation purposes. Figure 4.12 provides a history of how much effort will be spent for a given reason according to the change types.

A Pareto analysis suggests that the top five costs (high effort) were for requirement changes related to '*Functionality enhancement*', '*Design improvement*', '*Defect fix*', '*Product strategy*', and '*third-party software influence*'. Although this figure was based on the total estimated effort collected from the CR documents, it

could possibly be different from the actual effort required. The organisation (GDS) can use this information to identify improvement opportunities and as an input into strategic decision making and performance evaluation. This will be discussed further in Chapter 6.

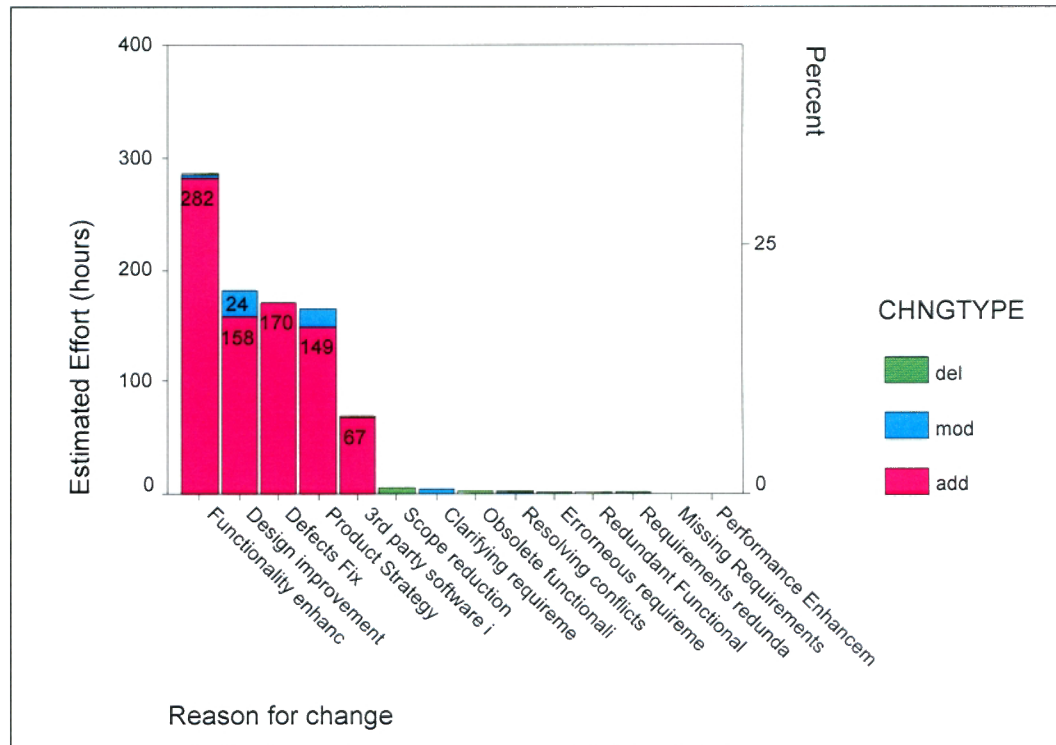


Figure 4.13 Pareto analysis of Total Estimated Effort by *Reason Category* and *Change Types – Project Release_A*

4.3.3.2 Change Origin and Change Types

Cross analysis between the two elements of the classification, such as *Change Origin* and *Change Types* against total number of requirements change is illustrated in Figure 4.13.

This Pareto chart indicates the top four *Change Origins* are “*Detailed design analysis*”, “*Design review*”, “*Project management*”, and “*Marketing group*”. It also shows that most of the additional requirements (*add*) originate from those four activities/groups. Modifications to existing requirements (*mod*) mostly originate from

“Detailed design analysis” and “Technical team discussion”, while the largest number of deleted requirements is resulted from “Design review” activities.

Other sources of requirements change, such as “Defect report” or “Engineering analysis”, are also as important as the top four sources. The organisation (GDS) can use this information to reassess their supporting software process, particularly when a large number of requirements changes are generated from the developers’ detailed design analysis.

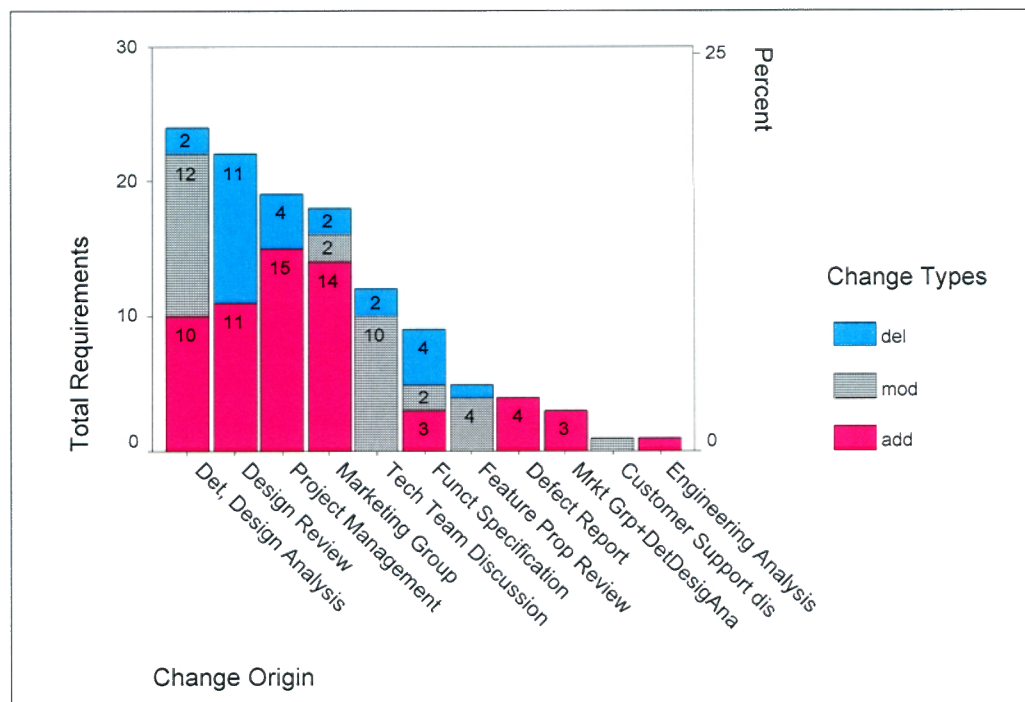


Figure 4.14 Total requirements change classified by *Change Origin* and *Change Types* – Project Release_A

4.3.3.3 Reason Category and Change Types

The next example, a mapping between *Reason Category* and *Change Types* provides an insight into the most significant reasons for requirements to be changed during the development of software project. This mapping is illustrated in Figure 4.15, which shows the highest to the lowest number of requirements changes according to the reasons for change.

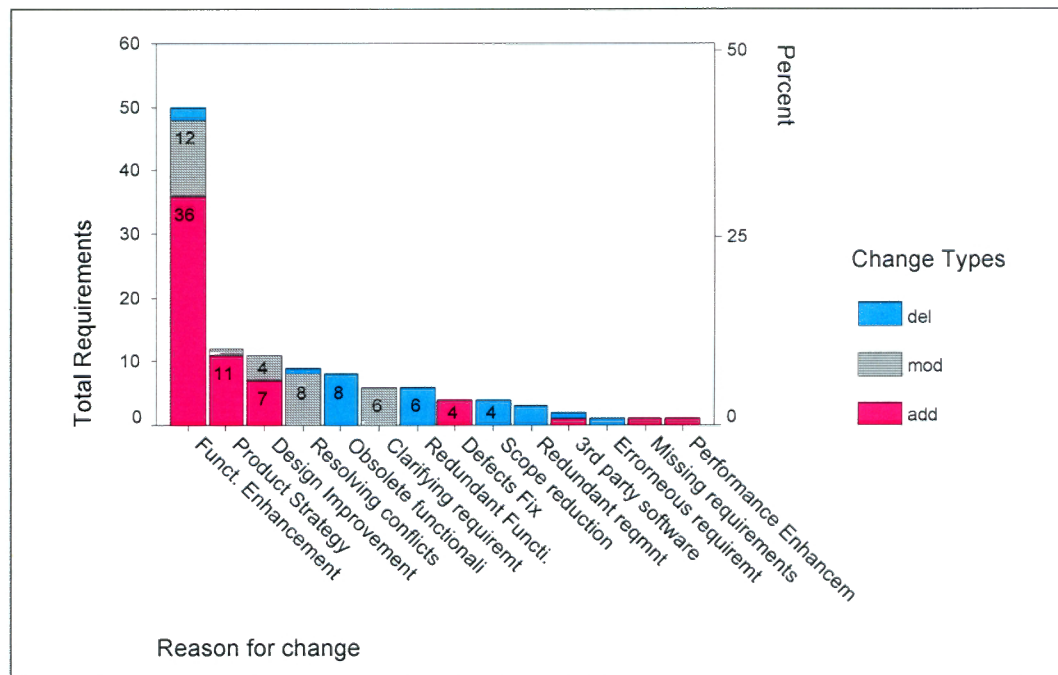


Figure 4.15 Total requirements change classified by *Reason Category* and *Change Types* – Project Release_A

As shown in Figure 4.15, the top reason for requirements change, particularly for requirement additions is “*Functionality enhancement*” requests. This reason is indeed aligned with the objectives of *Project Release_A*, which are maintaining functionality and enhancing software applications interoperability with new emerging technologies, such as .NET and the Java-based component environment. The other reasons for changing requirements are: “*Product strategy*”, “*Design improvement*”, “*Defects fix*”, and the influence of *third-party software*.

Both Figure 4.14 and Figure 4.15 presented in this section demonstrate a brief analysis of requirements change using the classification attributes. It provides timely information about the sources of requirements change and their reasons. This information leads to further assessment of improvement opportunities to anticipate requirements change in future software development projects at GDS.

The next sections present more detailed insights from using the classification to generate detailed information on the requirements change, for instance, the reason for

change and the change types that occurred at each phase of the development lifecycle.

4.3.3.4 Requirements Addition and Reason Category

Maintaining functionality and improving capability of the product release are part of the objectives of this release (*Project Release_A*). Aligned with these objectives, most of the new requirements added in the early phases of the development process are for the reasons “*Functionality enhancement*”, “*Design improvement*”, and other reasons, such as “*Performance enhancement*”, “*Missing requirements*”, “*Defect fix*”. The late additions (Jul02–Sep02) are mostly required to address functionality upgrades (migration issues), changes to screen capabilities, product packaging and installation media, and improvement of web server environment deficiencies. The detailed reasons for adding new requirements are illustrated in Figure 4.16.

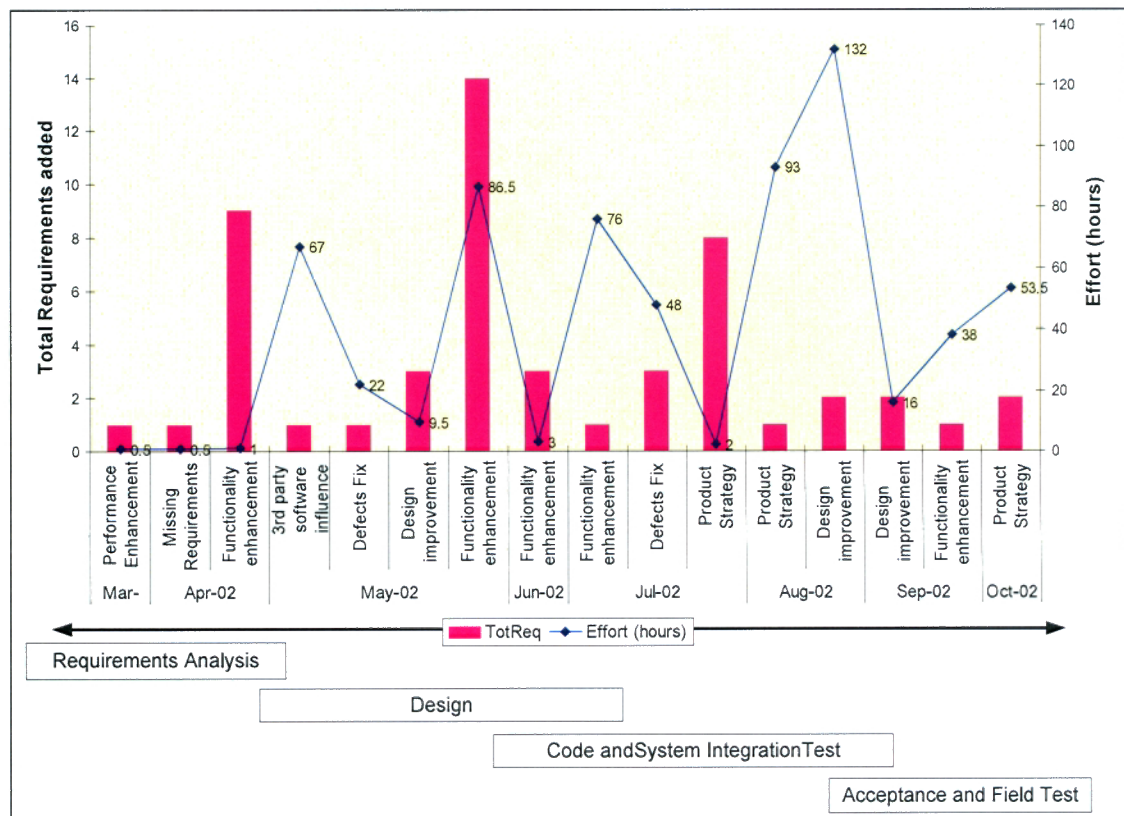


Figure 4.16 Requirements Addition by Reason Category – Project Release A

Additions of new requirements for “*Design improvement*” (May02, Aug02, and Sep02) were aimed at improving the product application design to enhance the product’s interoperability with new emerging technologies and security-related functionality. Examples of the new requirement statements are:

‘Application Dev Security shall provide the capability to restrict unauthorised users from creating and maintaining SQL Scripts’

‘The ASP Generator in the release XXC Tools software shall contain support for Line and Box drawing functionality.....’.

“*Product strategy*” is another reason for adding new requirements (or a new feature), which is related to the software release media packaging. This addition has no impact on the project schedule since it was included in the planned schedule earlier. An example of this new requirement is as follows:

“Release_A application developer release media shall be CD ROM”.

At the end of integration testing (October 2002), new requirements were added to address the requirements issues related to product branding, licensing, and packaging. The changed packaging and licensing was required to combine the two applications (Dev and Bld) packaging styles to simplify the license for the products. The requirements are:

“Application Dev CD shall be a keyless (concurrent) version”

“Application Dev and Bld shall not require a license key to install”

As the new functionality developed in this release had emerged from the functionality provided in the previous release, defects were discovered and reported by the users of the previous release. Fixing defects to improve the current release capabilities led to additional new requirements, which were intended to address specification deficiencies during migration from the previous releases, and to improve error handling options. The new requirement is as follows:

"The 'PrmyHstCldDwn' option shall be removed from the list of valid error-handling options of the transmission mode type ABW for RDB in the..."

The interdependency of the software project with third-party hardware or software vendors also contributes to requirements change. A requirement was added to address the continued support of the third-party software. This requirement was necessary since this release has to continue supporting the UNIX operating system. The new requirement is *"Release_A shall be supported on Unixware....."*.

4.3.3.5 Requirements Deletion and Reason Category

The main reasons for requirements deletion during the development lifecycle were related to *"Obsolete functionality"*, *"Redundant functionality"*, *"Scope reduction"*, and *"Requirements redundancy"*.

As shown in Figure 4.17, the deletions of redundant requirements and obsolete or redundant functionality were detected early in the development phase. This early detection is useful for the organisation in avoiding rework in the later phases of the development process. Besides those reasons mentioned above, reducing the scope of the work or removing a lower-priority sub-feature due to lack of resources has also led to removing some existing requirements.

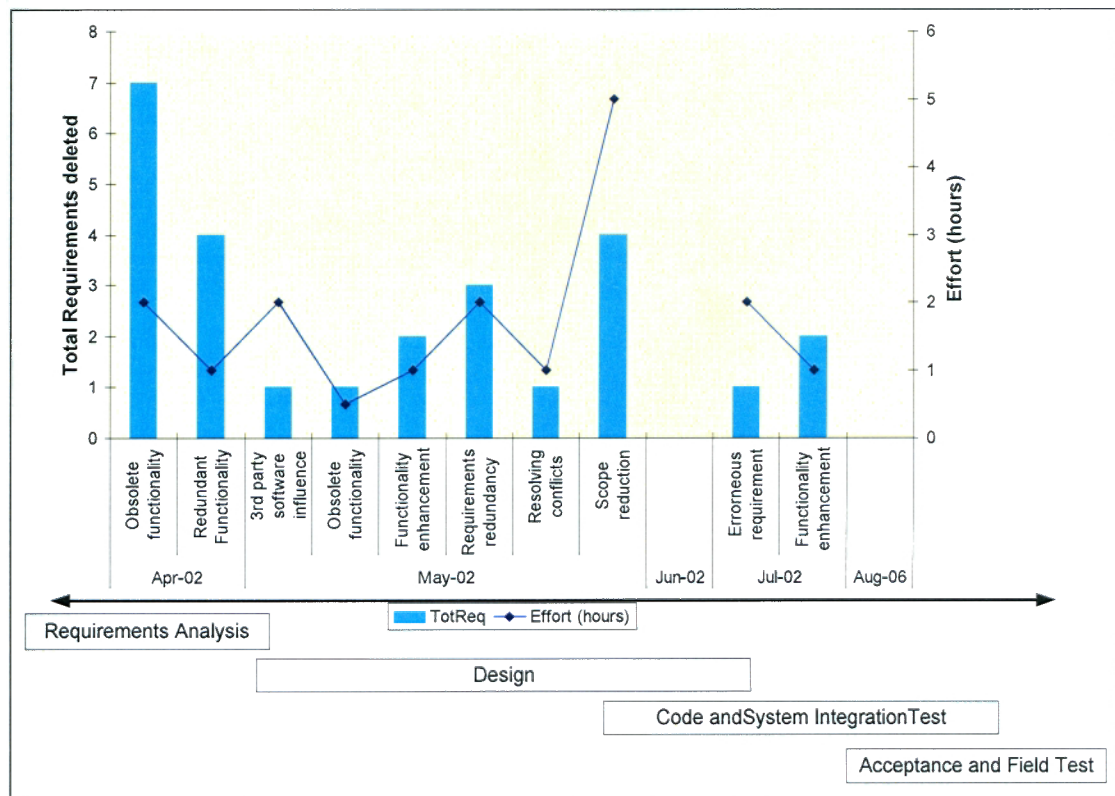


Figure 4.17 Requirements Deletion by Reason Category - Project Release A

Other reasons, such as “*Resolving conflicts*”, were intended to remove an existing requirement from this release that triggered functionality conflicts. This action indicates that the development team at GDS did not want to take the risk of keeping this requirement, which could trigger conflicts between two sub-functionalities.

The dependency of this release on “*third-party software or hardware*” also contributes to requirements deletions. An example of this deletion was removing a requirement relating to the third-party software (*Oracle*) because this party was not available to support the application being developed for a particular operating system.

Although most requirement deletions occurred in the early phases of the development lifecycle, requirements were still being deleted in the later phases due to “*Erroneous requirement*” and “*Functionality enhancement*”. The former was related to a requirement that was wrongly included in one of the software platforms, while the latter was intended to improve the consistency between two application environments of *Listbox Data File* handling.

4.3.3.6 Requirements Modifications and Reason Categories

Requirements are modified during software development for various reasons. A modification is referred to as rewording a requirement statement by adding, removing, or replacing words in the requirement statement that may or may not change the meaning of the requirement itself.

Most of the requirements modifications in this release were related to “*Functionality enhancement*”, “*Resolving conflicts*”, and “*Clarifying requirements*”. These modifications mainly derived from functional and design specification changes that occurred during May02 and June02 (see Figure 4.18). They are, for examples, rewording requirements to address the functional specification changes in a *data transfer mechanism*, rewording requirements to include *all the screen from all host runtime systems that can be accessed in ‘GIW’*, and to improve the integration of ‘*Listbox Data File Handling*’ within two application environments (either ‘*WDP*’ or ‘*WebServer*’). All these modifications indicate the enhancement of the product specifications to maintain their capabilities.

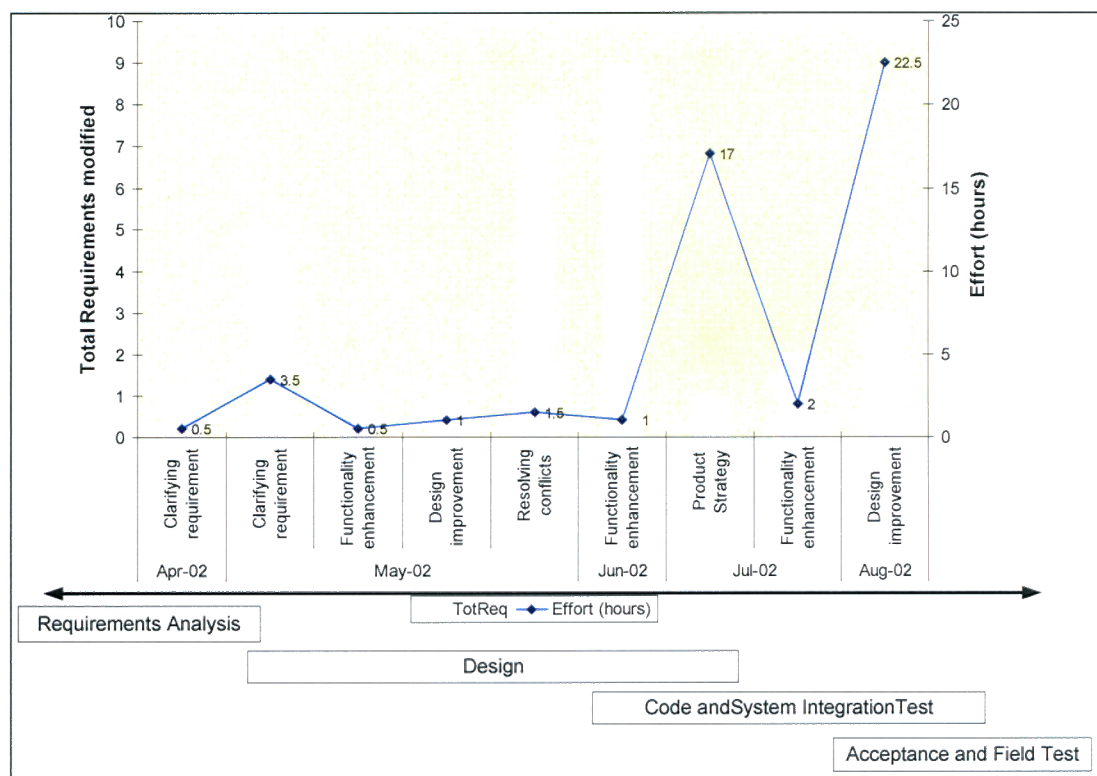


Figure 4.18 Requirements Modifications by Reason Category - Project Release A

An example is resolving functionality conflicts that exist in the functional specification when a unique version identifier (GUID) for graphical presentation is used, instead of a date-time comparison for the graphical presentation format. This functionality can lead to the circumstances where the software will not operate as the user would want it to. An example of this modification is as follows:

"The CS Implementation shall provide the capability to compare the current graphical presentation version-information on the local workstation against the latest-generated-screen-in-the-host-runtime deployed version on the Web Server or distribution master when a Web Server is not used to deployed the client interface files"

It seems that this modification captured more detailed information (the underlined portion) that improved the clarity of the requirements as well as removing the ambiguity of the requirement.

Rewording a requirement's text may or may not change the meaning of the requirement itself. For instance, as a result of feature proposal reviews, some requirements had to be modified to remove specific words in the requirements text that were not within the scope of this release. The following example presents a modified requirement text to remove a reference to "Unix", which was not included in this release:

"Application Release_A shall support read/write access of the generated application databases on 2200 and Windows ~~and Unix~~ platform from Dev test"

The following modified requirement is intended to amend the clarity of a requirement without changing its whole meaning. This modification originated from the Marketing group. The requirement was modified by adding 'multiple' to the requirement text:

"Application Release_A must be able to take advantage of the mpc architecture bythat on multiple commodity 4-x and 8-x servers"

Another example of modification requests resulting from design improvement initiatives was a modification of requirement text to reduce its complexity. The software developers at GDS claimed that this functionality simplification required less effort to implement. The modified requirement is as follows:

"Application Dev shall be enhanced to include the GIW functionality such that each text object can be defined as a multi-line display ~~with shift enter taking the cursor to the beginning of the next line~~"

"Product strategy" refers to changes that are concerned with product packaging, installation or licensing. The following is an example of a modified requirement for this reason, which was requested by the Marketing Group to include "*Business Integrator*" software in this product release. The requirement is modified as follows ("*Business Integrator*" added):

"CTs CD-ROM set shall contain installation software and the following components: CE; CE Viewer; ASP Generator;; Business Integrator and all the documentation for these components."

4.3.4 Phase_2: Additional Findings on the Change Control Process and the CR Form

In parallel with the quantitative analysis of requirements volatility conducted during the *Phase 2 – Change Analysis 1* activity, the researcher also examined the Change Control Process (CCP) document that is used by GDS. The following are various challenges encountered during this phase:

1. Most of the proposed CR forms have insufficient information to be used for analysing the importance of the proposed changes. Essential information such as rationales behind the proposed CR and work products impacted by the change are very important information during CR evaluation and approval. Although the CR template form provides these essential fields, they are not mandatory.

2. Although the CR template form provides fields to record the change impact analysis, most of the downstream impacts of requirements change were not captured and recorded in the submitted CR forms. No formal change impact analysis was performed, and no impact analysis guidelines were available.

In order to overcome this difficulty, the researcher had to elicit the missing information through direct communication with the person who submitted the change request (henceforth called the Initiator), asking questions, such as: *What is the change request about? Why do you need this change? How many work products are impacted by the proposed change?* During these informal discussions, the researcher took notes or wrote a memo (summary) of the discussion immediately after the meeting.

4.3.5 Phase_2: Summary

During this phase, CR documents were inspected and analysed, and requirements volatility rates were calculated and presented in graphical views. In further analysis, a classification of requirements changes was developed and used to characterise requirements volatility issues in depth. In addition, the GDS change control process (CCP) and CR form documents were also inspected.

The additional findings on the CCP and CR documents led the researcher to conduct the activity of *Phase_4*, which is a detailed assessment of the CCP documents and GDS participants' survey on the process. Findings for *Phase_4* activities are presented in Chapter 5.

The classification of requirements change consists of three components. One of the components, *Reason category* had to be defined carefully because not all the written CR forms explicitly stated the rationale for the proposed change, while other components such as *Change types* and *Change origin* were easily identified. This is a problem faced by the researcher in identifying patterns in the reasons for requirements change. In this phase, the researcher defined the *Reason category* attributes based on the researcher's understanding of information stated in the CR forms and also by gathering more detailed information from the project manager and the person who submitted the CR. Therefore, in order to validate and triangulate this

preliminary classification, the researcher conducted *card sorting* exercises with the GDS staff. This technique enabled us to gain insights into the way software developers or project managers classified change requests. Detailed explanations of the *card sorting* exercise are given in Section 4.4.

4.4 *Phase_3: Validation of the Classification*

This section describes research activity *Phase_3* (as illustrated in Figure 4.1), which is aimed at verifying and validating the classification of requirements changes generated from *Phase_2*, particularly the '*Reason category*' attributes. Since the preliminary classification (RC-1) in *Phase_2* was mainly based on the researcher's understanding and assumptions, it had to be validated in order to ensure the accuracy of the interpretation of CR data.

The *card sorting* technique was used as a data collection method to gather information about the requirements change classification from the software developers' perspective at GDS. Face to face interviews during the card sort exercises were also employed. GDS's software engineers and change request documents were the main sources of evidence used in this phase.

This section also describes in detail the process and procedures of conducting the card sort exercises as well as the analysis of the card sort data. Findings from *Phase_2* activities are then presented.

4.4.1 *Phase_3: Card Sorts Overview*

Although it is a simple technique for knowledge elicitation, the card sort is an established method and has been widely used in various fields such as psychology, knowledge engineering, software engineering, and website design. In the field of requirements engineering, card sorting is described as one of the effective methods for eliciting requirements (Maiden and Rugg, 1996).

Card sort exercises typically consist of the researcher creating approximately 60 index cards (3" × 5"), on which a description of domain entities is printed. The participants in the card sort arrange the cards into groups or categories, and explain

the criteria they use for sorting and the names they assign to groups (Palmer et al., 1988, Maiden and Hare, 1998, Upchurch et al., 2001).

It has been demonstrated that card sort exercises have many positive aspects that make them a useful elicitation tool, such as (Rugg and McGeorge, 1997):

1. the card sort technique can be used to investigate respondent's recall knowledge of the domain entity;
2. this technique is useful to distinguish between high- and low-level problems;
3. within card sort sessions, it offers more insights into the target population's views of the topic;
4. card sort results provide an input to be used for further analysis or classification;
5. card sort exercises can be done relatively quickly, at minimal cost, and are flexible and easy for the researcher to handle

In general, most researchers have suggested that the card sort method is an excellent approach to help develop classifications and that it delivers the classifications that people actually use.

4.4.2 Phase_3: Data Collection and Analysis Framework

Card sort and interview techniques were used as data collection methods in this phase. The card sort exercises were conducted at GDS with GDS's managers/software developers as participants. The interviews were carried out at the end of the card sort sessions using open-ended questions, and the interviews were recorded on an audio tape recorder and transcribed for a further analysis.

The main source of evidence used in this activity was the change request documents, in which the context of each change request was summarised. As the card sort sessions were completed, the data were analysed using card sort data analysis, while the interview data were transcribed and analysed using *content analysis*.

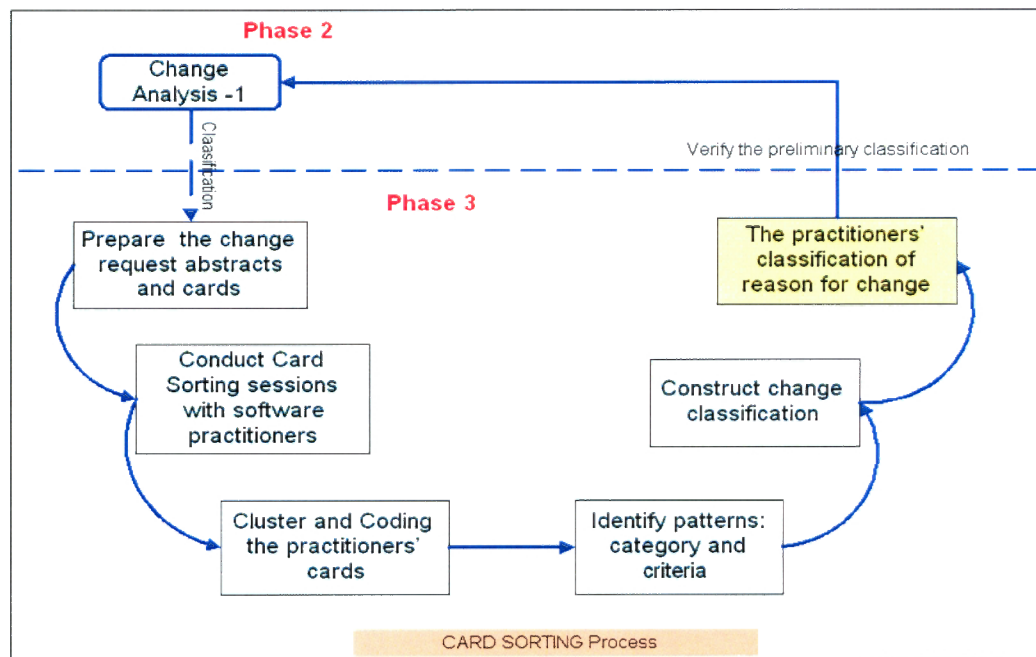


Figure 4.19 *Phase 3: The validation process*

The process steps involved in this phase are illustrated in Figure 4.19, in which at the end of the activity, the software developers' classification of requirements change requests was constructed. This classification is later used to improve the preliminary classification of requirements changes constructed in *Phase_2*. Findings of the card sort sessions have also been published in a major international conference⁸ (Nurmuliani et al., 2004b) to gain feedback from the relevant experts.

4.4.3 Card Sort Setting and Participants

Prior to the card sort exercises, verbal permission was obtained from GDS management to involve their staff. The management then selected 12 out of 20 senior software developers in the organisation. The management sent a notification letter to all selected participants requesting their contribution to the study. The software developers were contacted to introduce the goals of the study, and sought their commitment to the project. Only two engineers could not participate because they were on leave.

⁸ The 12th IEEE International Requirements Engineering Conference

Thus, a total of 10 senior software developers agreed to participate in the card sorting exercises, which were scheduled during a one-week period and the time slot for each participant was arranged according to the participant's availability. These participants represented various functional areas in GDS, such as senior manager, project manager, management lead, systems architect, and technical lead. They have also acted as Initiator, Reviewer/Approver, Implementer, or Verifier of CR.

4.4.3.1 Card Sort Materials

Before the actual card sort exercises commenced, a list of candidate entities was prepared, i.e. requirements change problems. The entities were extracted from the change request database. The preparation stage was the most challenging and time consuming. A brief description of each change request was extracted from the change request forms, as illustrated in Figure 4.20.

As a result of the initial preparation, a set of 52 cards were produced and each card contained a brief description of a requirement change. These cards represent actual requirements change requests submitted to the GDS projects. The description includes the type of changes, the rationale or reasons for a requested change, and change activities involved. The description of each item was printed on a 3" × 5" card. All the cards are the same size and each is numbered with a unique randomly generated number.

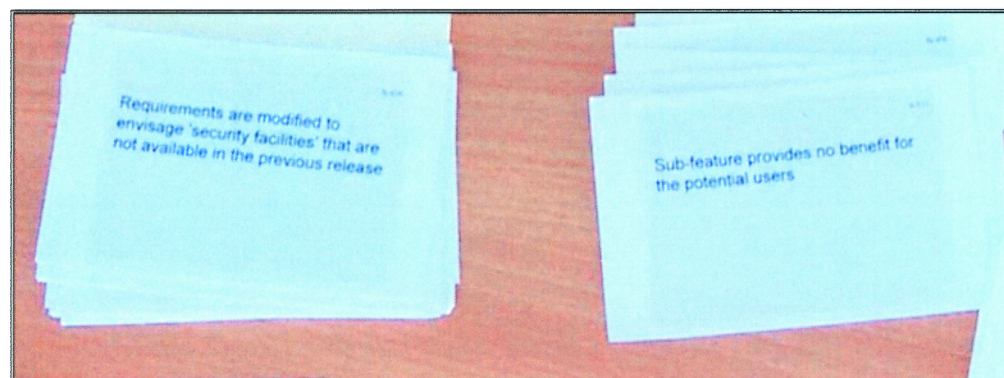


Figure 4.20 An example of the Cards

4.4.3.2 Card Sort Procedures

The card sort exercise was conducted as a one-on-one session with each participant. Since the software developers have very limited time available within their tight daily schedules, the exercise only involved 'single-criterion sorts', i.e. sorting the same set of cards, using a single criterion, which is *reason for change*.

The card sort proceeded in the following five steps:

1. At the beginning of the exercise, a brief explanation of the sorting exercise, and verbal instructions were given to the participant. The main purpose of the sorting was to classify the change requests related to changing requirements.
2. The participant was given the cards. Before the sort began, the participant was given time to read through all the cards to familiarise himself with the contents of cards.
3. The participant was instructed to sort the cards into groups of similarity according to his own single criteria. The cards were placed on the table and arranged into groups or piles. The participant was free to form as many groups of cards as they felt necessary,
4. After the sorting was completed, the participants' chosen criteria and categories were recorded. The unique number of each card was used to record which cards were placed in which categories. Each participant was also given a pad of blank Post-It stickers to write a label or the name of each of the categories. An example of a participant's set of cards is illustrated in Figure 4.21.
5. At the end of the sorting exercise, a set of questions (see Appendix D) was administered to obtain further information about the participants and their views on classification and the card sorting method. The interview was recorded on audiotape and transcribed.

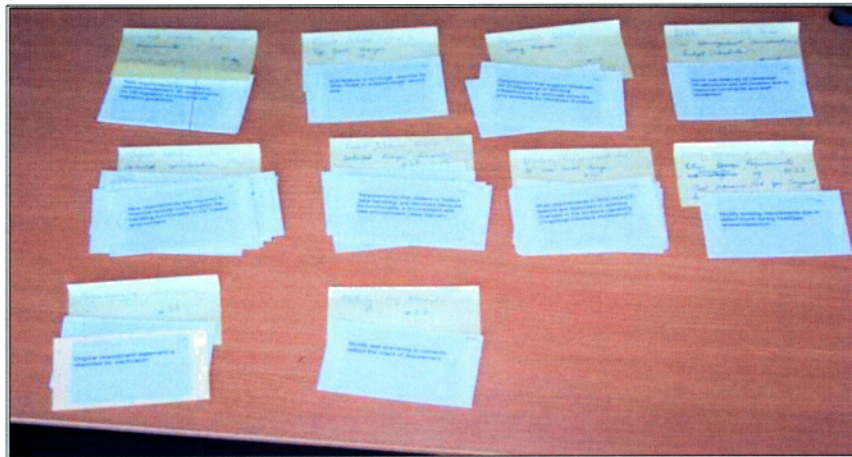


Figure 4.21 A participant's set of cards after sorting

4.4.3.3 Card Sort Analysis

Since the purpose of this activity was to establish the way managers/software developers categorise requirements change requests and to gather more information about the kinds of classification they produce, the analysis is primarily qualitative. Since a small sample of participants was involved in this exercise, statistical analysis was not appropriate. The card sort data was analysed in terms of: *lists of the participants' criteria, number of categories, lists of named categories, and the agreement between the participants on their categories.*

Content analysis was used to assess the agreement between respondent's classifications. There are two forms of agreement, *verbatim* and *gist*. Verbatim agreement takes place when different participants use exactly the same words. Gist agreement takes place when different participants use different words for the same ultimate meaning (Rugg and McGeorge, 1997). For gist agreement, *independent judges* (two other members of the Requirements Engineering Research Group at UTS) were asked to identify which criteria and categories are forms of gist agreement.

In addition to the content analysis, since this card sort is an open sort (no predetermined categories were used), the respondents' categories need to be standardised in order to effectively analyse the strength of a category.

Furthermore, the transcripts of the interviews with the participants were analysed to gather more information about the participants' perspective on the requirements change classification and the Card Sort technique.

4.4.4 Phase_3: Card Sort Findings

4.4.4.1 Criteria and Categories generated by the participants

Each participant completed the Card Sort exercise, in which each took an average of 40 minutes, though some took 60 minutes. The average number of categories given by each participant was 6.6 with the minimum number of categories 4 and the maximum number 10 (see Appendix E for the detailed results).

As a result of the Card Sort analysis, the criteria used by the SE practitioners include: *reason for change*, *general changes*, *schedule impacted*, and *the magnitude of effort involved*. The most common criterion, used by 60% of the participants was '*Reason for changing requirements*' (see Figure 4.22).

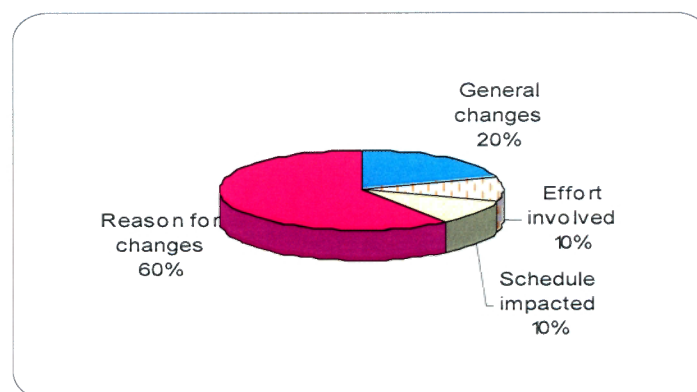


Figure 4.22 Criteria generated by the participants at GDS

A total of 66 categories were generated from the 10 participants' sorts. The distribution of each participant's categories and criteria, which shows that each participant has different number of categories constructed, is presented in Figure 4.23.

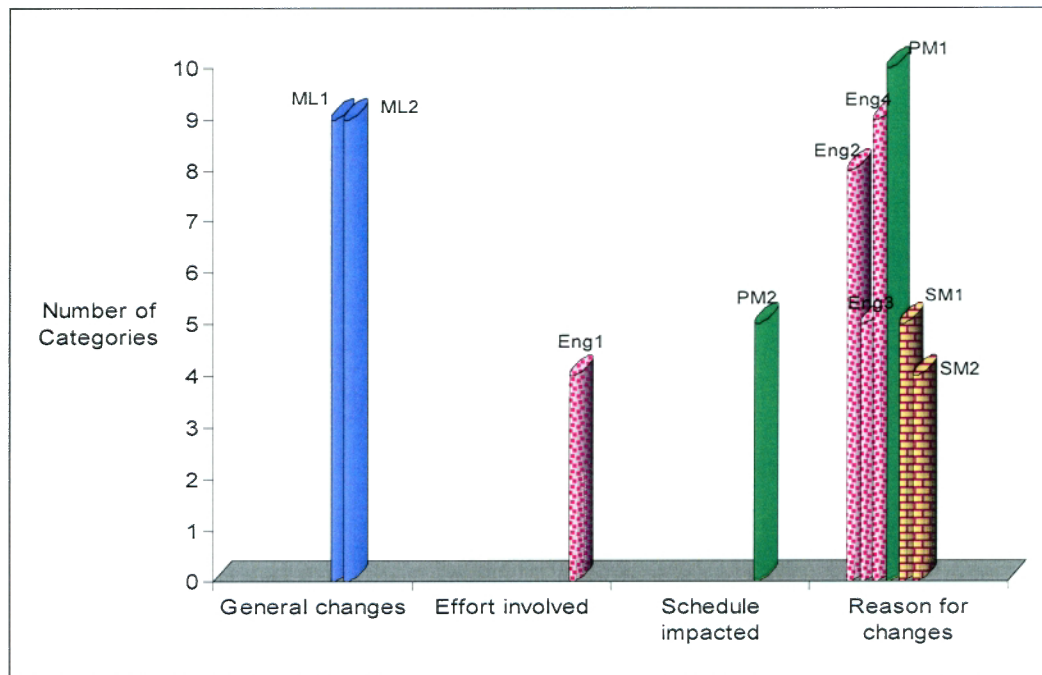


Figure 4.23 The distribution of categories according to the sorting criteria

(Note: SM=senior manager, PM=project manager, ML=management lead, and Eng=software engineer)

One interesting finding is that the two management leads used the same criteria to sort the cards and produced the same number of categories. However, they named their categories differently. Figure 4.23 also shows that only one participant (a software engineer), used “*the magnitude of effort involved*” as a sorting criterion, while the rest of the participants used “*reason for change*”. The number of categories varied across the different participants. The following bullet points present findings from those criteria.

- **Categories and Criteria: Reason for changes and General changes**

The following findings represent the card sort results from the participants who used the criteria of “*reason for change*” and “*general changes*”. Since the Card Sort exercise we conducted at GDS is an open-sort type and no predetermined categories were used, the participants’ categories are standardised by converting their raw categories into a higher-level abstraction. In this initial analysis of the cards, the agreement between the participants is assessed. Once the verbatim and gist agreement has been analysed, the raw categories are grouped into “superordinate constructs” (higher-level constructs), in which each construct was given a name that represents the identified categories. For instance, “software environment-related to

interoperation with COTS” and “external influences/customer/third-party software” can be grouped into the superordinated construct of “third-party hardware/software changes”.

Table 4.4 shows the superordinate constructs identified in this case study. It presents the number of raw categories, which represent the number of participants who generated the construct. A superordinate construct with a high number of raw categories (e.g. “*Design improvement*”) may indicate clear agreement amongst the participants on the name of the category where a group of the cards belongs, whereas a low number (e.g. “*Missed requirements*”) indicates the categories that few agree on.

The least elicited categories are the last five constructs, which are unique categories. These categories do not easily fit into any of the superordinate constructs defined above. Although these unique categories indicate that the participants have widely different opinion on the content of the categories, they are equally as important as the others when used to validate the preliminary requirements change classification.

The card sort findings may be used to investigate the placements of individual cards, categories, or defined group of categories. In this study, the focus of the card sort was mainly to identify the participants’ categories.

- **Categories and Criteria: The magnitude of effort involved and Schedule impact**

There were two participants who used two different criteria in the card sorts, e.g. ‘*the magnitude of effort involved*’ and ‘*schedule impact*’. These two participants produced significantly different categories. The first participant (*Eng*) used a criterion of ‘*the magnitude of effort involved*’ to group the cards into four categories:

1. “High Effort” – functionality changes in known complex area.
2. “Medium Effort” – requirements changes that have reasonable impacts on development effort.
3. “Low Effort” – minor changes to requirements or functionality.
4. “No Effort” – rewording requirements statement.

Table 4.4 Number of Categories Elicited in Superordinate Construct Groups

Superordinate construct	Examples of raw categories	Total raw categories elicited
Design improvement	<ul style="list-style-type: none"> ○ Requirements change to match design ○ Requirements change due to increased knowledge and further designs experience gained ○ Low-level design changes ○ Detailed design discovery ○ Top-level design changes 	8
Third-party hardware/software change	<ul style="list-style-type: none"> ○ Changes in external software ○ Changes to the consistency of configuration/platform-supported hardware/software ○ External influence/Customer/third-party software ○ Hardware/software environment changes or conformance to industry trends ○ Changes related to interoperation with COTS 	8
Scope reduction	<ul style="list-style-type: none"> ○ Management consideration on budget/schedule issues ○ Resources and technical reason (functionality not needed) ○ Eliminating requirements deemed to be of low or no value to customers ○ Requirements may affect commitments schedule ○ Resources availability 	6
Installation/Branding/Packaging changes (Product Strategy)	<ul style="list-style-type: none"> ○ Changes related to the physical form in which the product is delivered ○ Packaging/branding ○ Product marketing or packaging/branding or licensing 	5
Clarification	<ul style="list-style-type: none"> ○ Requirements clarification ○ Rewording requirements to eliminate chances of misinterpreting requirements 	5
Functionality enhancement	<ul style="list-style-type: none"> ○ Migration-related to maintaining/ managing functionality as the product is released at known versions ○ Functionality changed (upgrade) 	4
Redundant requirements	<ul style="list-style-type: none"> ○ Resolving requirement overlaps or redundancy ○ Rewording/removing duplicates or redundant requirements 	3
Missed requirements	<ul style="list-style-type: none"> ○ Missing requirements – identified via traceability ○ Missed requirements that were not captured at the beginning of the product definition 	3
Testability	<ul style="list-style-type: none"> ○ Modification of requirement due to test scenario/test specification changes ○ Modification to increase test coverage or ensure that test correctly address requirements 	3
Wrong requirements		1
Unnecessary Requirements capture		1
Architectural incompleteness		1
Bug fix		1
Incomplete requirements		1

The second participant (*PM2*), categorised the cards into five categories based on '*schedule impact*'. His categories are:

1. New requirements with high impact on schedule,
2. New requirements with low impact on schedule,
3. Modified requirements with significant impact on schedule,
4. Modified requirements with no impact on schedule,
5. Removal of requirements that have no impact on schedule (i.e. less work).

Thus, this participant was mostly concerned about the impact on project schedule of adding new requirements or modifying existing requirements.

Based on the card sort findings, the superordinate constructs identified and presented in the Table 4.4 are valuable information for the validation of the preliminary classification of requirements change and for constructing a future classification. Besides the findings described earlier, the recorded interview data were transcribed, analysed, and the summary of the interview is presented in this section.

4.4.4.2 Verification of the Preliminary Classification

Based on the card sort findings, it appears that GDS participants' classification produced more or less similar categories as the "Reason Category" within the preliminary classification. Although there are differences in a few categories between the two classifications, such as, "*Obsolete functionality*", "*Redundant functionality*", "*Resolving conflict*", and "*Testability*", the majority of the categories are similar. The most elicited categories from GDS participants' classification and the reason category from the preliminary classification are listed in Table 4.5. In terms of 'Reason Category', these two classification attributes are the valuable sources of information to be used for further analysis of requirements change.

Table 4.5 The Reason Category attributes from two classifications

Reason Category	Preliminary Classification	GDS's classification
Third-party software/tool influence	•	•
Clarifying Requirement	•	•
Defects Fix	•	•
Design Improvement	•	•
Erroneous Requirements	•	•
Functionality enhancement	•	•
Missing requirements	•	•
Obsolete Functionality	•	
Product Strategy (Installation/Branding/Packaging)	•	•
Redundant Functionality	•	
Requirements Redundancy	•	•
Resolving Conflict	•	
Scope change	•	•
Testability		•

4.4.4.3 The Participants' Feedback on the Card Sort

At the end of each card sort exercise, the participant was asked to give his or her opinion about the requirements change classification scheme and the card sorting technique. The full questionnaire is presented in Appendix F

The majority of participants responded very positively to the card sort exercise. They viewed a card sort as a good technique for grouping the cards that represent the requirements change requests quickly. The following comments reflect their positive reaction:

"...It is good and forces you to make decision"

"...It not only forces you to make decision, it also gives you an ability to make decision"

"It is a useful technique because you can separate or sort them (requirements change requests) easily"

Another participant explained that the card sort is useful “for me to start thinking about categories”. He continued to say “It forces me to put cards into some sort of categories that I never thought of before”.

Regarding the classification of requirements change requests, the majority of participants considered that the change classification may provide substantial benefits in improving their current change process and in the way they analyse changes.

4.4.4.4 Benefits of Change Classification

The following lists the benefits of the change classification (in general) extracted from the participants’ interviews:

1. The classification could be used as a means of controlling and managing changes.
2. It can help in assessing the impact of requirements change in a reliable way.
3. It can promote a common understanding within the software development team of what the changes actually mean.
4. It can be used to identify risk associated with each change request or the group of changes.
5. It can help in determining the change acceptability (i.e. reject or approve changes), hence supporting crucial decision making throughout the software development lifecycle (*project manager and system architect*).
6. The categories and the subsequent constructs could be used to develop a multi dimensional matrix of all change requests. “The particularly useful dimensions for more effective decision making are schedule, effort and reasons for changes”, said one of the project managers, *PMI*. He further commented that the matrix could be populated with all the requirements change requests after categorisation had been performed. For example, when project managers need to schedule change implementation, they could consult the matrix to identify the effort, schedule and the reasons for each change request for their assessment (*PMI*).

4.4.5 Phase_3: Summary

The card sort technique has been applied to validate the preliminary classification attributes developed earlier in the *Phase_2*. A set of participants' categories and criteria on requirements change classification has been identified. This classification was also used to verify and improve the preliminary classification developed earlier.

The card sort exercises have been successfully conducted in a real-life software industry environment. The findings generated from the card sort data are interesting and provide insights into the way software practitioners classify requirements change requests. These results also revealed that the criteria used by the participants to classify change requests were likely associated with the practitioner's role in the organisation.

4.5 Summary

This chapter has covered the empirical findings from the first three phases of the research activities (*Phase_1*, *Phase_2*, and *Phase_3*). Different data collection methods, sources of evidence, and analysis methods were applied to address the objectives of each phase (see Figure 4.1). The detailed process of each phase to meet the objectives is described and the findings of each phase are also discussed.

Phase_1 is the preliminary research activities conducted to learn about the organisation structure and processes used in managing software requirements. The GDS organisation was chosen because it met the criteria for a case study site considered in this thesis to investigate the problems of requirements volatility during software development lifecycle.

Phase_1 activity was followed by *Phase_2* which describes change request analysis activities. A classification technique was used to analyse each change request and a preliminary classification of requirements changes was constructed. This classification, with its three main elements (*Change Types*, *Reasons Category*, and *Change Origin*), has proved to be a strategic tool to characterise the nature of requirements volatility.

The rates of requirements volatility throughout the development lifecycle were measured and represented in graphical forms. This measurement and analysis approach was found useful to improve our understanding of requirements volatility problems. The classification developed in *Phase_2* was also used to assess the amount of effort required to implement each change. Findings from this assessment provide valuable information for effort estimation purposes. However, because the source of effort data is the 'estimated' effort data as identified in the CR form, a more accurate analysis would benefit from having actual effort.

Phase_3 is the research activity to elicit the perspective of GDS software engineers. This activity was aimed at validating the classification constructed in *Phase_2*. The card sort technique was chosen to address the *Phase_2* objectives because it is a simple technique that can be used to investigate GDS software engineers' recall knowledge of requirements change issues. The results of the card sorting activities represent the perspectives of GDS software engineers. These results have been used to refine the preliminary classification developed in *Phase_2*.

Chapter 5: Case Study Findings – Part 2

5.1 Introduction

This chapter presents empirical findings that resulted from the last three phases of research activities carried out at GDS. As highlighted in Figure 5.1, the phases are *Phase_4* – Change process assessment, *Phase_5* – Change analysis_2, and *Phase_6* – Evaluation of a new change process. Various data collection methods, sources of evidence, and analysis techniques were applied across the three phases. This chapter is organised as follows:

- Section 5.2 describes the assessments of the existing change control process and pre-survey on the process conducted at GDS (*Phase_4*)
- In Section 5.3 empirical findings of requirements volatility study from *Project Release_B* data are presented (*Phase_5*)
- In Section 5.4 the evaluation of a new change control process is described (*Phase_6*)
- Section 5.5 provides a summary of this chapter

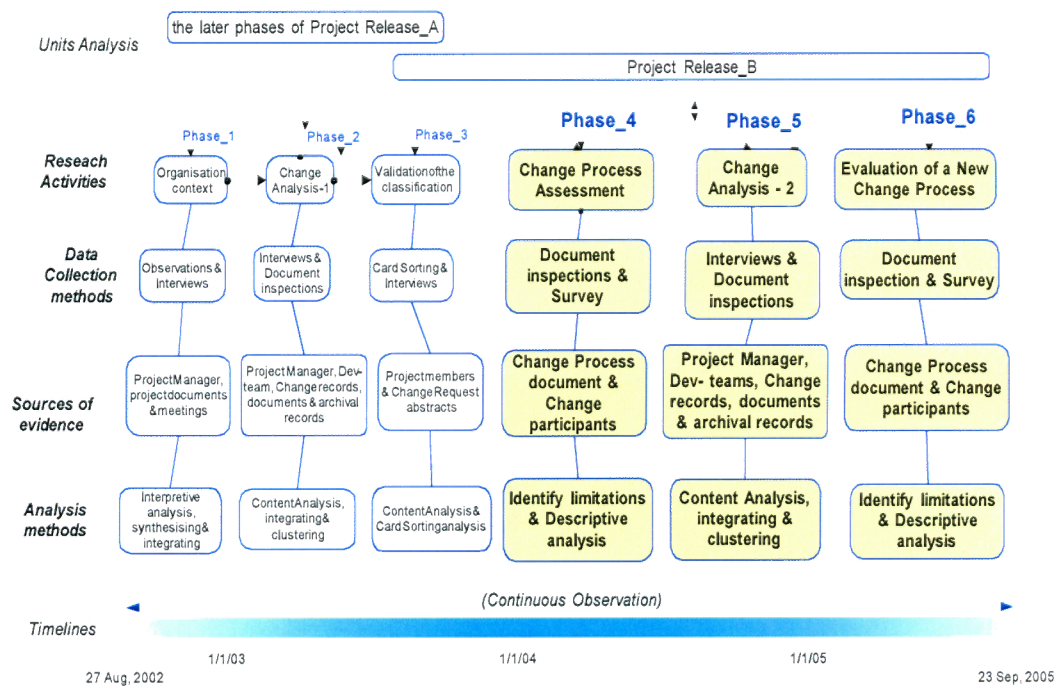


Figure 5.1 The last three phases of research activities

5.2 Phase_4: Change Process Assessment

This section presents the findings of *Phase_4 (Change Process Assessment)* activities. As illustrated in Figure 5.1, this phase is a follow up to the activity carried out during *Phase_2*. The objective of *Phase_4* is to further analyse the existing Change Control Process (CCP) documents and to investigate the usage of the CCP and its accompanying Change Request (CR) form in the context of the users' perspectives (GDS change participants). This activity addresses the following research questions:

[RQ3b] What strategies does the organisation use to manage requirements volatility in practice?

[RQ3c] What are the current limitations of these strategies?

In this section, data collection methods and sources of evidence used to achieve the objectives are described. The overview of the Change Control Process (CCP),

limitations of the process and its usage based on the change participants' perspectives are also presented.

5.2.1 Phase_4: Data Collection and Analysis Methods

Data gathering in this phase was accomplished using document inspections and a survey technique. Extensive inspections of the CCP document and the CR form were initially conducted during the activities of *Phase_2* and continued during this phase. In addition, a survey technique with a structured questionnaire was used to collect GDS participants' perspectives or opinions about the current Change Control Process (CCP) and the Change Request (CR) form. The survey includes a set of questions, which were reviewed and refined through several iterations before administering to the participants. A pilot of the survey questionnaire was conducted at the Requirements Engineering Research Laboratory of UTS, Faculty of IT. Four colleagues of the researcher participated in this pilot. The questionnaire for this survey is included in Appendix G.

The participants of this survey are GDS employees who have been involved in processing change requests. The change participants are the *Initiator*, *Reviewers/Approvers*, *Implementers*, and *Verifiers* of the change requests. Their names were collected from the change request records. This pre-survey was conducted prior to rolling out of the new CCP and CR form. The following describes the steps for conducting this survey:

- Step 1: A list of the participants' names was collected from the CR documents submitted to the project repository and 50 staff were selected as the survey participants. However, after a careful assessment and discussion with the management, five staff were not available at the time this survey was conducted. They have either left the organisation, were on leave, or were on overseas duty. As a result, the total number of participants for this pre-survey was 45 respondents.

- Step 2: An invitation letter with the pre-survey questionnaire was emailed to all participants through the project manager. This pre-survey was conducted in a four week period, from April 2004 to May 2004.
- Step 3: A reminder was sent to the participants in the third week after the questionnaire has been distributed.
- Step 4: Completed questionnaires were collected both via email and in person within the survey time frame.
- Step 5: At the end of the time frame, information from the completed questionnaires was recorded and analysed by the researcher.

5.2.2 Phase_4: Findings on the Assessment of Change Process Documents

In the beginning of *Phase_4*, several discussions were initiated with GDS's project manager of *Project Release_A* about the implementation of the existing CCP and CR form. Besides the document inspections, the researcher also observed the way the project manager tracks and monitors the progress of proposed/approved change requests. Details of the observations and inspections are described in this section.

5.2.2.1 Change Control Process

The CCP is the centre of change management scheme, which supports GDS in controlling changes to the project deliverables during development process. The CR form, as part of the CCP, is used as a medium to report problems or requests for changes to project deliverables that are placed under configuration management. The CR form is a MS Word-based form and is submitted electronically to the project database. The project manager is responsible for facilitating change requests submitted to the project.

Based on the CCP documents inspected, there were three types of changes covered by this process. They are as follows:

1. Changes to project deliverables and milestones, for examples changes to requirements specification, functional specifications, and project plans or milestones. Any changes to these documents are communicated through a CR form.
2. Minor changes to design documents. These types of changes are communicated through email and peer-review among software developers. If the changes have impacts on the requirements (e.g. MKR, TCR, ITR), a CR form will be raised to communicate the changes.
3. Changes to code, which generally results from system integration testing and system acceptance testing. This type of change is recorded through a 'System Incident Log' (SIL) tracking system. If the requirements or functional specification are affected, a CR form will be raised.

The process of handling the proposed CR forms consists of four main activities: *Initiate CR*, *Review/Approval of CR*, *Implement CR*, and *Verify CR*. A change request is raised and submitted to the project manager. The project manager is then responsible for managing and tracking the submitted CRs for each project. It is stated in the CCP document that the project manager's responsibility is to validate the CR, enter the CR information into the CR database, amend the change impacts on project schedule and schedule the work for change implementation. In addition, the project manager also tracks and reports the progress of the CR regularly. It seems that the project manager has two major responsibilities, managing the project and managing day-to-day changes. This substantial workload may contribute to unresponsiveness in tracking the progress of CRs, particularly when no automatic tool to support the process was in place.

The CCP consists of four main activities. The descriptions of details of these activities were obtained from further inspections of the CCP document, observations and informal discussions with the project manager. The change participants who are involved in this process include:

- *Initiator*, the person who submits a CR
- *Reviewers*, two or more software developers who review and subsequently approve/reject the proposed CR
- *Implementer*, the software engineer(s) who implements the approved CR

- *Verifier*, a member of Quality Assurance (QA) team who tracks whether the approved changes have been implemented

The following are the descriptions of the four main activities based on the researcher's observation and as stated in the CCP document:

1. Activity 1: Initiate and Validate Change Request

The *Initiator* submits a proposed change to the project manager using the CR form and stores it in the project repository. Before the CR submission, the *Initiator* is encouraged to complete the CR form and provide detailed information, such as problem description, rationalisation of the proposed change, and product areas impacted by the change. However, it was stated in the CCP document that it is the responsibility of the project manager to perform an impact analysis of the change. Therefore, on many occasions the *Initiator* submitted the CR without completing the change impact analysis. These findings suggest that change impact analysis is not so simple and it should be consolidated by assigning various sources. For example if the change has potential impacts on three different areas, the *Initiator* and project manager should consolidate information from the three impacted party.

In the existing CCP document, the validation step is conducted by the project manager to check the completeness of the proposed CR, particularly when the requirement specification is impacted. In analysing the change impacts, the project manager discusses this issue with the relevant software engineers including the validity of the proposed change and estimated effort needed for implementing the change.

In completing the CR form, the *Initiator* needs to nominate required *Reviewers* to review and approve the CR. Sometimes the project manager also assists in determining appropriate *Reviewers* for particular change impacts. Two types of reviewers were involved in the process: external and internal reviewers. The external reviewer, from Marketing group (USA), is required for those proposed CRs that have impacts on the high-level requirements (marketing requirements/business requirements) and/or external milestones. The internal reviewers, from GDS, are required for changes that have no impact on the external functionalities or marketing requirements. These changes only need to be reviewed internally by GDS management, software developers, testers, product information and quality assurance

team. This conditional review of the CR, however, is not clearly described in the CCP document. However, it is included in other project documentation (e.g. Configuration Management Plan) together with a list of the appropriate reviewers.

2. Activity 2: Review and Approve Change Request

In this activity the *Reviewers* also act as *Approvers*. After the validation of the CR is complete, the project manager circulates the proposed CR form to the *Reviewers* via email. The *Reviewers* then review the proposed CR, in terms of the nature and clarity of the proposed change, its impact on the project schedule, the reasonableness and feasibility of the proposed change, prior to the CR approval or rejection. The *Reviewers* provide their responses (approve or reject) and notify the project manager. Once the CR is approved, the project manager then allocates resources and schedules the CR implementation. If the CR is rejected by one or two reviewers, then the CR approval will be escalated to GDS senior management for further analysis.

3. Activity 3: Implement the Change Request

This activity starts when the proposed CR has been approved and the change becomes part of product development. During this implementation activity, communication and change coordination among the implementers are very important, so they can trace the changes across the impacted areas. However, in this case study it was observed that it is left to the project manager and the development team members to trace the changes throughout the change implementation. Often, the approved CR does not provide sufficient information about the change implementation in various impacted areas. This situation often causes difficulties in tracking whether the CR has or has not been implemented.

4. Activity 4: Verify Change Request

The CR verification is part of the Quality Assurance (QA) tasks. The verification is intended to verify that the changes have been implemented correctly and the related documents have been updated. If the verification is successful, the CR is closed. Otherwise, the project manager or the implementer will be notified. In this circumstance, the implemented changes need to be investigated further and the CR remains open.

5.2.2.2 Change Process Flow

The inspections of the CCP document revealed that there was no process flowchart outlined in this document. The researcher managed to gather more information about the CR activities and its process flow by observations and communications with the project manager. An indicative version of the CR process flow is illustrated in Figure 5.2.

This flowchart represents the four main activities that are described above. It illustrates a generic change request process. However, this process flow does not explicitly distinguish the three types of change stated in the existing CCP document (*changes to project deliverables and milestones, minor changes to design, and changes to code*). These changes, as observed in practice, had various levels of impact that required different levels of approval. This issue has led GDS project management, assisted by the researcher, to update the existing CCP.

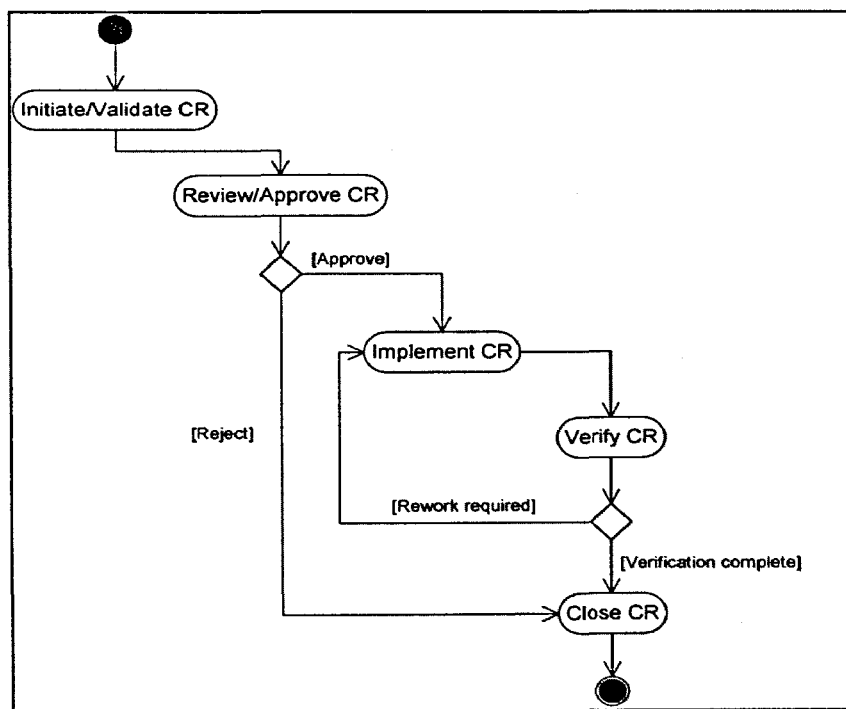


Figure 5.2 Change Control Process Flow

5.2.2.3 Change Request Form

The change request form can be used to report a problem or amendments to some work products or project deliverables with all associated impacts and other attributes. At GDS, the CR form is used to request changes or amendments to the work products and project deliverables that have been baselined. The requests were produced by the customer (Marketing group), development team members, project management, or others.

The document inspections revealed that the existing CR form provides little guidance and few mandatory fields to capture change information comprehensively, such as the information related to the impacts of a change and change effort. Furthermore, there had been no major updates to the CR form or the CCP prior to this study.

The existing CR form contains several fields that need to be completed by the *Initiator*. The fields are as follows:

- Change identification number
- Project name
- Dates (submission date, approved date, and implemented date)
- Initiator name
- Summary of change description (for all change types)
- List of approved documents to be changed and estimated effort
- List of other documents affected by the change and estimated effort
- List of reviewers (approvers) and optional reviewers.

5.2.2.4 Limitations of the existing change control process and form

The case study findings indicate that among the CCP activities, the CR review, approval and implementation stages are critical. It was noted that a large number of the proposed CRs were waiting for approval or the approved CRs were waiting for their implementation. There was no clear indication of who were the reviewers required for particular change impacts. In addition, there was a lack of coordination

and documentation in the CR implementation stage. This CR backlog could halt the progress of the project.

Based on the case study observation and document inspections, some limitations of the existing CR form were identified. These limitations are as follows:

1. Most of the submitted CRs contained insufficient information related to the rationale for the change. The change information was inadequate for the analysis of the importance of the change to be made. This kind of information is necessary for analysing the problems effectively and understanding the proposed changes.
2. Although the CR form template provides fields for the change impacts and effort estimation, they are not followed rigorously by the *Initiator*. This is possibly due to the lack of guideline on how to identify the change impacts and the form itself is not well structured to capture that information. Therefore, because of missing information on the details of change impact, the *Initiator* or project manager would find it difficult to estimate change effort. In addition, there is no impact analysis checklist established and followed to perform the analysis.
3. Lack of essential fields, such as change types as stated in the CCP document (changes to requirements, design, code, or other project deliverables).

In the context of the existing CCP, several limitations of the process were also identified. They are as follows:

1. In terms of the appropriate reviewers and approver, there was no clear instruction outlined in the CCP document to choose the types of reviewers for changes with particular type of impacts, such as external or internal impacts. The project manager often referred to the project CM Plan to assist the Initiator in choosing the reviewers.
2. Roles and responsibilities of the parties involved in the CR process are not clearly stated.
3. The time to process the CR is considered slow, i.e. from CR initiation to CR approval and from CR approval to CR implementation. The change participants only used manual email notification to communicate with other participants.

4. There is no classification established in the CR form, such as the change types (addition, deletion, and modification), the change impacts (external and internal), problem severity, or implementation priority. Although GDS monitors the number and status of CRs, there is also no classification used by the project manager in analysing and scheduling the work for CR implementation.

In light of the assessment results mentioned above, it was found necessary to improve the organisation's change control process to help the organisation effectively manage changes. At the end of the *Phase_2- Change Analysis 1* and in the early *Phase_4 – Change Control Assessment*, the researcher offered some recommendations to GDS management for improving the CR template form as well as the change process in order to address all the limitations identified above. At the same time, this improvement initiative coincided with the overall Software Process Improvement (SPI) effort at GDS. Thus, the researcher began to work with GDS staff to update the CCP document and CR form.

While the existing CCP and the CR form were still being used in GDS projects (*Project Release_A* and *Project Release_B*), the new version of these documents was under development. The process of updating these documents progressed slowly. The GDS procedures for process document changes had to be followed and the proposed new documents went through the 'peer-review' process several times and were discussed in several meetings with the project managers from the two projects, engineering management leads, and QA lead. As a result of this improvement initiative, the new version of the CCP was established. The content and structure of the new CCP are significantly different from the existing CCP document. Similarly, the new CR form was also being redesigned to address the limitations described above.

Before rolling out the new CCP and CR form at GDS, a pre-survey was conducted for two main reasons:

1. to gain insights into the change participants' perspectives regarding the existing CCP and CR form.

2. to gather more information about the usage of the existing CCP and CR form, so that this information can be used as a benchmark to evaluate the effectiveness of the new version of the CCP and CR form later.

5.2.3 Phase_4: Survey Analysis and Findings

This section presents the findings from pre-survey analysis on the usage of the existing change control process and CR form. The pre-survey data table is presented in Appendix H. Interesting issues highlighted by the participants and their feedback are outlined and discussed.

5.2.3.1 Change Participants Overview

The response rate for this pre-survey was 58%, which indicates 26 out of 45 participants returned the completed questionnaire. The functional roles of the participants at the time of conducting this survey were:

- 16 Engineers/Developers (61.5%)
- 6 Management Leads (23.1%)
- 2 Testers (7.7%)
- 2 QA members (7.7%)

The participants were asked to rate their understanding of the CCP in their work practices. The survey responses show that 76.1% of the participants have ‘good’ to ‘excellent’ understanding of the CCP. Furthermore, when they were asked to rank the importance of the CCP, the findings indicate that the CCP is ranked by the participants from *somewhat important* (30%) to *very important* (70%). This is an indication that the change participants have a good understanding on the importance of the CCP in their work, which is one of the essential factors in managing change.

5.2.3.2 Change process activities

In this pre-survey, the change participants’ opinion on several CR activities was also sought. A set of questions were constructed, in which each question represents

each activity of the CCP. These questions were intended to gain insight into the participants' view on the importance of the change activities. The participants were asked to rank their answer using a 5-point Likert scale (*lowest=1, low=2, medium=3, high=4, and highest=5*) for each activity. Figure 5.3 illustrates three change activities that are interesting to note.

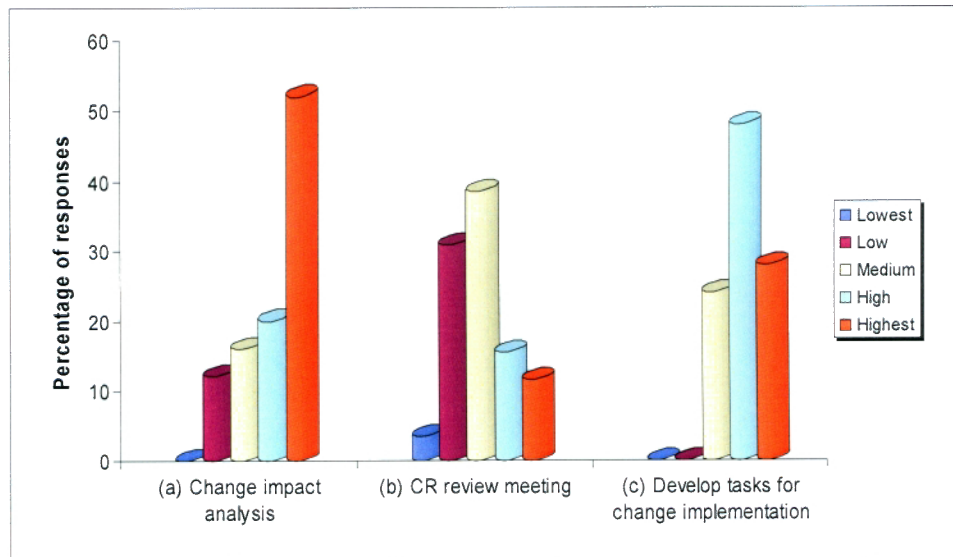


Figure 5.3 The importance level of the CCP activities by the participants

Examining the results as shown in Figure 5.3, it can be observed that more than 70% of the respondents ranked '(a) *performing change impact analysis*' as highly important (20% high, 52% highest) change activity. This indicates that the participants understand the importance of this activity, particularly during the analysis of a proposed change request. The evidence from the researcher's observations and inspections described earlier, however, suggest that this activity was not properly performed and that it needs significant improvement.

Similarly, more than 75% of the respondents (48% high, 28% highest) ranked the importance of '(c) *develop tasks for change implementation*' as high level. While this survey result points out the importance of this activity, an early assessment of the existing CCP indicates that the *Initiator* or the *Approver* (Project Manager) rarely developed or documented tasks for CR implementation once a CR was approved. The task for change implementation is to define the tasks required to implement the approved CR and assign software developer(s) who is (are) responsible to implement them. This activity is essential during the verification stage where the completion of

the CR is verified against the documents to be changed. It helps the verifier to track the CR implementation.

It appears that the participants seem reluctant to add another meeting into their day to day schedule, as 73.1% respondents ranked ‘(b) *performing CR review meetings*’ as medium to lowest level of importance. Occasionally informal meetings or discussions among the change participants during the CR review activity were conducted. However, a regular formal CR review meeting should be part of the change process because it will help the project members to promptly resolve issues or concerns among the reviewers and may reduce the time to process proposed change requests.

5.2.3.3 The Initiator’s perspectives

Of 26 participants, 22 (84.6%) participants had previously submitted change requests to the projects. The participants were asked whether they agreed or disagreed with a set of statements provided in the questionnaire regarding the issues of CR submission (see Appendix G section 2 questions 2). Figure 5.4 shows the data from the 22 responses on three main issues during CR initiation.

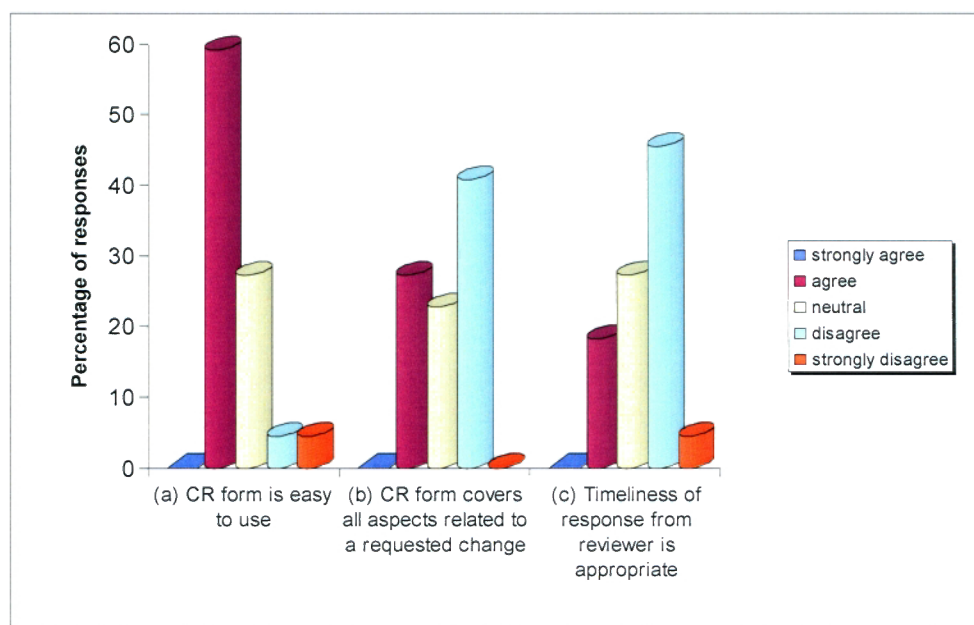


Figure 5.4 Agreement with the CCP and CR form

Assessing the results illustrated in the Figure 5.4 shows a less positive result for the issues regarding '(b) *the CR form covers all aspect related to the proposed change*' and the '(c) *timeliness of response from the reviewers is generally appropriate*'. As shown in Figure 5.4, 41% of the participants disagreed that the existing CR form submitted to the project generally covers all aspects of a proposed change (b). This finding supports the preliminary findings on the limitation of the CR form described earlier. The existing CR form does not fully cover the essential fields for change information, such as the rationale for making a change, other software artefacts (work product) impacted by a change, estimated effort for individual change.

The timeliness of the reviewers' response is another problem faced by most initiators. 50% of the participants disagreed (45.5% disagreed, 4.5% strongly disagreed) that the reviewers' responses to the proposed CRs were appropriate. This is not a surprising result since some of the change participants stated that '*... CR takes too long to be approved ...*'. Regardless the change types, the case study analysis indicated that the average time to approve a CR is approximately 20 days.

Interestingly, having disagreed with item (b), it seems that the initiators agreed on the fact that the CR form is easy to use. Most of the participants (60%) agreed that the '(a) *The CR form is easy to use*'. The existing CR form is a simple form; however, it does not provide essential fields to capture change information in more detail.

5.2.3.4 The Reviewer's (Approver's) perspectives

In the existing CCP document, the review and approval activities of the proposed CR were combined. The reviewer has authority to approve or reject the proposed CR. The project manager is responsible for tracking the progress of the CR and acting as a moderator during the review and approval activities.

Twenty-one (80.8%) responses were collected from change participants who had been involved in reviewing or approving CRs. Figure 5.5 shows the participants' agreement on the issue of rationale for change and the adequacy of change information.

In the context of the rationale for a proposed change, 43% of the participants were neutral and (33%) agreed that ‘(a) *the rationale for a proposed change stated in the CR form is sufficient*’. This response is contrary to the finding reported by the researcher, in which most of the proposed CRs have insufficient information on the rationale for making a change. The survey results show that less than 20% supported the researcher’s findings (9.5%: disagreed, 9.5%: strongly disagreed). This discrepancy is possibly because the participants (*Reviewers*) might misunderstand the meaning of ‘rationale for change’ (as stated by the researcher).

Another interesting issue highlighted in this survey result was *the adequacy of change information in the CR form if the proposed change will not be implemented* (b). Figure 5.5 shows 52% of the participants disagreed (47.6% disagreed), 4.8% strongly disagree) that the change information related to this issue is adequate. This highlights the importance of change impact information and implementation details to be included in the CR form. These change details can help the reviewers/approvers in assessing the consequences of the proposed change if the change will or will not be introduced.

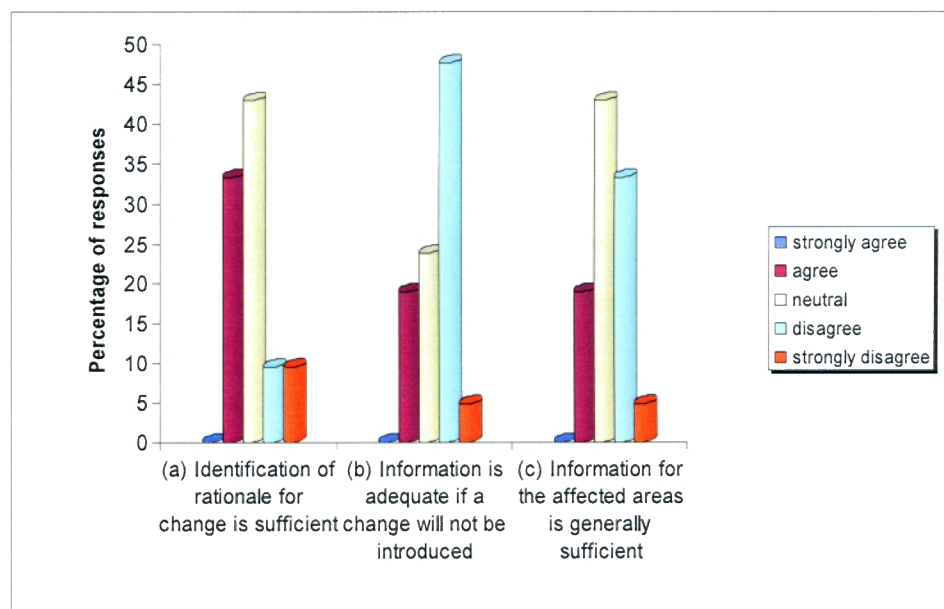


Figure 5.5 Participants’ Agreement on the aspects of CR review

Feedback was also collected from the participants in the context of ‘*change impact information* (c)’. The impact analysis issue is the main concern of the reviewers and the findings presented here support the researcher’s claim on the

limitations of the CR form described in Section 5.2.2 (item 5.2.2.4). 33% of the participants disagreed that the change impact information stated in CR forms is generally sufficient. This supports the previous assessment results reported by the researcher. Most of the change requests analysed in this study provided little information about the impacted areas of the changes. However, 43% of the participants have a neutral response toward this issue which indicate that they were probably not aware of the areas (other software components) impacted by the change.

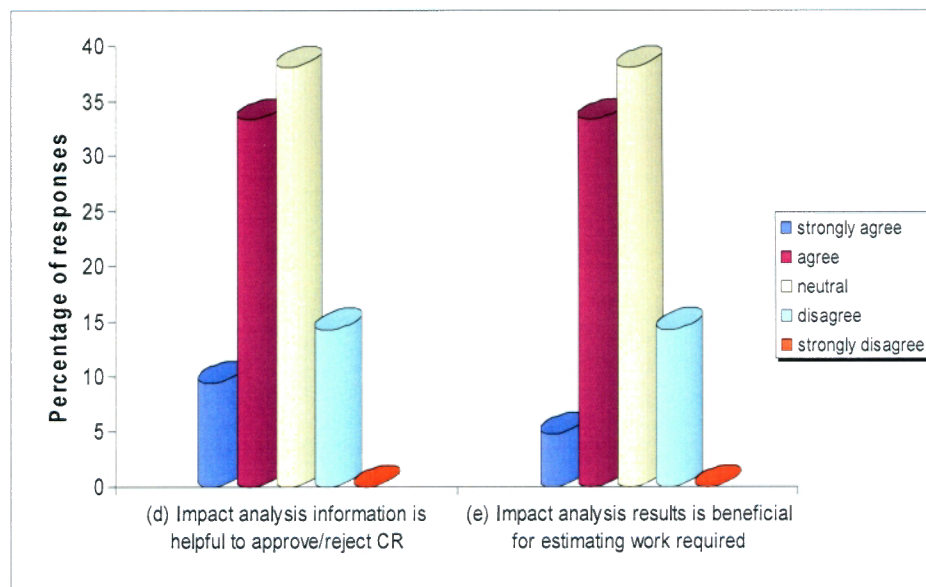


Figure 5.6 Participants' Agreement on the change impact information

Change impact analysis is an importance activity in managing requirements change. Feedback was also collected from the participants on the benefit of change impact information during the change process. Examining the results shown in Figure 5.6, 38% of the participants have a neutral response on both issues: (d) the *impact analysis information is helpful* in assisting the reviewers to *approve/reject CR*, and (e) the *impact analysis result is beneficial for estimating work (effort) required* to implement the change. Similarly, 33% of the participants agreed with these benefits of the change impact information.

5.2.3.5 The Implementer's perspectives

Figure 5.7 illustrates data from 19 (76%) responses on the question of *'the extent to which you agree/disagree with the aspects of CR implementation'*. The aspects of the CR implementation include the agreement on the written information stated in the approved CR sufficient for the change implementation and the effectiveness of change communication among the change participants who are involved in the implementation activity.

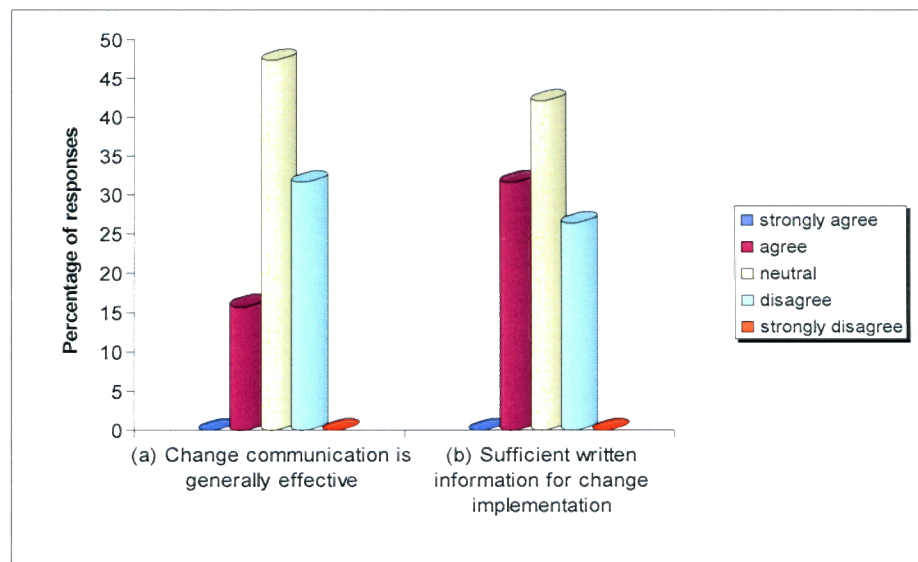


Figure 5.7 Participants' Agreement on the aspects of CR implementation

There seem to be mixed responses on the two issues shown in Figure 5.7. Most of the participants (42%–48%) are neutral about both issues: (a) the *change communication is effective* and (b) the *written information in the approved CR for change implementation is sufficient*, whereas only a few participants disagreed (26%–32%). The latter support the researcher's findings that there was a lack of information on the change implementation details found in most CRs and the change communication was not effective in terms of communication among the parties (project members) who are involved in the implementation. The results of change process assessment also showed that the implementers or development team members only relied on email communication and the awareness of each party to inform the other parties involved.

5.2.3.6 The Verifier's perspectives

Based on the existing CCP document, verification activity is carried out by the Quality Assurance (QA) team. However, the other project members, i.e. Project Manager or Management Leads, are also often involved in performing this activity.

The survey data from 11 (42.3%) responses provide the verifiers' perspectives on *the easiness of tracking the changes made to the project documents* that were based on the information available in the CR form. The verifiers' responses are illustrated in Figure 5.8.

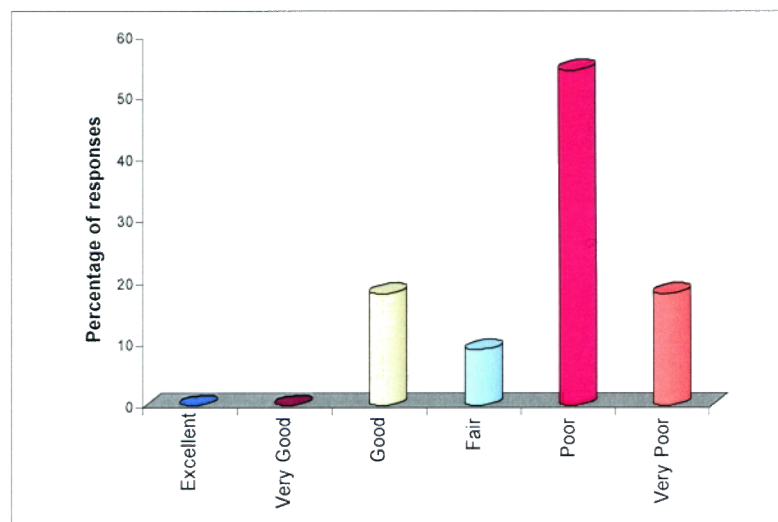


Figure 5.8 The easiness of tracking the implemented Change Requests

This finding is surprising since majority of the verifiers (54.5%) gave a '*Poor*' rating for tracking the changes and verifying the implemented changes. In addition to this result, the QA reports suggest that the verification for several implemented CRs is difficult to perform due to the lack of implementation details in the CR form. Furthermore, there is no traceability link established between the impacted areas and the relevant CRs. This survey response support the researcher's findings on the limitation of the CCP and CR form.

5.2.4 Phase_4: Summary of the Change Process Assessment

The Change Control Process is a core process in software development that supports the project in controlling changes to the project deliverables, including requirements. In this phase, the CCP document and CR form were examined and the usage of these documents in practice was observed during the case study period. The findings on the change process assessment identified by the researcher are consistent with the findings derived from the change participants' perspectives (pre-survey results). This evidence clearly emphasises that in order to better manage changes in the software development environment, the support process, such as Change Control Process, should be established and closely followed. Furthermore, the CCP should be developed according to the organisation needs. As the only means to raise a change, the CR form should be designed and structured well to capture as much information as possible and to guide the *Initiators* in writing better change descriptions. In addition, the change information, such as the severity of the change (change impacts), delivery schedule impacted or estimated change effort, are essential parts of the CR form to be used by the reviewers or the project manager in assessing the importance of the proposed CR.

The process assessment findings have instigated efforts to update and improve the CCP document and CR form, which also aligned with the GDS process improvement initiatives. The problems identified during this case study served as inputs for the development of a new CCP. The following issues, as well as those described earlier, are addressed in the new process document and CR form:

1. Lack of detailed information of the change impacts
2. Lack of change implementation details
3. Insufficient details to plan and schedule changes
4. No traceability link between the CR and impacted areas
5. Clear roles and responsibilities of the change participants need to be defined
6. Improving effective communication and coordination of the change implementation by all affected parties

7. Reducing time to process each CR (from initiation to approval)
8. Need automatic tool to support the process.

Following up the *Phase_4* activity is the improvement of the existing CCP/ CR form and its implementation. The evaluation of the new process and form together with the utilisation of new tool are described in *Phase_6*.

5.3 Phase_5: Change Analysis_2

This section presents findings from research activities conducted during *Phase_5* (*Change Analysis-2*). As shown in Figure 4.3, this phase is a repetition of the change analysis conducted during *Phase_2*. However, in this phase the analysis was conducted on a different project, i.e. *Project Release_B*. The aim of the activity *Phase_5* was also to address the research question, *RQ3a: what sources of, reason for, types of changes and the impacts of requirements volatility are most significant in practice?*

Research continued by investigating RV in *Project Release_B*. This second change analysis contributed significantly to broaden our understanding of the requirements volatility problems. This is because the researcher could observe the project from its beginning until the end of the product development lifecycle. Besides analysing the project's change request data and relevant project activities, this case study is intended to reinforce the findings derived from *Project Release_A* data and to triangulate the case study findings.

The following sections revisit data collection methods used and analysis techniques employed during this phase.

5.3.1 Phase_5: Data Collection and Analysis Framework

The activities conducted in this phase are similar to those described in *Phase 2 – Change Analysis 1*. The only difference is, as shown in Figure 5.9, in the construction of the requirements change classification.

Previously, we developed the classification from the *Project Release_A* data and the *card sorts* results. The *Reason category* attributes of these two classifications

served as a reference for the researcher during the change request analysis conducted in this phase. The rationale for the proposed changes to this project release may have similar reasons as the previous release or different reasons could emerge.

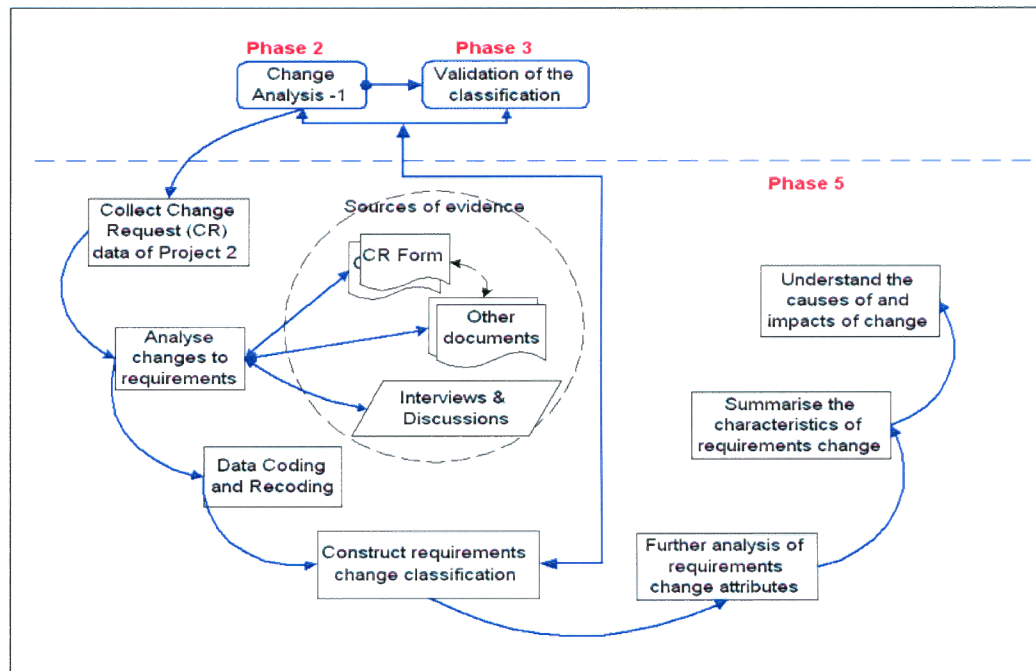


Figure 5.9 Data Collection and Analysis Framework - *Project Release_B*

Project Release_B followed the same Change Control process as the previous release to track and monitor the process of the change requests submitted to the project. The researcher spent extensive time in GDS site monitoring the change request process, analysing the change request information (see Appendix I), and tracking the information that was missing or not included in the CR form via discussions with the Initiator and the Project Manager. The researcher also attended weekly project meetings throughout the project's life. In these project meetings, the researcher captured the issues, challenges and difficulties this project faced. This valuable information was recorded in the researcher's case study notes to be used as additional information to the case study findings.

5.3.2 Phase_5: Empirical Findings

5.3.2.1 Change Request Arrival Rate

An insight into the arrival rate of the CRs submitted for this release was obtained during the initial analysis. The arrival rate includes the CR arrival and the CR against requirements (Requirements-CR). Figure 5.10 illustrates the arrival rate for both the CR types throughout the project duration (18–20 months). This arrival rate may be used as an indicator for the dynamic environment of the products being developed. Figure 5.10 also shows that requirements change requests were still arriving in the later stages of the development lifecycle. The X-axis label represents the month when the change requests were approved and integrated to the project.

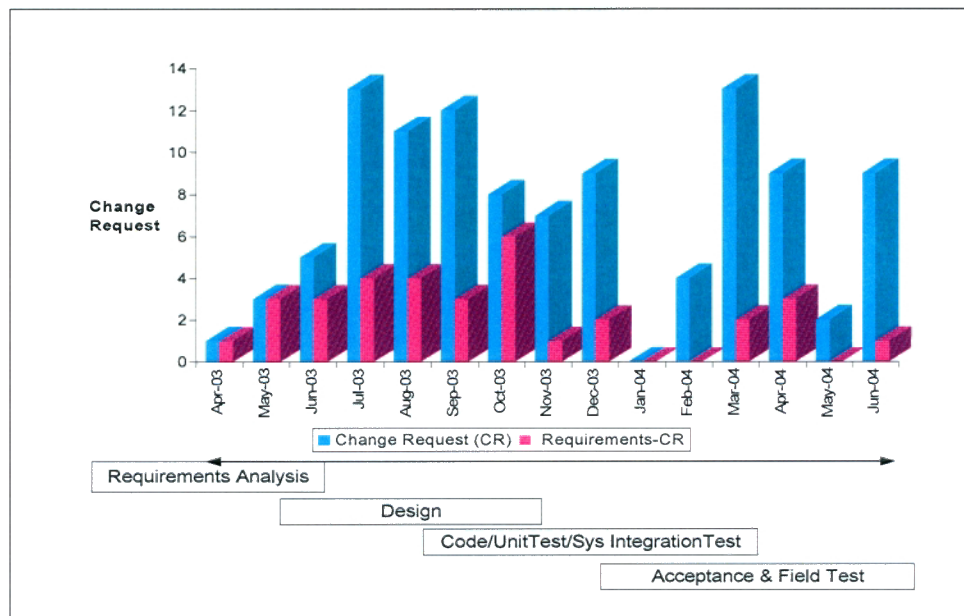


Figure 5.10 Change Request Arrival Rate for *Project Release_B*

A total of 122 change requests (CR) were collected, of which 29 CRs (22.48%) were requirements-related change requests (R-CRs). The arrival of the CRs in general increased sharply from July 2003 to September 2003, while the requirements CRs increased steadily from May 2003 and peaked during October 2003. This high number of the requirements CRs occurred at the end of the design phase or after design review were completed. The mapping of the CR arrival per month can be used by the Project Manager as an indicator of whether the project development process is still under control.

5.3.2.2 Measuring Requirements Volatility

As a result of change request analysis, a total of 147 requirements changes were identified. The changes include 26 high-level and 121 low-level requirements changes. These two levels of requirements were developed at GDS and traceability links between them were also established. The requirements volatility rate for these levels of requirements are computed and mapped against time (month) through the development lifecycle.

Of the 26 **high-level requirements**, 19 were modified for clarification purposes, three new requirements were added and five existing requirements were deleted. The modifications of requirements were instigated as a result of requirements capture and review (at the end of May 2003). The other reason for amending the high-level requirements was to enhance the functionalities of this product release, such as adding more detailed information regarding migration issues, printing functionalities in the new environment, or changing the inaccurate headings of requirements. The overall picture of requirements volatility rate for the high level of requirements is illustrated in Figure 5.11.

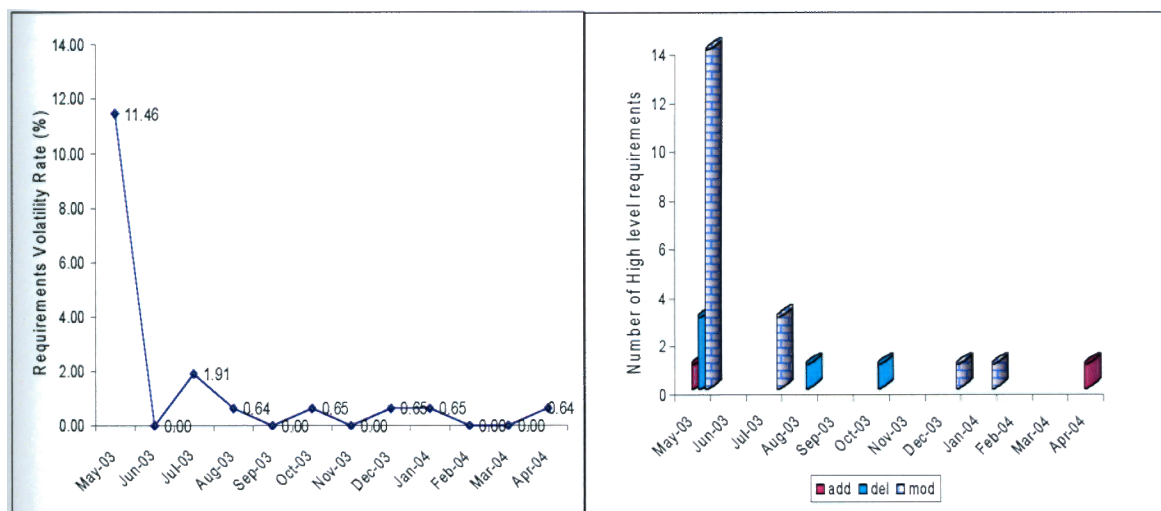


Figure 5.11 Requirements Volatility Rate - *Project Release_B* (High-level requirements)

The rate of changes for the high-level requirements seems stable toward the later phases of the development process, except in the earlier phase (May 2003) when requirements capture and analysis was completed (see the left side of Figure 5.11).

Interestingly, at the end of integration testing (April 2004), a new requirement was still added into the project (as shown in the right hand side of the Figure 5.11). This new requirement was intended to address a change request from the Marketing Group to add disclaimer information to install scripts and a clam-shell CD form of packaging. Based on the researcher's observation during regular project meetings, this requirement was subjected to long discussions and negotiations between the Marketing group (USA) and GDS due to legal issues and copyright and licensing problems, and the impacts of requirements on some areas, such as FS, Codes, Test specification, and Product Information (PI).

Any changes to the high-level requirements may or may not change the low-level requirements and vice versa. In *Project Release_B*, a substantial number of requirements changes were integrated into this project release. Of the 121 **low-level requirements** changes, 48 new requirements were added, 44 existing requirements were deleted or deferred for the next release, and 29 existing requirements were modified or amended. The overview of requirements volatility rates for the low-level requirements throughout the development process is illustrated in Figure 5.12 (the left side), and the detailed requirements changes in terms of change types is also illustrated on the right side of the Figure 5.12.

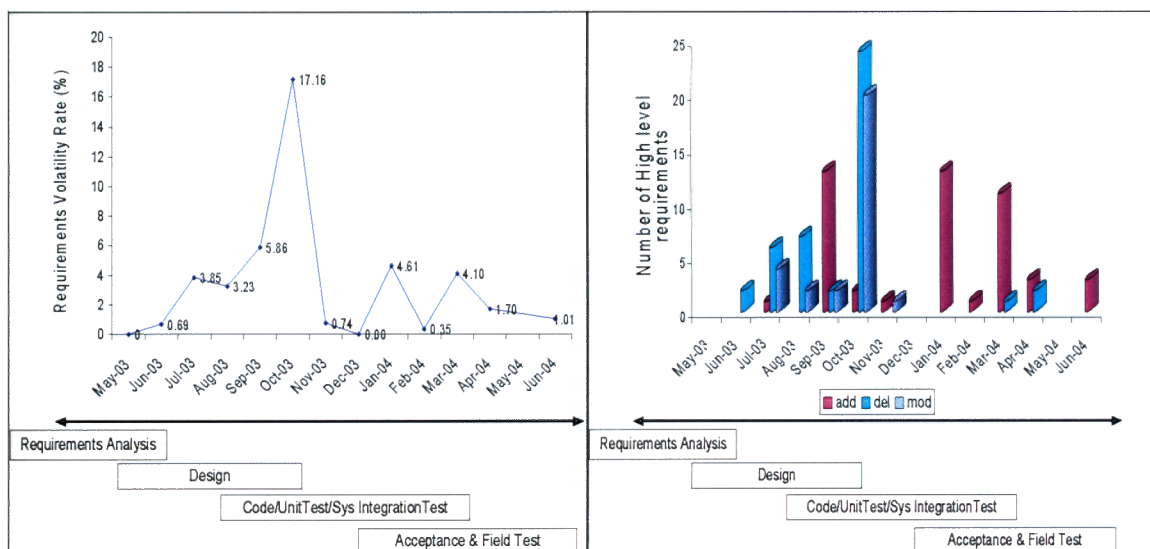


Figure 5.12 Requirements Volatility Rate - *Project Release_B* (Low-level requirements)

These diagrams clearly show that this project experienced extreme requirements volatility across the duration of the project. Although a significant number of requirements were deleted or deferred from this product release, there were substantial numbers of new requirements still being added to the project toward the later stages of the development process. The high peak of the requirements volatility level occurred at the end of the design phase (October 2003), when 24 requirements were deleted, 20 requirements were modified, and 2 new requirements were added. The large number of requirements that were deleted in the middle of the project's life was mainly instigated as a result of discussions between GDS management and the Marketing group when it was realised that some functionality could not be achieved or delivered for this release. The other reason for requirements to be deleted from this release was due to the availability of the third-party software or tools to support the functionality that was under development.

Figure 5.12 also shows that new requirements were constantly introduced and integrated into the project, i.e. from September 2003 to November 2003, and from January 2004 until April 2004, and June 2004. This finding indicates that some functional requirements for this release are not stable and potentially become a source of project risk. The volatility of requirement occurs not only during the design phase, but also in the later phases of software development. A further analysis of this occurrence at certain phases of the software development will be discussed in the next section.

5.3.2.3 A Classification of Requirements Changes

A classification of requirements changes has been used successfully in the analysis of the CR data and in the characteristics of requirements volatility during research activity *Phase_2*. In this section, the classification of requirements changes for *Project Release_B* data is presented.

Three components of the classification as defined in *Phase_2 (Change Analysis-I)* continue to be used in this phase. They are *Change Types*, *Reason Category*, and *Change Origin*. In this phase, the attributes of *Reason Category* and *Change Origin* are slightly different to the attributes elicited in *Phase_2*.

The preliminary classification of requirements change (RC-1, *Phase 2*) and software practitioner's classification (RC-2, *Phase 3*) have been used as a reference guide to construct the classification in this phase. Several additional attributes (in **bold** and underline fonts) for *Reason Category* and *Change Origin* emerged in this case study analysis. The following tables list the attributes of the classification.

- **Change Types**

Table 5.1 Requirements change types

Type	Description
4) Addition	Adding new requirements into the software or system being developed
5) Deletion	Deleting or removing existing requirements from the system
6) Modification	Modifying or rewording requirements statements

- **Sources of Change (Change Origin)**

This is the third component of the classification. Attributes of the change origin identified in this phase are as follows (Note: underlined attributes are the new attributes added to this release):

Table 5.2 Descriptions of Change Origin

11) Detailed Design Analysis	An analysis, which is not part of the design review, conducted by designers/ engineers/ analysts to improve or resolve design issues
12) Design Specification Review	A formal peer review process on design specification documents
13) Engineering + Management discussion	Irregular discussion between engineering group and management
14) Engineering + Marketing group	Irregular discussions regarding technical issues initiated by the engineers
15) Marketing Group	Representing market trend or current customer needs
16) Engineering Analysis	An analysis conducted by the engineer(s) during development phase or the analysis triggered by issues / problems identified
17) Functional Specification Review	A formal peer review process on functional specification documents
18) Project Management	Management considerations that are generally associated with the business goals and policy
19) Product Integration Test	Issues arising during system integration testing
20) Requirements Review	A formal review process on requirements specification document during the requirements analysis phase
21) Technical Team Discussion	Irregular discussions initiated by the technical team to resolve issues or problems faced

- **Reason Category**

This component is related to the categorisation of the changes in terms of the rationales behind the proposed change. The reasons for change identified in this study are as follows (Note: underlined attributes are the new attributes added to this release).

Table 5.3 Descriptions of Reason Categories

1) Requirement redundancy	Requirements overlap, or requirements are already covered by other requirements
2) Clarifying requirement	Changes to requirements text or rewording requirements text, i.e. remove unnecessary words or were discovered after design refinement
3) Missing requirement	Requirements that were not captured during the initial product definition
4) Implied unnecessary constraint	Changes to functionality that may trigger an unnecessary constraint on the user
5) Scope change (reduction)	Changes to requirements due to management consideration to reduce work in the current release or the functionality is beyond the current scope
6) Functionality enhancement	Maintaining or managing functionality or capabilities for the product releases, e.g. technical upgrade, functionality upgrade, security upgrade, etc.
7) Third-party software/tool influences	Requirements that cannot be fulfilled due to the availability/unavailability of the third-party software/tool, configuration/platform changes, or conformance to industry trend
8) Resolving conflict	Functionality conflict or sub-functionality interferes with the integrity of other functionalities that exist in the system.
9) Product Strategy	Changes to the product packaging, installation, or licensing
10) Testability	Inconsistency between requirements text and test scenario
11) Design complexity	The degree to which multiple components have a design that is difficult to implement or verify
12) Functional Integration	Resolving a lack of integration between two functions in the system deployment
13) Traceability	Requirements that have no traceability link to a particular software component or subsequent phases

The three components of the classification can be represented in a graphical view as shown in Figure 5.13. This overall view of the classification clearly indicates the reasons for requirements change as a result of particular development activities, such

as customer requests, action items from peer-reviews, team discussions, detailed design analysis, or management decision. While the mapping of these attributes is a useful way to understand the big picture of why requirements change during software development project, another way to utilise this classification is in the empirical analysis of change.

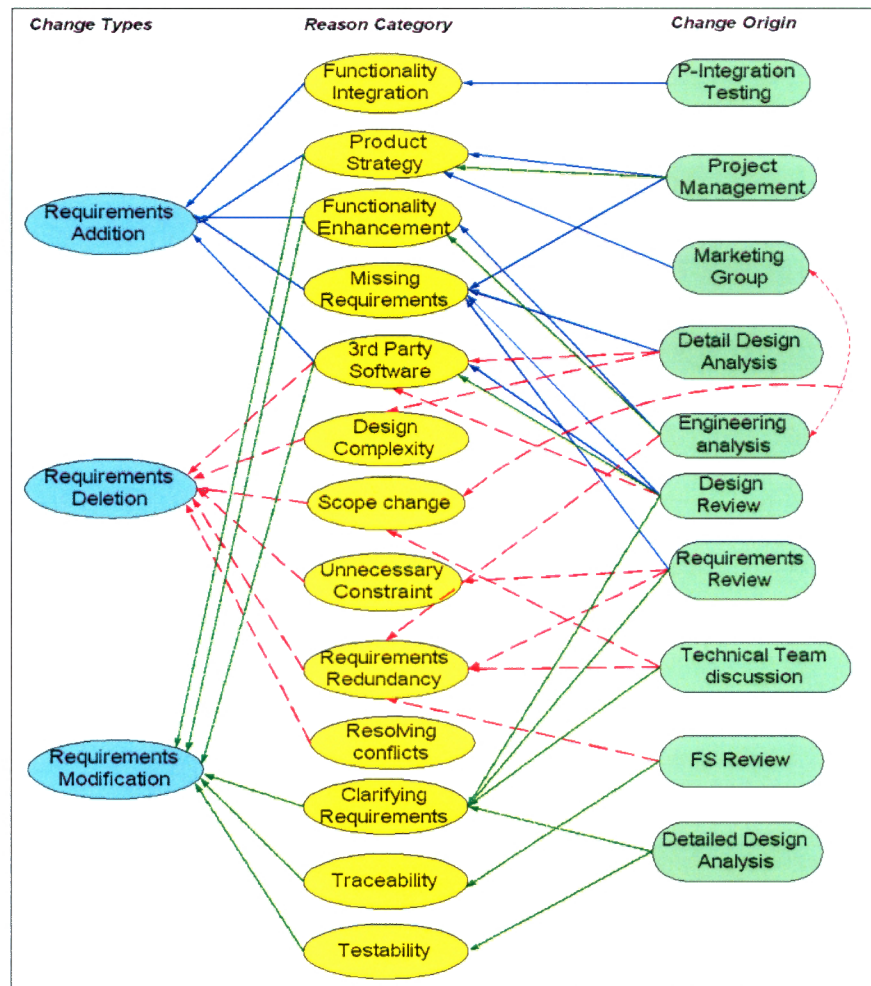


Figure 5.13 An overview of requirements change classification – *Project Release_B*

5.3.2.4 The Cost of Requirements Volatility – *Project Release_B*

Similar to the mapping produced earlier in *Phase_2*, the total estimated effort for the implementation of requirements changes in *Project Release_B* is also mapped to the rates of RV. Figure 5.14 illustrates this mapping, which provides an insight into how much requirements change costs the organisation throughout the project's life.

The total estimated effort per month presented here is the total effort spent for the three change types, i.e. additions, deletions, and modifications of requirements.

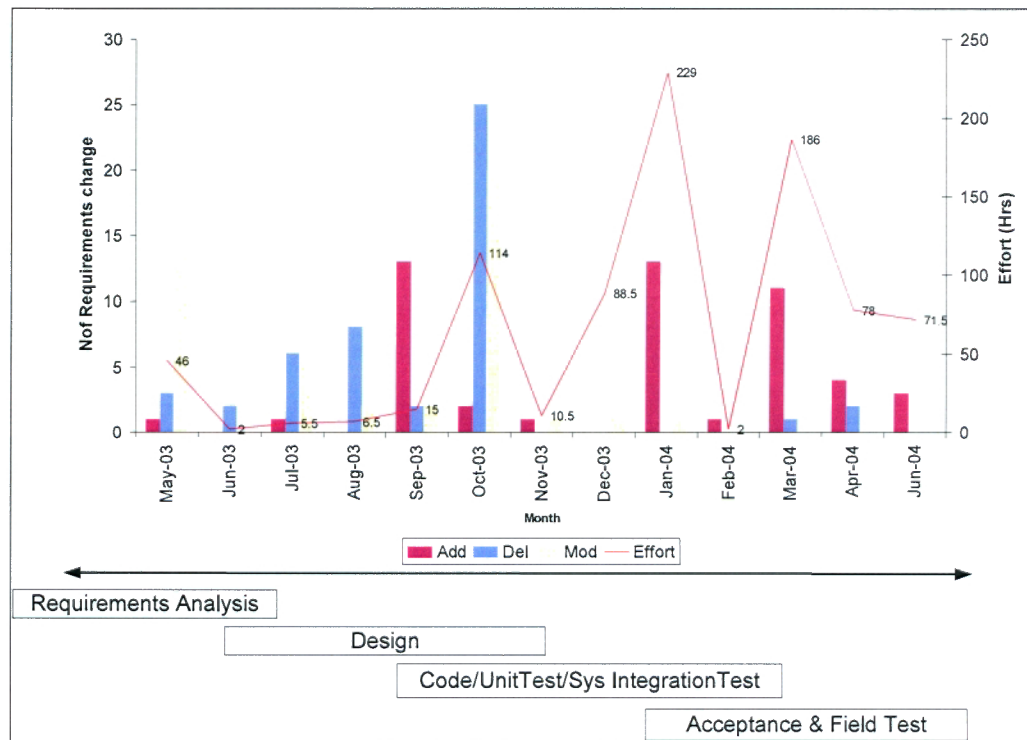


Figure 5.14 An overview of RV and total change effort spent throughout the development lifecycle of Project Release_B

Figure 5.14 also shows the distribution of requirements changes throughout the development lifecycle and the associated cost (labour hours). Examining the data shown in Figure 5.14, it indicates that although more requirements were changed early in the development lifecycle (e.g. up to September 2003), less effort were required to implement them. In contrast, when new requirements were integrated into the project at later phases, more effort was needed, particularly at the later design phase or during testing. When changes are unexpected or not part of the project plan, they will likely cause disruption to the project. In this project, new requirements were still being introduced during system integration testing and field testing (January 2004 and March 2004). Besides the number of the new requirements added, the complexity of the change impacts also contributes to the amount of rework (effort) required.

The analysis of these two variables (requirements changes and effort) reveals that higher numbers of requirements change do not necessarily imply a higher

development effort required to implement the changes. The later in the development lifecycle the changes occur, the more effort will be needed. Although these findings are brought out from only two software project releases, this trend of requirements changes cost confirms the results from previous studies (Boehm and Papaccio, 1988, Davis et al., 1993, Malaiya and Denton, 1999). In a comprehensive analysis of the cost of changes, other factors should also be considered, such as the change types (addition, deletion, or modification), the phases of the lifecycle where the changes occur, the size of the change impacts and the complexity of the changes. The analysis shows that the cost of RV tends to be high when new requirements are added in later phases. These late changes are likely to require substantial amounts of effort because of their size or the number of work products affected by the changes. In summary, requirements volatility in the later phases increases the amount of rework.

The above classification can also be used to identify the high cost of changes according to the reason for change and change types. Figure 5.15 presents the top two costs to this organisation according to the reason for change.

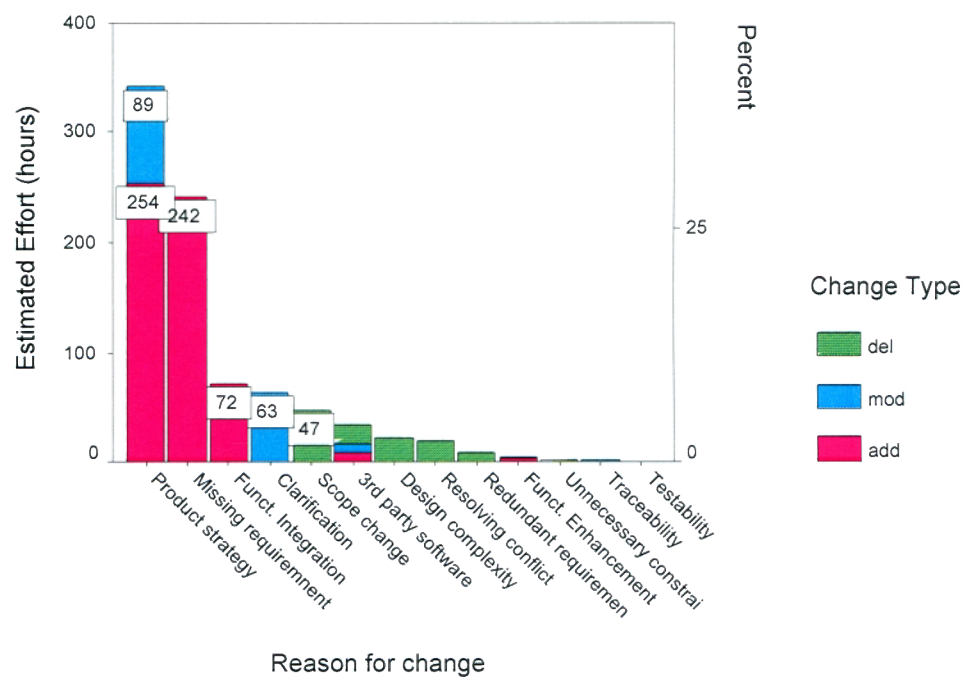


Figure 5.15 Pareto analysis of Total estimated effort by Reason Category and Change Types - Project Release_B

As indicated in Figure 5.15, the two highest costs (in effort) were for requirement changes related to *Product Strategy* and *Missing requirements*. This figure is based on the total estimated effort collected from the CR documents and is probably different from the actual effort. However, GDS can still use this information for future estimation and planning. The total number of requirements also affects the estimation of development effort (hours) in this figure.

5.3.3 Phase_5: Empirical Analysis of Requirements Volatility using Classification – Project Release_B

A further analysis to characterise requirements volatility was conducted. In this analysis, the types of requirements changes were investigated further and mappings between the reasons for change and change origin are presented.

5.3.3.1 Change Origin and Change Types

The following example demonstrates the strategic analysis of requirements change using the classification. This analysis is very simple to conduct using a Pareto chart. The first example is illustrated in Figure 5.16, which shows that a significant number of requirements changes were originated from *Design Specification Reviews*. The majority of the changes from this source were requirement additions and modifications. This is an interesting finding since new emerging technologies and new types of application were introduced in this project.

The other significant Change Origin attributes include *Requirements Review*, *Functional Specification (FS) Review*, *Detailed Design Analysis*, *Project Management* consideration, and *Engineering Analysis*. Pareto analysis indicates that most changes originating from *Requirements Review* are requirements modifications. This implies those requirements were being modified to improve their clarity. It seems that many requests for change originated from peer-review process activities. In other words, this finding re-affirms that reviews (of all kind) are effective mean of findings requirements problems or issues.

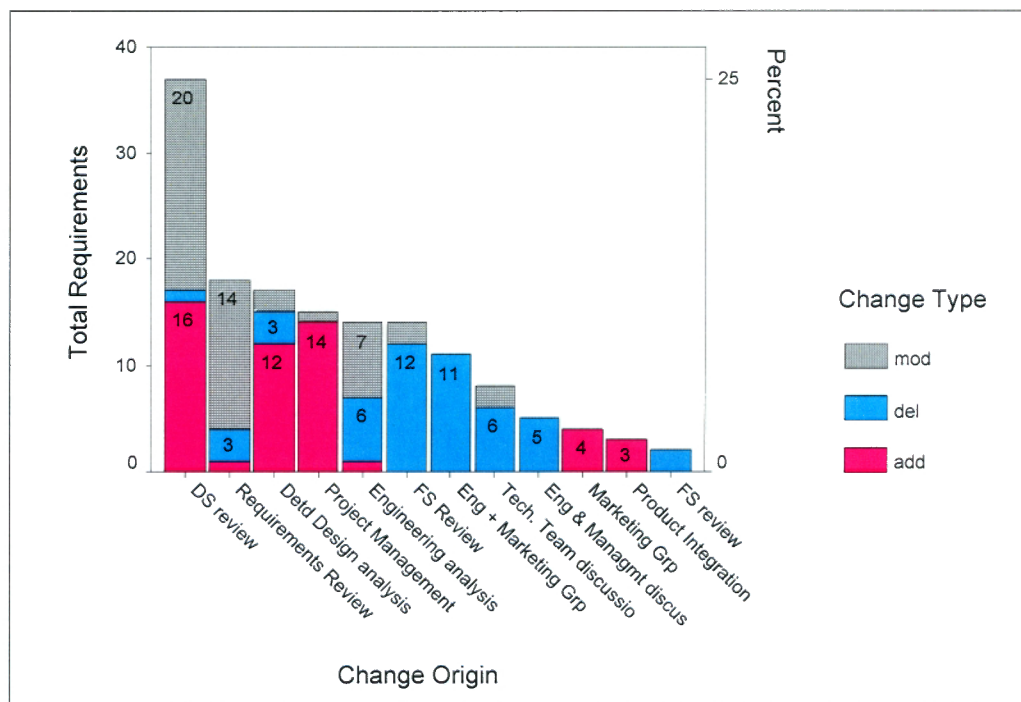


Figure 5.16 Total requirements change classified by *Change Origin* and *Change Type* - *Project Release_B*

5.3.3.2 Reason Category and Change Types

The next example is illustrated in the Pareto chart in Figure 5.17, which represents total requirements changes according to the *Reason Category*. Pareto analysis indicates that the top four reasons for requirements changes during *Project Release_B* were *Clarification*, *Missing requirements*, *Scope change*, and *Product strategy*. Understanding these reasons for requirements change is essential to identify opportunities for improvement initiatives on processes and products.

Based on the change types, Figure 5.17 also shows that the main reasons for modifying existing requirements were *Clarification* and *Functionality Enhancement*. The former is related to rewording requirements text to reduce ambiguity or removing unnecessary words to reduce redundancy. Since the product of this release is based on the functionality provided from the previous releases, maintaining and enhancing the current functionality is one of the project objectives. This enhancement of the product then resulted in modifying existing requirements and adding new requirements.

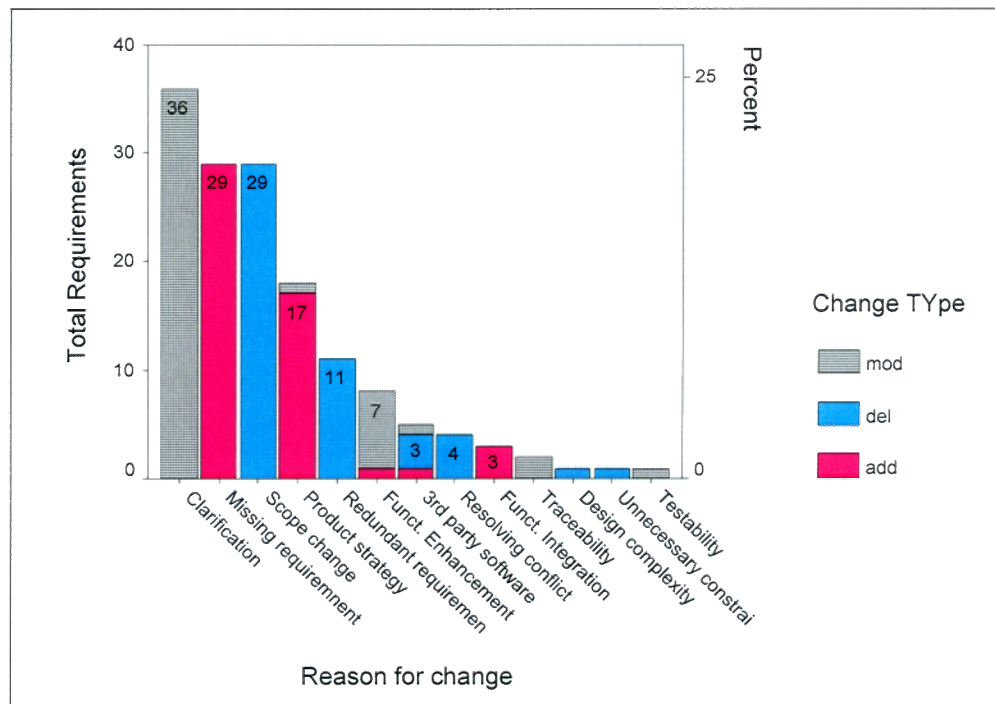


Figure 5.17 Total requirements change classified by Reason Category and Change Type – Project Release_B

Surprisingly, a large number of new requirements were added to the project due to ‘*Missing requirements*’. These requirements were missed either because they were not captured during the product definition (requirements analysis) or there were certain functionalities that existed in the system design with no requirements linking to them. When the project manager of this release was asked about the missing requirements, he pointed out that these requirements were required to address the new component technology (‘*Windows.Net*’ platform) being introduced into the project. He further added that “*no one knew in the beginning that we would need these requirements...*”.

These *missing requirements* were identified during the peer review activities on the design documents. It can be speculated that they were not captured earlier due to the development team’s lack of understanding of the product domain, particularly during the development of *Project Release_B* where a new software technology and a new product were introduced. This finding also demonstrates that it is impossible to know all the requirements upfront, so that the requirements specification will always be incomplete (Zowghi and Gervasi, 2003). Furthermore, this finding

reaffirms that ‘completeness’ of the requirements specification is the most difficult specification attribute to define (Davis et al., 1993).

As described earlier, out of 121 low-level requirements, 44 existing requirements were deleted from this release or deferred for the next release. Figure 5.17 also shows the main reasons for requirements deletion were *Scope change* (reducing work due to management commitment or functionality is beyond the current scope), *Requirements redundancy* (removing overlapping requirements or requirements that are already covered by other requirements), *Resolving conflict* (functionality or sub-functionality that interferes with the integrity of other functionalities that exist in the system).

Classification is a strategic mechanism to analyse problems and identify risks. In the next section, utilising the classification attributes to explore in more detail the underlying causes of requirements volatility and its cost in term of effort is presented.

5.3.3.3 Requirements Addition and Reason Category

The reason category attributes identified in *Project Release_B* were a reflection of the objectives of this release as well as the market and customer needs, and the software developer’s understanding of the applications. As illustrated in Figure 5.18 and mentioned earlier, the most significant reason for adding new requirements are *Missing requirements* and *Product strategy*. As additional information that will be useful to understand the cost of requirements change, Figure 5.18 also illustrates the amount of effort (hours) required to add new requirements for each reason category.

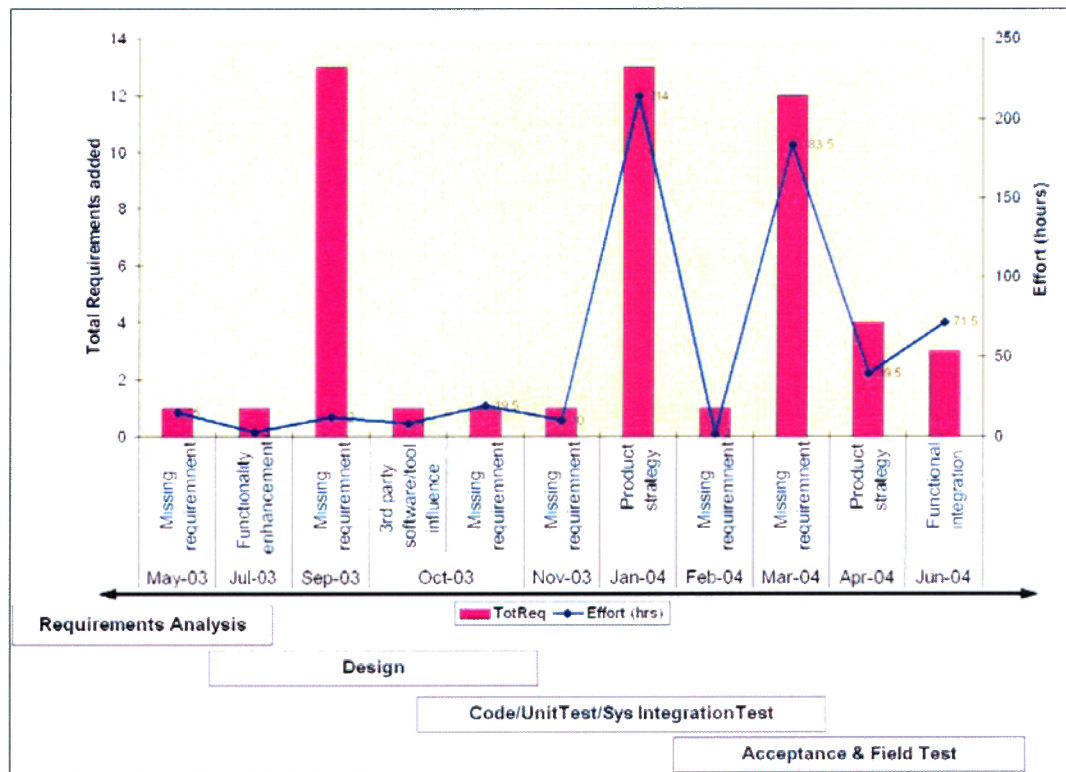


Figure 5.18 Requirements additions, reason categories, and change effort
(Project Release_B)

Surprisingly, missing requirements were detected almost at every phase of the development lifecycle. These requirements were not captured during the initial product definition and subsequently not addressed. Introducing new requirements in the later phases of the lifecycle is claimed to be expensive due to its greater impact on multiple components and other established documentations (Boehm and Papaccio, 1988, Davis and Leffingwell, 1999, Malaiya and Denton, 1999). The findings shown in Figure 5.18 demonstrate that adding new requirements in later phases (e.g. January'04 and March'04) required substantial effort (hours) because the new requirements impacted multiple software components and other existing product documentation. Thus, the implementation of these new requirements required much rework and impacted the planned schedule. Substantial numbers of new requirements that are still being integrated in the later phases of the development should be considered as a warning sign for GDS management because the change impacts in this later stage of development would be very expensive.

Another significant reason for adding new requirements to the project is *Product Strategy* (January 2004 and April 2004). This reason is related to customer expectations or market needs. During the development of this release, the Marketing group had requested different forms of product installation package. The products are packaged in a *CD clam-shell* and the application tools and application environment are merged into one installation package. Based on the researcher's observations during weekly project meetings, this issue was the subject of long discussions and negotiations between GDS management and the Marketing group (since November 2003).

Added requirements during the software development phase were caused by *Functionality enhancement*, *third-party software/tool influence*, and *Functional integration*. Added requirements in the early design phase (July'03) were triggered by the need to improve the application's *security* functionality. In October 2003 a new requirement was added to address specification changes on the *View/Difference/Merge* functionality. In the very latest phase of the software development lifecycle (June 2004), a new requirement was added as a result of an issue raised during the product integration test. This requirement was needed to improve the integration between two functions during system deployment.

5.3.3.4 Requirements Deletions and Reason Categories

Deletions of requirements during the development lifecycle often cannot be avoided. During the development of *Project Release_B*, significant numbers of existing requirement were deleted from this release because of *Scope change* (see Figure 5.19). *Scope change* is also referring to scope reduction, which is driven by project management and engineering team considerations. Basically, it is caused by the inability of the project team to fulfil or deliver certain functionality for this release. Lack of resources to work on functionalities that were not available in the previous release also triggered the scope to be changed (October 2003).

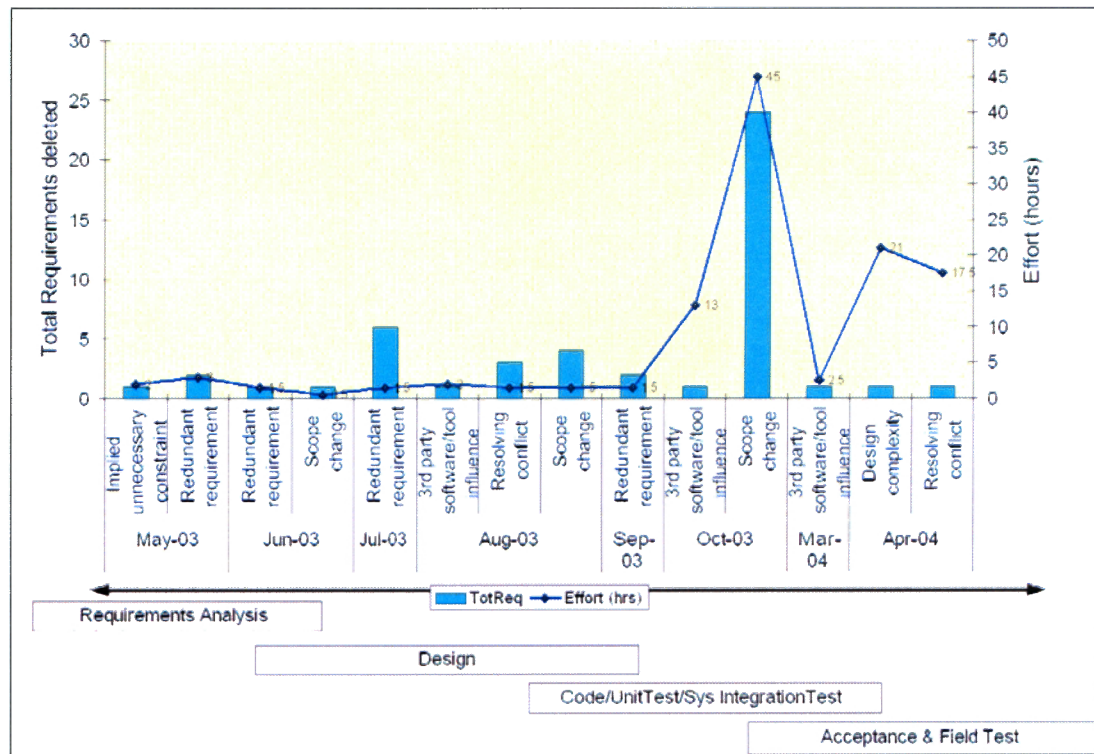


Figure 5.19 Requirements deletions, reason categories, and change effort (Project Release_B)

As shown in Figure 5.19, removing the existing requirements in later phases of the development lifecycle may cause disruption to the project. In addition, late removals of existing requirements are likely to cost more than those in the early phases. This supports the previous finding that the later a change (requirement deletion) is addressed, the greater the cost or risk (Malaiya and Denton, 1999).

The other reason for deleting existing requirements is to remove *Requirements redundancy*. A number of redundant requirements were identified during formal reviews/inspections of the requirements specification document (May'03) and functional specification document (Jun'03). The peer review process conducted here seems to work well in identifying early changes. Some redundant requirements were also detected during the design phase (Jun'03–Oct'03). The following are examples of two requirement statements, in which the first one was deleted since it was already covered by the second requirement:

Requirement #1:

"The installation of on-line help and product documentation for Developer will be incorporated into the Developer installation program" (deleted, as a redundant requirement)

Requirement #2:

"A setup program must be provided to allow users to install the CILKN product, comprising of Developer, Builder, Debugger, VersionControl, RT Administration Tool, Support Libraries, Protocol Adapters, Comp Enabler and Product Information. Setup and installation must follow common Microsoft guidelines"

The case study analysis identified other reasons for deleting the existing requirements: *Resolving conflict*, *third-party software/tools influence*, and *Design complexity*. Although these changes occurred in a later phase that would place the project at risk, the deletion of these requirements could not be avoided. The development team argued that these existing requirements would actually require extra work to implement them, thus increasing the risk of the project not meeting its planned schedule. It is understandable that this kind of action is taken and the requirements deferred to the future releases.

The following is an example of requirement deletion for the reason of *Resolving conflict*:

"The CILKN application deployment process must create and initialise a user's database"

It was realised by the developers that the words '*create*' and '*initialise*' triggered functionality conflicts during the implementation phase (code and unit testing). These words in the context of database commands caused the software developers to redesign several parts of the software component. In order to resolve this problem, the development team decided to remove the requirements from this release or to defer them to the next release. This problem is also associated with the complexity of the current design, which needed further refinement.

During system integration testing (April'04), an existing requirement was deleted from this release due to *Design complexity*. This requirement was identified by the development team before the integration test had started. It was estimated that a great deal of effort would be needed to implement this requirement because of the design and code complexity. The team predicted that the risk of not finishing the work within the planned schedule of this release was very high.

Interdependencies between the applications being developed and the availability or capability of the *third-party software/tools* can also trigger requirements deletions. For example, the third-party software does not provide capability for some functions stated in the requirement specifications. Thus, the decision was made to remove this requirement from this release or defer to the next release. The following is an example of the third-party software item ('SQL Editor') that has no support for *syntax highlighting* or *auto-completion*:

"The Developer SQL editor must support syntax highlighting and auto-completion"

5.3.3.5 Reason Category and Requirements Modification

The main reasons for modifying requirements in this release were *Clarification* and *Functionality enhancement* (see Figure 5.20). Modifications made to existing requirements are generally related to improving the clarity of requirements statements or reducing ambiguity by rewording or removing or replacing some words. Similar to the effort for requirements deletion as shown in Figure 5.19, modifying requirements in the later phases is also likely to consume more effort compared to modifications in the early phases. Figure 5.20 suggests that a few requirements that were modified on December 2003 for *Product Strategy* reasons required substantial effort (88.5 hours). The GDS organisation can learn a lesson from this evidence to pay more attention to changes that are related to this reason.

The following is an example of a requirement that was modified without changing its meaning:

"The user must be able to install and run multiple instances of Windows runtime systems."

Note: '*instance of*', added to the requirement text for clarity.

A requirement modified by changing the context would be like the following example:

*“It must be possible for **CILKN** to co-exist on the same workstation as an installation of any previous version of L3x Devp **and L3x Windows Runtime**”*

Note: “**CILKN**” and “**and L3x Windows Runtime**” were added to the requirement text. This addition clearly specifies the product (**CILKN**) and two application environments (*L3x Developer* and *L3x Windows Runtime*). The findings reveal a modification of requirements that changed the meaning of the requirement text and created more work.

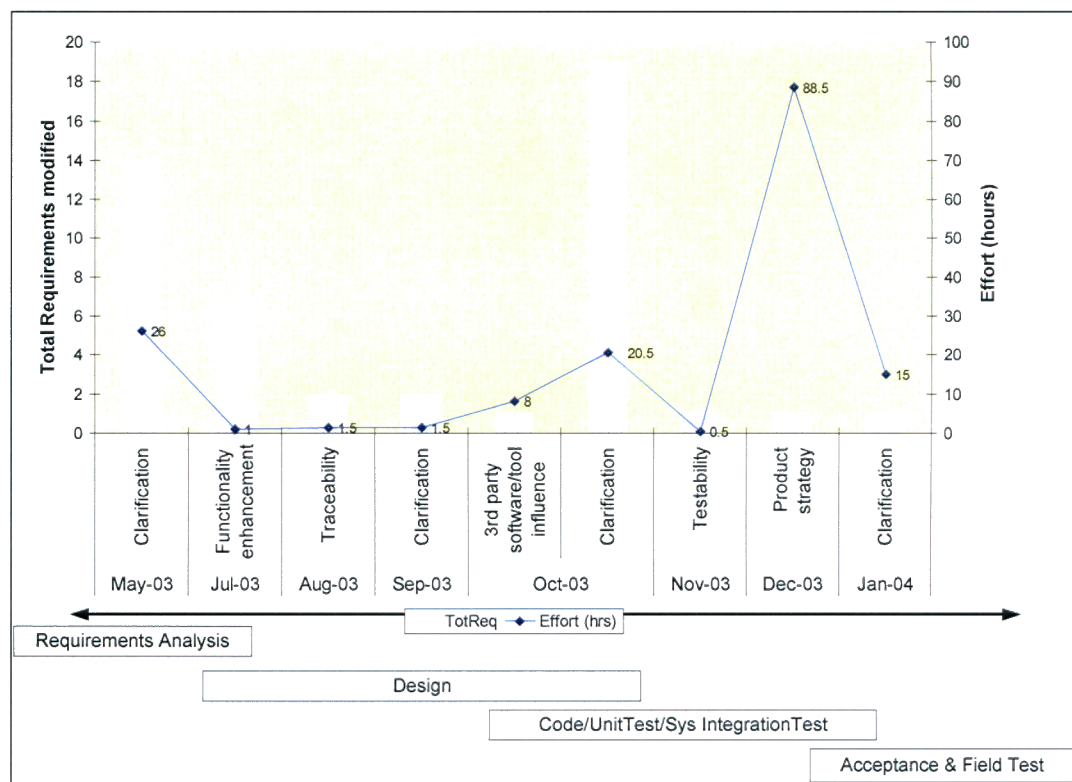


Figure 5.20 Requirements modification, *reason categories*, and change effort (*Project Release_B*)

Modifying requirements for the reason of *Functionality enhancement* was intended to maintain the functionalities or capabilities of the current product release. Some requirements were amended in the early design phase to address, for example,

application security that was not available in previous releases. The following is an example of a requirement modified for this reason:

"Secure communications transport between clients and servers must be provided possible for a Windows Runtime system using Web services"

Note: transport and using Web services added to the requirement text and the word '*provided*' is replaced by '*possible*'.

Other reasons for modifying requirements include *third-party software/tools influence, Product strategy, Traceability* and *Testability*. Modifying requirement for *Testability* is related to an inconsistency between the requirement text and the test scenario. In this case, the test scenario is correct and the requirements text was amended.

Traceability is associated with creating or updating traceability links between requirements, FS and DS. This requirement traceability link is recorded in the requirement management tool (RequisitePro™).

5.3.4 Phase_5: Additional Findings on the Change Control Process and the CR Form

The existing CCP and CR form were still used during the development of *Project Release_B*. However, the *Reviewers* were required to complete the review activity within a certain period of time. They introduced a *Sunset-Clause*⁹ of two weeks to review each CR. When the proposed change request was sent out for review, the project manager set a time frame for the *Reviewers* to complete the review activity. The use of the sunset clause was intended to speed up the process of change request review and approval.

The researcher was able to observe the way the team members of *Project Release_B* implemented the existing CCP and used the CR form to request a change. There is no automatic tool used to support the change control process. Microsoft Word, Excel, and email are the only tools that support the process of a proposed CR. Therefore, this project, like *Project Release_A*, experienced ineffective

⁹ Sunset-clause: "If an approver's response is not received by review due date, it will be considered approved by him/her"

communication and coordination of change information among the development team members.

Although the CR form was used to submit a change, the change information stated in the form was generally still inadequate. The researcher initiated half-hour weekly meetings with the project manager during the first six months of the project duration to obtain more information about the submitted change requests. This meeting was different to the weekly project meeting that the researcher attended every Thursday (1.5 or 2 hours). Findings from the inspections of the CR forms indicated that most of the submitted CRs lacked information on the impacted areas and effort estimation. The initiators, who were also the members of the project, seemed unable to identify the impact of the change on other dependent deliverables (multiple platforms and multiple components), testing, product documentation, or work products such as functional specification, design specification, or source code. In addition, there was no guideline or checklist established to help the project team perform a better change impact analysis. In order to overcome this lack of information on the changes' impact or effort, the researcher directly collected the information both from the project manager and the initiators or other relevant software developers whose areas were impacted by the change.

In the middle of the development process, some technical team members suggested having a "Change Notification" (CN) for simple changes that did not need more than two reviewers to approve them. A simple change refers to any changes to work products that have no impact on other dependent deliverables or teams. The objective of using CNs for those simple changes is to fast track and speed up the change approval and implementation. The project manager of this release allowed the project members to use the CN for a while. However, the Quality Assurance (QA) Lead then rejected the CN because it did not comply with the established Change Control process.

It is understood that this CN created some confusion among the development team. It encouraged them indirectly to use CN for any changes to work products, which of course contradicted the CCP. However, since the *Project Release_B* has adopted an iterative design and development approach, which allows them to develop a small subset of requirements, functional and design specifications

iteratively, rapid changes are expected. Therefore, a controlled mechanism for handling frequent changes to small parts of the work products is needed. The introduction of CNs to the change management has led to modification of the old CCP.

During the research activity of *Phase 5 – Change Analysis 2*, the researcher worked with the GDS management to update the change control process document and the CR form in order to improve the information captured on the CR form and to update the current Change Control process. In the new CCP document, the CN mechanism described above has been replaced with a type of change request (Type_C), which has a short process lifecycle. The new CR types are described later in Section 5.4.2.

5.3.5 Phase_5: Summary of the Change Analysis_2

This phase has covered the analysis of change request data from the *Project Release_B* database. A classification of requirements change is used to analyse the change requests. The rates of RV throughout the development lifecycle are measured and presented in a graphical form. The detailed characteristics of RV are also described. This leads to understanding the magnitude of each requirements change request and measuring the change effort. Interesting findings emerged from the mapping between the RV rates and the total amount of requirements change effort throughout the software development lifecycle. These empirical findings provide valuable information about the RV issues and enrich the existing findings of this case study.

5.4 Phase_6: Evaluation of the New Change Control Process and CR Form

This section describes the research activities and findings of *Phase_6* (see Figure 5.1). The activities were associated with the evaluation of the new Change Control Process (CCP) and CR form that were introduced at GDS. These documents have

been reconstructed to address the limitations and problems identified both from the case study findings and the first survey results.

Updating and improving the content of the existing CCP and CR form were aimed to improve the way GDS manages requirements volatility effectively in particular and manages software changes during the development lifecycle in general. The new CR form is expected to be used as a strategic tool to effectively capture and record more details of change information for strategic decision making purposes.

This phase was intended to continue gathering more information regarding the perspectives of the change participants on the usage of the new CCP and CR form. At the same time as the implementation of the process and form, an automated requirements management tool (Telelogic™ - DOORS) and change request tool (Telelogic™ - Change Synergy) were also introduced as supporting tools.

5.4.1 Phase_6: Data Collection and Analysis

A survey method with structured questionnaire was used as the main data gathering technique. Document inspections and participant observations were also considered. In this phase, the survey questionnaire (second survey) is a modification of the first survey questionnaire that was developed in *Phase_4*. Changes to the first questionnaire are only minor and relate to adding a few more questions to capture the usage of tool to support the new change process (see Appendix J).

The change participants who participated in this survey were GDS staff who had been involved in processing change requests, including initiating, reviewing, approving, implementing, or verifying CRs. Their names were collected from the change request records. Six months after the new CCP was implemented, the survey was conducted within a three-week timeframe. The survey questionnaire was emailed to the participants.

The survey data were analysed mainly using quantitative analysis with descriptive analysis in the form of means, medians, numbers and percentages. The findings accumulated in this phase will support the overall case study findings in developing effective strategies to handle requirements volatility.

5.4.2 Phase_6: Findings

5.4.2.1 Overview of the new CCP

GDS has restructured the process and form to control changes during the development project lifecycle. This new updated process and form is now implemented across GDS projects. The CCP is a single mechanism to control any changes to the baselined/approved work products that are put under Configuration Management. The new CCP contains an approach to classify three types of change request according to the work products impacted. Each type of change request is described as well as its process activities and steps.

Structuring the change process into the three types is a major improvement in GDS. It encourages the change initiator to specify the type of change will be made, the type and size of change impacts, and to nominate the appropriate reviewers. The descriptions of the three change request types and their flowcharts are presented as follows:

Type-A CR – has potential impacts on exposed functionality visible to external users, functionality committed to marketing, or externally committed milestone dates. Its impacts include changes to marketing requirements (MKR), technical requirements (TCR), and/or milestone delivery dates. A process flowchart that refers to this CR type is shown in Figure 5.21.

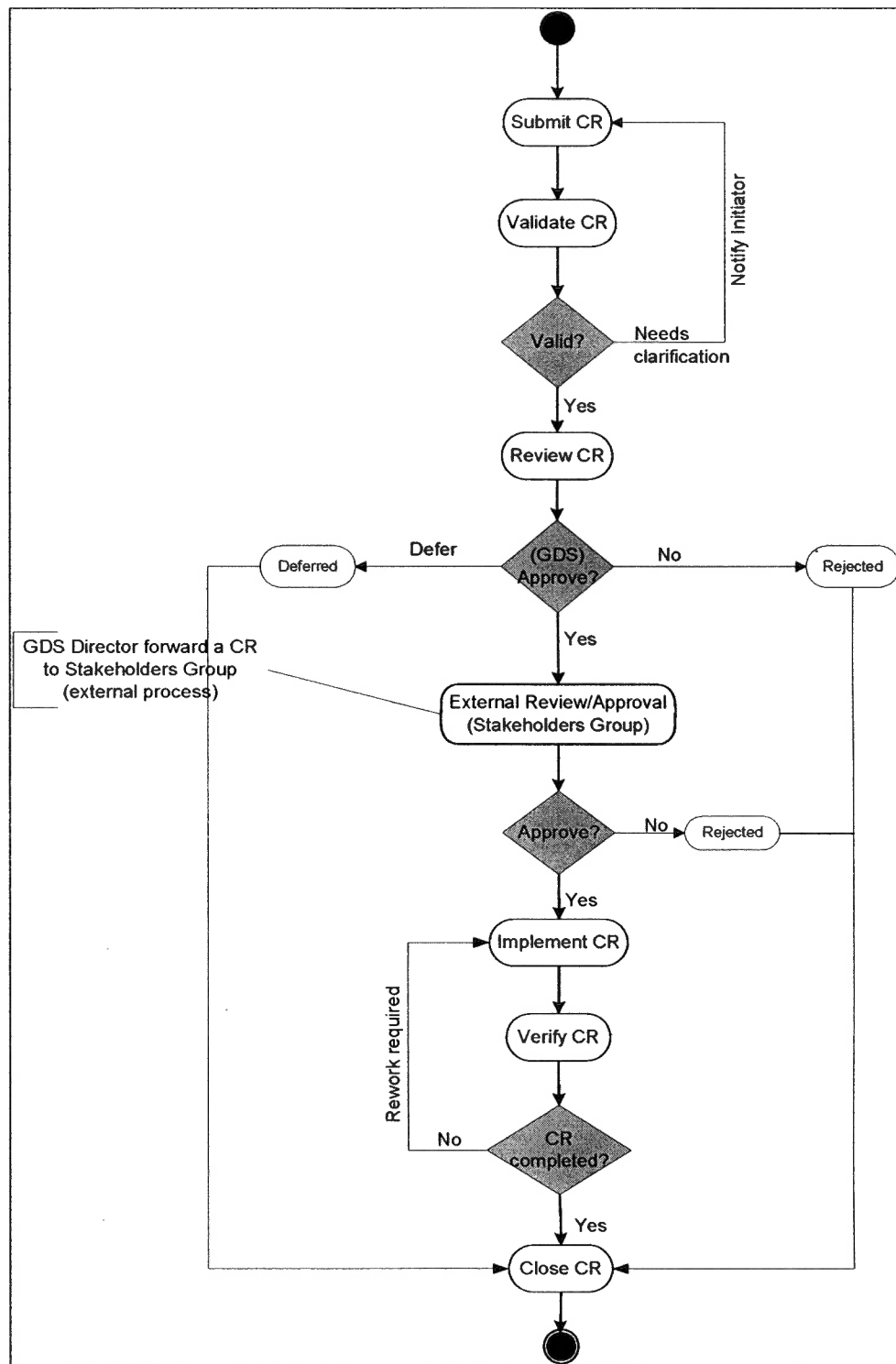


Figure 5.21 Type-A Change Request Process Flow (Source: GDS document)

- **Type B-CR** – has potential impacts on internal functionality and other dependent deliverables, such as changes to internal requirements (ITR), system architecture detailed design, design specifications of other subsystems. A process flowchart that refers to this CR type is shown in Figure 5.22.

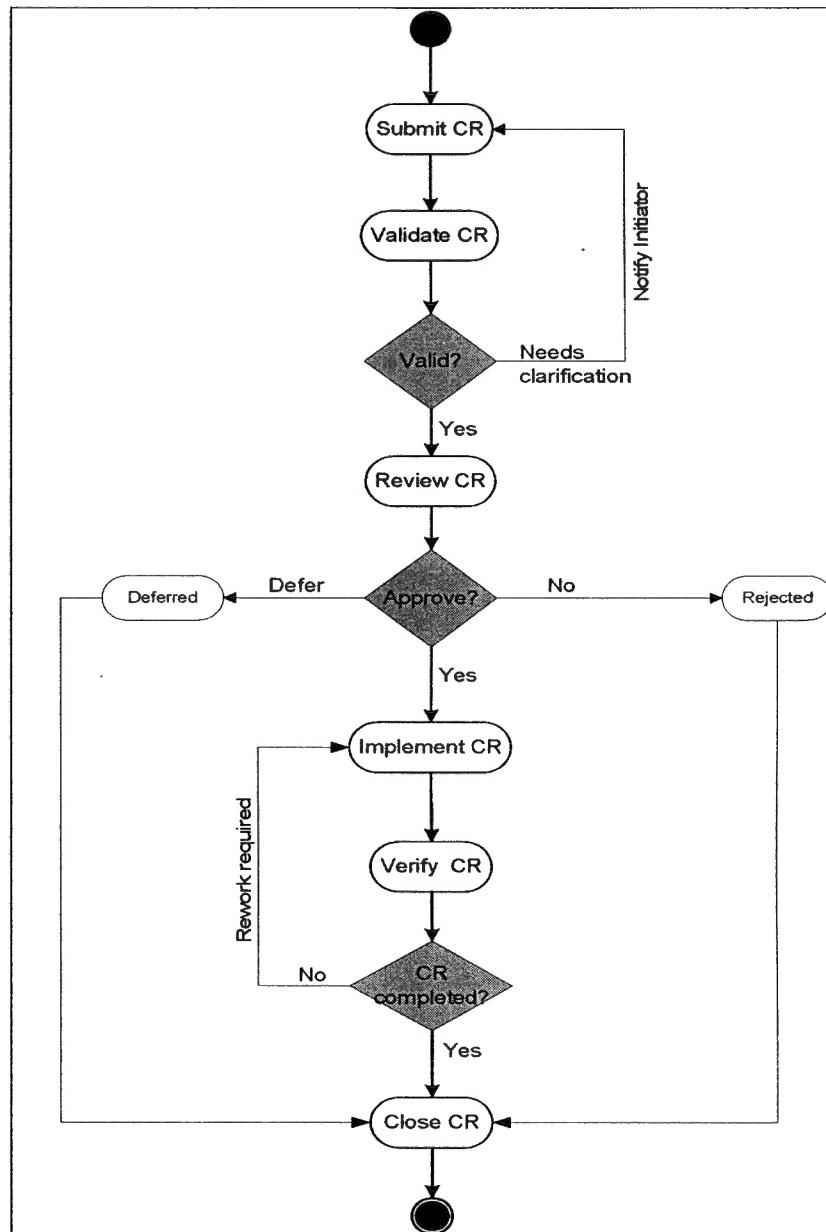


Figure 5.22 Type-B Change Request Process Flow (Source: GDS document)

- **Type C-CR** – is related to any changes to work products that have no impact on other dependent deliverables. For example, changes to sections in the functional specification document only, or changes to font properties in the design specification.

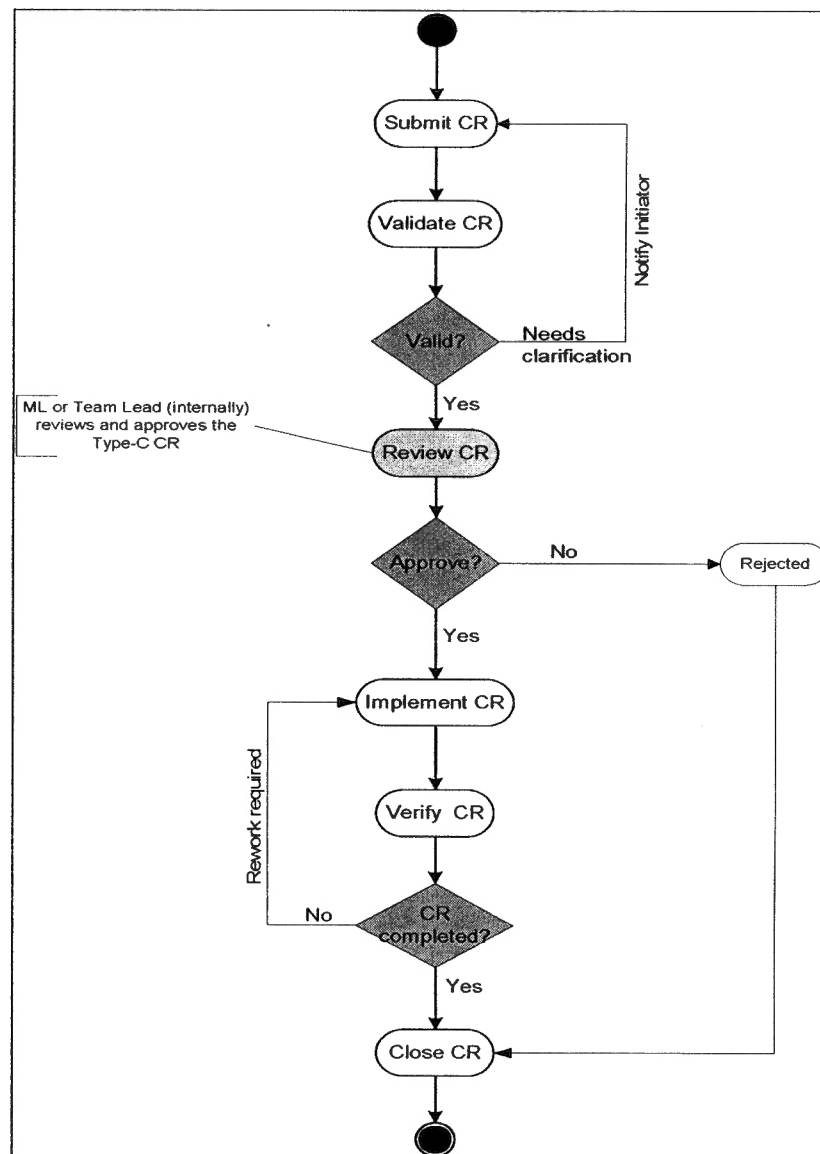


Figure 5.23 Type-C Change Request Process Flow (Source: GDS document)

The benefit of classifying a CR based on its impact is intended to assist GDS in choosing the appropriate *Reviewers* and prioritising CR implementation. It is understandable that the Type A-CR requires an external approver (the Marketing group in the USA) since its impact is visible to the user. This type of CR will first go

through the internal approval (in GDS) and then be sent for external approval (Marketing group). Thus, this CR type requires a long time to be approved and implemented.

Table 5.4 shows a summary of improvement actions made to the new CCP and the main issues that have been addressed.

Table 5.4 A summary of the existing CCP and the new CCP

Category of changes	Existing CCP	New CCP
Types of change	The process covers three types of change: 1) changes to project deliverables (RS, FS) and milestones, 2) minor changes to DS, 3) changes to Code, Each change type has a different processing mechanism, i.e. changes to DS are handled by peer-review/email, changes to RS and FS are handled via Change Request.	This new process distinguishes three types of change request (CR) based on the change impacts on the key deliverables: 1) Type A-CR 2) Type B-CR 3) Type C-CR All change types must be processed via the Change Request process.
Process activities	The process activities only relate to the first type of change (RS and FS)	The process activities covers all types of CR in details
Process flowchart	There was no process flowchart available	Three different process flowcharts defined for the three CR types
Roles and responsibilities	Inadequate information on roles and responsibilities in the process	Detailed descriptions of roles and responsibilities are described. A change administrator role is introduced and the level of review according to the CR types is established
Review deadline	No time limitation for the reviewers is given	'Sunset-clause' is applied with 10 working days maximum to review the CR.

The summary in Table 5.4 indicates that the new CCP has a more effective structure than that the existing one. The new CCP also provides a rigorous way to capture proposed changes to the project deliverables. In addition, the new tools are employed that can speed up the CR process activities. It is expected that the change management tool will solve the coordination and communication problems during change implementation since the tool provides email notification to the change participants for each change activity.

5.4.2.2 New Change Request Form and Tool

The existing CR form was redesigned to address its deficiencies. It is now an electronic form. It is integrated into the Telelogic™ Change Synergy system, which is a supporting tool for the process. Change participants who are involved in the CR process will get an-email notification at every stage of the process. The CR will not be processed to the next stage if it has not been completed. Everyone in the project can monitor the status of the CR and its schedule. Table 5.5 describes the improvement actions made to the CR form.

Table 5.5 A summary of the existing CR form and the new CR form

Category of changes	The existing CR form	New CR form
Change Fields	It is a simple CR form with few fields which are insufficient to capture complex change information. The fields are as follows: <ul style="list-style-type: none"> ○ Change ID ○ Project name ○ Dates (submission date, approved date, and implemented date) ○ Initiator name ○ Summary of change description ○ List of approved documents to be changed and estimated effort ○ List of other documents affected by the change and estimated effort ○ List of reviewers (approvers) and optional reviewers 	The new form was developed to address three CR types (A, B, and C). Various important fields are included to capture more detailed change information including change impacts and effort estimation. Examples of additional fields: <ul style="list-style-type: none"> ○ CR types, ○ Change description ○ Reasons for change (by type) ○ Lists of documents affected and other documents impacted ○ Effort estimation for each document affected ○ Lists of work products to be changed ○ Implementation priority and the implementer(s) to make the changes ○ Implementation Schedule
Change Classification	No classification	Initial change classification introduced, such as changes to: <ul style="list-style-type: none"> ○ Requirements (MKR, TCR, and ITS), ○ Functional Spec ○ Design Spec ○ Code ○ Documents, etc
Tools	Uses MS-word and Excel. No automatic tool is used to raise CR.	This new process is supported by the new tools (Change Synergy and DOORS).

5.4.3 Phase_6: Survey Findings

This section presents the findings from a survey that was conducted six months after the new Change Control Process introduced. Detailed survey results can be seen in Appendix K.

5.4.3.1 Change Participants Overview

The response rate of this survey was 56%, which indicates 27 out of 48 participants returned the completed questionnaires. The participants were asked to rate their understanding of the new Change Control Process. The survey data indicates that 63% of the participants have *good* to *excellent* understanding of the new CCP, while the rest of the participants (37%) have *Fair* or *Poor* understanding. It is understandable that software developers at GDS will take some time to become used to the new CCP.

5.4.3.2 Change process activities

Similarly to the previous survey, the change participants' opinion on several CR activities was also sought (Question 4 of the questionnaire, Appendix J). Overall, the survey findings indicate that the participants' opinions on various change process activities are not much different to the responses from the previous survey. Some of the important activities rated by the participants are illustrated in Figure 5.24.

The majority of the participants (63%) ranked *Change impact analysis* as a highly important activity during the change request process. As observed in this case study, the change participants (e.g. *Initiator*) had to collate and consolidate information about the change impacts from the affected stakeholders. Although this activity is important, often the initiators could not complete the change impacts information when they proposed a change.

Regarding the *Change request review meeting*, 78% of the participants ranked this activity as medium to highest level of importance. It seems that the participants had begun to understand the importance of this meeting to quickly resolve issues and

differences among the reviewers, approvers, and initiators. However, a formal change request review meeting has not been included in the new CCP.

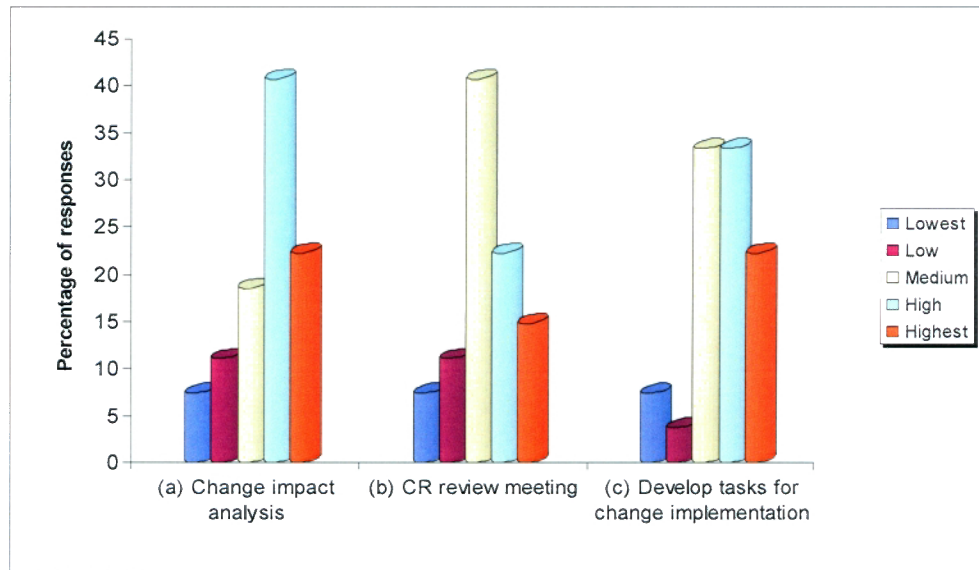


Figure 5.24 The importance level of some of the CCP activities

Develop tasks for change implementation is another change activity considered. 55% of the participants ranked this activity as high to highest level of importance, while 33% of the participants thought it was a medium level. Note that the new CCP is supported by an automatic web-based tool. Within this tool, the change implementation is scheduled based on the number of tasks generated by an approved change request. This survey finding reflects the importance of creating and distributing tasks after a change request is approved.

5.4.3.3 The Initiator's perspective

Of 27 participants, 16 (59%) of participants had previously submitted Change Request forms. Based on the *Initiators* perspective, this section presents the survey findings of their opinions on the new CCP.

It was expected that the implementation of the new CCP would not run smoothly in the first six months because of the *learning curve effect* or the new experience for GDS development team to understand a new process, CR form, and new tool. This is reflected in the survey findings (see Figure 5.25) that 44% of the initiators disagreed (38% disagreed, 6% strongly disagreed) on the *usability* of the new CR form.

However, the findings also reveal that 31% and 6% of the initiators agreed and strongly agreed on the fact that the new CR form is easy to use.

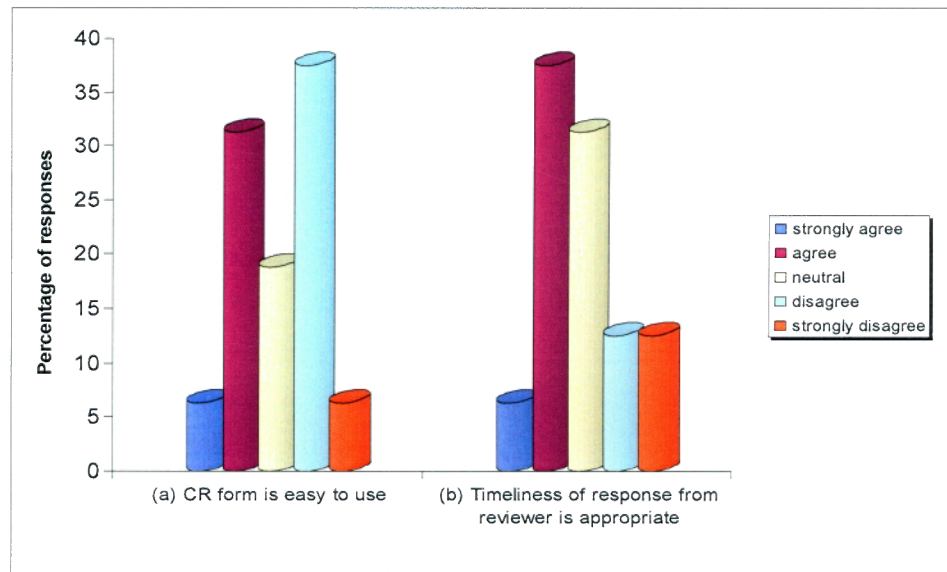


Figure 5.25 The Initiators' agreement on the CR form and reviewer's response

There is positive feedback toward the response time from reviewers. 44% of the initiators agreed (strongly agree (6%) and agree (38%)) that the *timeliness of responses from reviewers is generally appropriate*. Interestingly, 31% of the initiators were neutral. Based on the researcher's observations, the Change Synergy tool that supports the new CCP has significantly improved communication about changes among the initiators, reviewers, and other stakeholders who are affected by the change. The tool is an automatic web-based tool that enables the change participants to communicate with each other via email. Automatic change control is one of the strategies for better handling of requirements volatility (Sugden and Strens, 1996). In addition, the structure of new CCP also contributes to faster change review activity.

In this survey, the participants were also asked to provide their opinion on the new CR Types. These questions were intended to evaluate the inclusion of the CR types (Type A, B, and C) in the new CCP. They were asked whether they agree or disagree with the following statements:

1. The inclusion of CR types is useful for identifying and analysing change impacts.
2. The inclusion of CR types is useful for choosing appropriate reviewers.

The survey results as presented in a pie chart (Figure 5.26) show that 25% of the participants are neutral and less than 50% of the respondents (6.3% strongly agree, 37.5% agree) agreed on the usefulness of the CR types to assist them in identifying and analysing change impacts. This low response is understandable, since there many other factors that might be involved in the change impact analysis. The intention of introducing the CR types is to improve the initiators' understanding of the potential impact of each CR. This approach alone however, is insufficient and GDS should consider a change impact analysis procedure to assist the initiators or developers. Wiegers (Wiegers, 2003) points out the importance of change impact analysis in managing requirements change and he also provides a step-by-step procedure to perform the impact analysis.

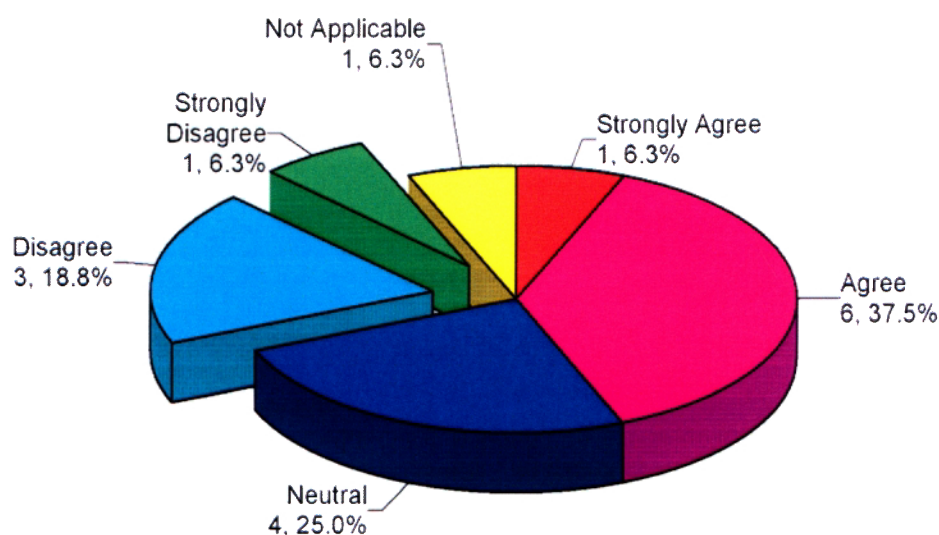


Figure 5.26 The Initiators' perspective on CR types and change impact analysis

The second new question of the questionnaire was intended to assess whether the CR types are useful in identifying the appropriate *Reviewers* to review the proposed CR. The participants' response is illustrated in Figure 5.27.

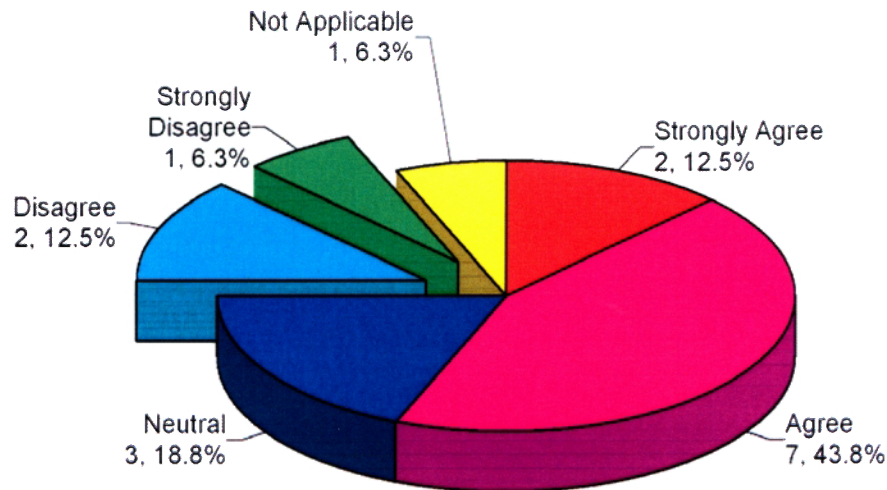


Figure 5.27 The Initiators' perspective on CR types and appropriate reviews

It is interesting to note that 56% of the initiators (12% strongly agreed, 44% agreed) think that the CR types in fact are useful in assisting them choosing the reviewers, while 25% of the participants are neutral. These findings indicate that the new CCP has rigorously distinguished the different types of CRs, which have different process flows and has thus led to different assessments by reviewers and approvers (see Figure 5.21 to Figure 5.23).

5.4.3.4 The Reviewer's (Approver's) perspectives

Of 27 participants, 20 (74%) had previously reviewed a proposed change request. Based on the *Reviewers'* perspectives, this section presents the survey findings of their opinions on the new CCP and CR form.

Feedback was collected from the participants on some aspects of the new CCP. Figure 5.28 illustrates three important issues during the CR review activity. The survey data indicated positive results from the reviewers on the rationale for change and information for the affected areas. It shows that about 45% to 55% of the reviewers agreed that the information given in the CR form for the (a) *rationale of a proposed change* and (c) *information for the affected areas* is generally sufficient.

This finding indicates the new CCP and CR form have addressed the weaknesses of the old change process and form.

Another important issue, which had been discussed earlier and raised by some change participants, is (b) *the adequacy of change information in the CR form if the proposed change will not be introduced*. The survey results indicate that 35% of the reviewers were neutral toward this issue, while 30% of the reviewers agreed that the information is adequate. Again, this issue was taken into account during the development of the new CCP and CR form. It is important information for the reviewers, approvers, and project managers to anticipate changes to the planned schedule or development effort.

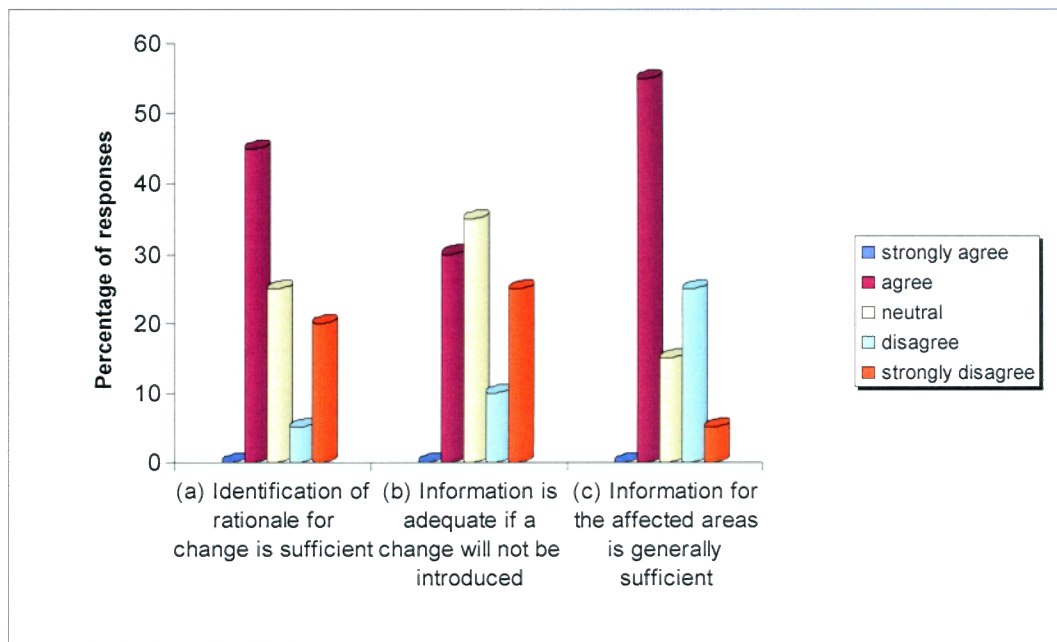


Figure 5.28 The Reviewers' agreement on the aspect of new CCP

Other change impact-related issues were also sought from the reviewers' perspectives. The field for impact analysis information in the new CR form was expanded and restructured so that the initiators could foresee the potential impacts of a proposed change on various software artefacts, and it helps the reviewers/approvers to effectively analyse the feasibility of a proposed change. Figure 5.29 shows the reviewers' opinion on the aspects of the CR form relating to the change impact information. More than 70% of the reviewers agreed that the impact analysis

information is generally helpful in assisting them to approve or reject the proposed CR. This is a good response toward the new structure of CR form.

It is interesting to note the reviewers' positive feedback on other aspects of the change impact analysis. As seen in Figure 5.29, 50% of the reviewers agreed or strongly agreed that the impact analysis information is also beneficial in helping them to estimate the work or effort required to implement the change. This issue has been widely discussed in the literature and approaches or procedures to conduct formal change impact analysis have been suggested by many researchers. For example, Wiegers (2003) provided a step-by-step procedure for requirements change impact analysis that can easily be adopted and followed by industry practitioners.

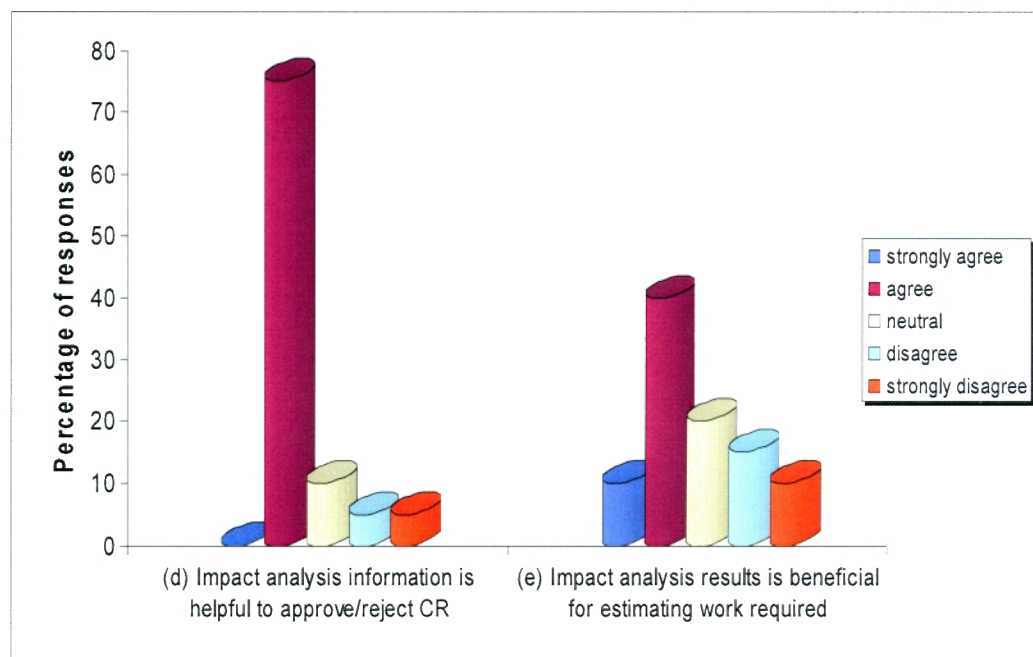


Figure 5.29 The Reviewers' agreement on the change impact information

5.4.3.5 The Implementer's perspective

Of 27 participants, 16 (59%) had previously implemented a proposed change request. Based on the *Implementers'* perspective, this section presents the survey findings of their opinions on the new CCP and CR form.

Among many issues identified for CR implementation, Figure 5.30 exemplifies two important issues from the survey: *change communication* and *written*

information for change implementation (a complete finding from the survey is presented in Appendix L). There seem to be positive response on these two issues. Half of the implementers (50%) agreed or strongly agreed that the communication among impacted parties is generally effective with the new automatic CCP and CR form, while the other 31% of the implementers were neutral.

More than 50% of the implementers also agreed that the written information stated in the approved CR form is sufficient. This means that with the new CR form the initiators or project manager/development team are encouraged to list the change implementation tasks and developers who are responsible for the tasks.

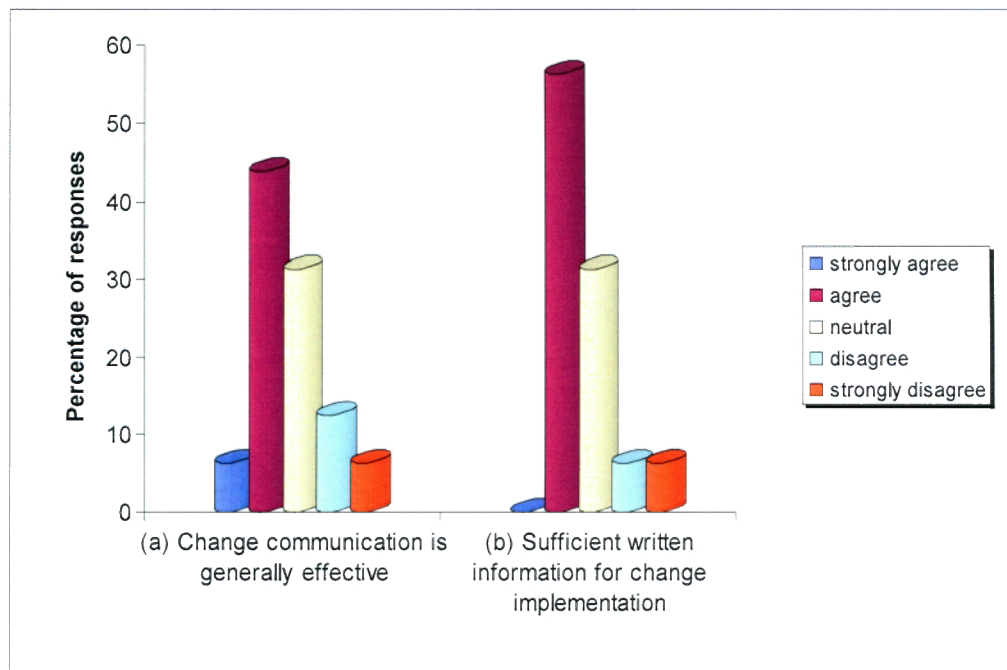


Figure 5.30 The Implementers' agreement on aspects of CR implementation

5.4.3.6 The Verifier's perspectives

Four participants (14%) had previously been involved in CR verification. Their responses on *the ease of tracking the approved and implemented changes made based on the information available in the CR form* is Fair (50%) and Good (50%). This is a very good response, which is caused by the rigorous documentation of change request information using the new CR form and an automatic supporting tool.

5.4.3.7 Overall view of the new CCP and its supporting tool

Since the new CCP and CR form were developed to address some limitations of the previous process and to improve the way GDS manages change during software development projects, the change participants' feedback on the overall view of the new process and its supporting tool were also sought.

Figure 5.31 shows that more than 50% of the participants agreed that the new CCP meets their needs for requesting and processing a change (the left chart). This means that the new CCP provides clear and well-defined change activities and change types that meet the change participants' needs. Regarding the tool, more than 50% of the participants also agreed that the tool is easy to use (the right chart).

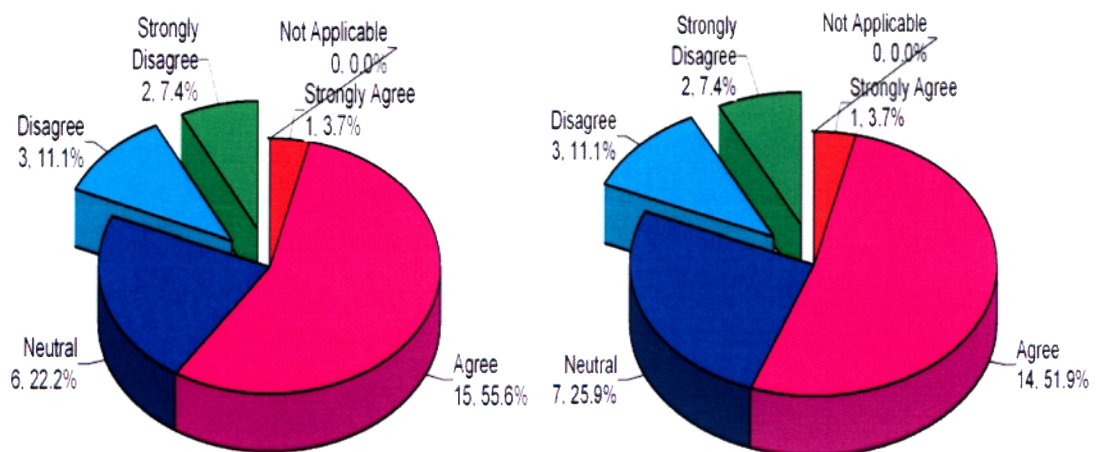


Figure 5.31 Overall view of the new CCP (left) and its supporting tool (right)

5.4.4 Phase_6: Summary of the new CCP evaluation

This phase has covered the activities associated with the evaluation of the new Change Control Process (CCP). This process document was reconstructed to address the limitations and problems identified both from the case study findings and from the previous survey results. A survey questionnaire was also used in this activity to gather the change participants' opinion on the usage of the new CCP and its supporting tools. Findings from the survey data are presented and discussed. There

are quite positive responses about the new CCP, CR form, and its tool. Some important issues related to the change process activities are also discussed.

5.5 Summary

This chapter has covered the empirical findings from the last three phases of the case study activities. The results of the change process assessment on both the existing and the new update version of the CCP and CR form have been discussed. The findings from the change request analysis on *Project Release_B* data have also been described.

The pre-survey results produced in *Phase_4* generally support the findings derived from the documents inspections and the observations. The existing change process model followed by GDS is described in detail and is presented in flow-chart form. The findings from the two software releases reveal the deficiencies of the CCP and CR form. These deficiencies were identified during the case study observations, document inspections, and from the pre-survey data. For the CR form, the deficiencies included:

- Lack of impact information and effort estimation
- Lack of change implementation details
- The CR form did not provide sufficient detail for project manager to plan and schedule the changes
- Insufficient change descriptions
- There was no clear indication about the change type, e.g. adding new requirements or functionality changed.

In addition, the deficiencies of the CCP document were as follows:

- It was a manual process (lack of tool support)
- There was no clear indication of who the targeted reviewers were, for what artefacts, from the CCP. This caused delays in reviewing and approving CRs
- Lack of coordination on the change implementation by all affected parties
- No traceability link existed between CRs and affected products

- No clear responsibility stated in the CCP for filling up each sections of the CR form

These limitations were significant opportunities for process improvement program at GDS. As a result, the new updated CCP and CR form were established and applied to GDS projects.

Phase_5 activities replicate *Phase_2* activities. The data collection and analysis techniques used in *Phase_5* are similar to those in *Phase_2*. Interestingly, the findings from *Project Release_B* on the change request analysis show slightly different results from the findings from *Project Release_A*. *Project Release_B* experienced substantial volatile requirements throughout the software development lifecycle.

Phase_6 activity was the evaluation of the implementation of the new CCP and its supporting tool. The survey results show positive responses from the participants toward the usage of the new CR form and the automatic change control process. Overall, the participants seemed mostly satisfied with the new process and tool.

Based on the overall analysis (Chapter 4 and Chapter 5), key findings are listed below and will be discussed in Chapter 6:

1. Measuring requirements volatility throughout the development lifecycle can help the organisation monitor the progress of the project.
2. Representing requirements volatility measures in graphical views provides informative insights to better understand the RV problems.
3. A classification of requirements changes is a useful mechanism for empirical analysis of change request data. It can be used to strategically analyse the change problems and identify risks. The classification should be integrated into a change management process.
4. A combination of classification attributes and graphical views is a strategic scheme to represent the characteristics of requirements volatility and to track the progress of change request process.
5. Understanding the issues behind the needs for requirements change and techniques to analyse the change led to the development of effective strategies for handling or controlling the adverse impacts of change.

Chapter 6: Analysis and Extension of Theory and Practice

6.1 Introduction

Following the analysis of findings from the two software project releases and change process assessments, this chapter provides a synthesis of the research findings on requirements volatility identified in the current practice. This chapter addresses the following research questions:

RQ4. What gaps exist between current theory and current practice?

The analysis includes characterising and representing the rate of requirements volatility, identifying effective approaches in handling RV problems, and understanding different aspects of RV that were identified in current theory and those reported in this case study.

This chapter begins with a discussion of the use of the classification scheme in analysing change request data. This is followed by a discussion of aspects of RV, to explain how they contribute to better understanding the underlying causes of requirements change and their consequences for software development projects. The attributes of requirement change, such as *change types*, *reason for change*, and *sources of change (change origin)*, as well as the impact of RV on development effort are the main aspects of requirements volatility discussed in this study. The strategies for handling RV proposed in the literature and the approaches used in practice (GDS) are also discussed.

6.2 Classification of Requirements Change

The primary aim of developing the requirement change classification scheme was to gain a deeper understanding of the nature of RV. The other usage of this classification is as a mechanism to monitor the change request processing time for

each type of requirements change and to examine the amount of effort that is required to implement different types of changes. The immediate benefit of this mechanism is that it is easy and simple to use within software development practices with relatively limited available resources.

The review of literature in Chapter 2 reported that a classification is one of the most common analysis techniques used to analyse the complex nature of requirements change. In practice, the classification is also used as the basis of software measurement (Fenton and Pfleeger, 1996). In this case study, a *classification of requirements change* is proven to be effective for analysing requirement change requests, particularly in identifying the causes and the costs of requirements changes during software project duration. The classification developed in this study also provides valuable information when it is used for analysing multiple dimensions of requirements change issues (*change types, reason for change, and sources of change*) as comprehensively presented in Chapter 4.

Previous studies have introduced various requirements change classifications which represent diverse contexts. For example, requirements change classifications in the context of *stable* and *volatile* requirements (Harker and Eason, 1993), in the socio-technical relationships (Kotonya and Sommerville, 2002), and in software maintenance (Stark et al., 1999). Although those classifications represent various contexts, the main objective is still the same, that is, to better understand the change problems being studied. Thus the classification provides the opportunity for the problems to be empirically examined and analysed.

This case study constructed the *classification of requirements change* according to three basic elements that can be easily obtained from the project change database (Change Request data). These elements are *Change Types* ("what" factor), *Reason Category* ("why" factor), and *Change Origin* ("where" or "who" factor). These elements form the core of change analysis techniques to assist in identifying multiple aspects of requirements volatility. As an effective technique, the *classification of requirements change* can be used to analyse each requirements change request by assessing its sources and its impacts on the software project as well as identifying the most volatile requirements.

Change request data from two software releases was used to develop our classification. Various attributes that emerged from this data represent the problems experienced by each release within its current development activities. For example, the attributes of the *Reason Category* and *Change Origin* varied across the two software project releases studied, which is possibly due to the characteristics of each project and problems encountered during the development of the projects. These attributes are valuable information to enrich the classification constructed in this study and they are also significant factors to improve our understanding on requirements changes.

6.2.1 The development of Requirements Change Classification

As described in Chapter 4, the classification's attributes were derived from both the change request data and the software practitioners' perspectives. While the former was mainly developed with respect to the researcher's skill and understanding, the latter confirmed and supported the researcher's classification. Overall, these classifications represent valuable information which was constructed from three sources through many iterations: *Project Release_A* (change request data), *Software Engineering Practitioners at GDS* (Card Sorting analysis), and *Project Release_B* (change request data). The process of developing the classifications is illustrated in Figure 6.1.

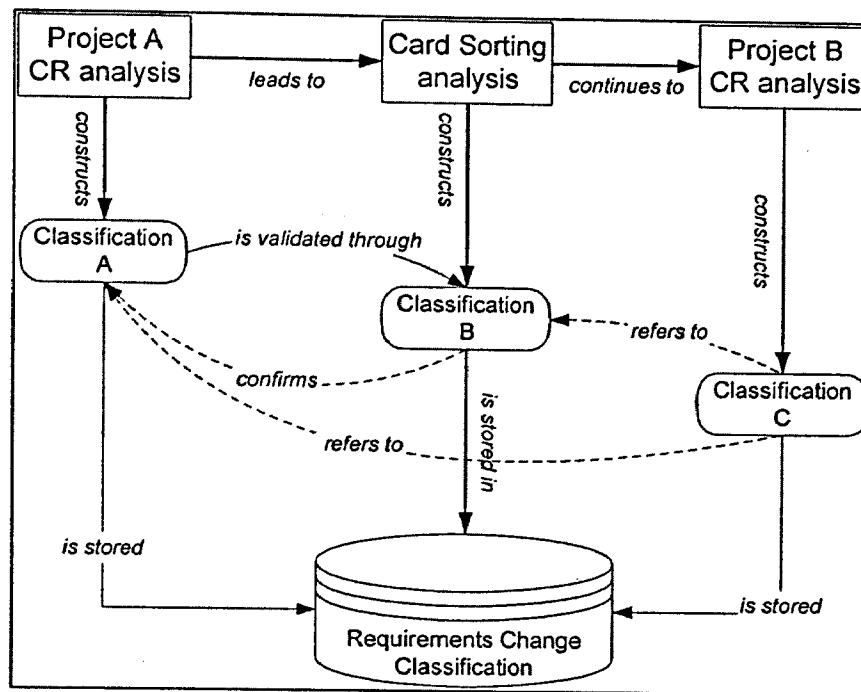


Figure 6.1. Requirements Change Classification Process

The results of categorising requirements change attributes from the three sources above have led to the collection of common change attributes that can be used by GDS in managing future changes. These classification attributes should be stored in a repository within the project database and be utilised during change request analysis. During interviews, the project management confirmed the value of the classification scheme and that it should be integrated into the change request activities, particularly during review and approval of change requests.

Having developed a requirements change classification based on the change request data collected from a manual-based change control process and change request (CR) form, difficulties were encountered during the classification process. For example, identifying '*reason for change*' was problematic; due to insufficient change information the researcher had to conduct extra data collection activities. In order to overcome this problem, the change reason information had to be elicited from each change request iteratively. When necessary, the initiator of the change request was asked to provide the missing information.

There are two perspectives involved in the process of categorising reasons for making changes. Beside the researcher's perspective, this research also takes into

account the perspectives of GDS's software engineers. Certainly, these two different perspectives have enriched the classification of requirements change. However, we needed to resolve these differences or to validate the initial classification (*Classification A*) by conducting the *card sorting* activity.

The results of this activity improved the initial classification attributes. The classifications derived from these two perspectives show few differences in the context of the reason categories, which were discussed in Chapter 4. Another reason to support the *card sorting* activity is that the classification developed here is the product of the case study research at GDS where it is likely to be used within GDS environment. Therefore, we believe it is necessary to take into account the perspectives of GDS's personnel in the classification process.

In future, GDS management will be able to classify requirements change requests using the methods developed in this case study. For example, applying the classification elements, such as *Change Types*, *Reasons for change*, *Change Origin*, is essential for the day to day change request management at GDS.

6.2.2 The Benefits of Classification

As presented in Chapter 4 and Chapter 5, various empirical and strategic analyses on requirements change problems show the usefulness and value of the classification. The classifications have provided the following benefits:

1. characterise the nature of requirements volatility;
2. understand the causes of requirements volatility;
3. assess the impact of changes, e.g. development effort required to implement the change;
4. provide information in relation to improvement opportunities on software processes, e.g. peer review process, requirements engineering process;
5. learn about previous changes (history of changes) and their consequences throughout the development lifecycle;
6. provide a means of controlling and sizing changes by mapping the impacts of each category.

6.2.3 Requirements Change Attributes

In order to understand the problems of requirements volatility, it is important to analyse each request for change, including characterising the nature of changes. This is an essential step that leads to the development of effective strategies to address the RV problems. In this section requirements change attributes are discussed and the discrepancies between the attributes identified in this case study and those reported in the literature are also presented.

6.2.3.1 Change Types

A review of the literature reveals various types of requirements changes. Many researchers have classified the change types into *additions of new requirements* (Add), *deletions of existing requirements* (Delete), *modifications of existing requirements* (Modify), or scope changes (Costello and Liu, 1995, Malaiya and Denton, 1999, Stark et al., 1999, Pfahl and Lebsanft, 2000). In the context of risk assessment, Luqi and Nogueira (2000) define the requirements change types as ‘*birth-rate*’ (the rate of new requirements incorporated), ‘*death-rate*’ (the rate of existing requirements dropped), and ‘*change-rate*’ (the rate of existing requirements changed). Although the researchers may use different terms to classify requirements change types, they all agree in the operational terms of requirements addition, deletion, and modification as described above. These change types are also commonly used as basic measures of requirements volatility (Costello and Liu, 1995, Malaiya and Denton, 1999) and as one of the risk factors for assessment model (Luqi and Nogueira, 2000), which will be discussed later in this chapter.

In this case study, the change types Add, Delete, and Modify were used as the metrics together with other change request metrics recorded in the project repositories and were used as additional information during project management reporting. However, the collection of these metrics is not commonly utilised in project management activities, particularly for *Project Release_B*. Based on the

researcher's observations, these metrics were rarely used for project tracking purposes. In addition, there was no formal measurement and analysis process established at GDS, therefore the metrics collected (in this case 'change request data') were not fully utilised to support decision making processes. It was basically left to the discretion of the project managers.

The *Change Types* attributes also provide quantitative and qualitative measures that can be used to understand the issues behind requirements volatility and their associated risks. In other words, the change types provide direct measures of the significance of changes by linking them to the change impacts. For example, the calculation of project risk based on a particular type of change (e.g. additions of new requirements) and its impact (e.g. development effort) that occurs at different phases of software development lifecycle (e.g. the design or testing phase) should provide strategic information for project managers in decision making. In further analysis, the risk of adding new requirements in the later phases was proven to be higher than in the early phases. Empirical evidence presented in the case study findings (see Chapter 4, Figure 4.13 and Chapter 5, Figure 5.14) demonstrates that the total cost of adding new requirements to the projects is high, more than 70% of the total cost of requirements changes.

Although adding new requirements in the later phases of the development lifecycle could not be avoided for various reasons, the findings suggest that GDS or other similar organisations should pay more attention to the late requirement changes and should be able to anticipate their adverse impacts. The findings also reveal that the earlier changes are incorporated within the lifecycle, the less effort is required and the risks associated with the changes can be minimised.

Thus, these types of requirements change (Add, Delete, and Modify) should be intensively used by organisations in their project tracking activities since they are important metrics, easy to collect and analyse, and also the immediate benefits to the project are apparent. Costello (1995) suggested that *Change Types* together with the reasons for change can be used as an indicator to monitor whether the degree of requirements changes is consistent with current development activities as well as to better control the requirements engineering process. Thus, the classification of

requirements change types is a simple analytical tool that provides significant benefits to the management of software projects.

Beside *Change Types*, another important element of the classification that is useful to characterise the nature of requirements volatility is reason for change (*Reason Category*), which is discussed in the next section.

6.2.3.2 Reason for Change

Understanding the causes of RV is an essential factor that should lead to more effective ways of managing its impacts. Defining the causes of RV is not an easy task; however, an accurate definition can be constructed through the analysis of change request data, as explained in this case study. This includes eliciting and categorising the *Change Types* (described in Section A) together with reasons for requirements change (*Reason Category*) and the sources of change (*Change Origin*).

In this section, the attributes of the *Reason Category* are discussed. These attributes provide valuable information that should be explored in more detail because they represent the current change problems experienced by the project. They also represent the purpose of requirements change, which is considered crucial information in analysing and reviewing requirements change requests. During the case study investigations, various reason attributes were identified from the two software project releases.

Previous researchers (Costello and Liu, 1995, Stark et al., 1999, Bohner, 2002) reported on various reasons that commonly triggered requirements changes in software development projects. There are some similarities and differences between the reason categories described in the literature and those identified in this case study (see Figure 6.2).

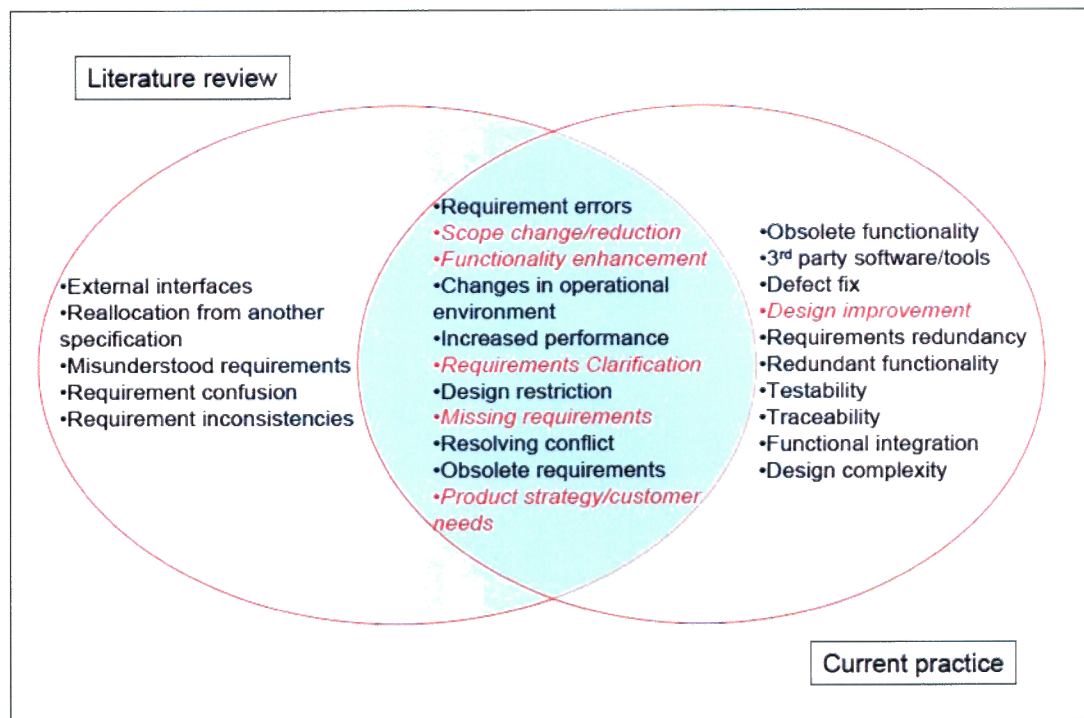


Figure 6.2. A summary of *Reason Category* attributes derived from the current research (literature) and the current practice (this case study)

The area in which the reason categories overlap represents the common categories identified in the current case study and in the literature. Among these common categories, the five most frequent reasons for change are *Functionality enhancement*, *Requirements Clarification*, *Scope change/reduction*, *Missing requirements*, and *Product strategy*. These findings suggest the common reasons for changing requirements that GDS and similar organisations should pay more attention to when analysing similar requirement change requests.

Bohner (2002) emphasised that most changes would probably occur at the same part of software and that using the change history is very helpful. For example, based on the reason for changes, a comparison of the current requirements changes with similar changes within the change request history can be used to assess work products affected by the change during change request process.

Since GDS develops product line software where the main baseline features remain stable with minor modifications from one release to the other, it is likely that most future changes will be similar to the past releases. With this assumption, understanding these common reasons or referring to the change history is essential if

the organisation is to learn from the past releases and use this knowledge to better manage future changes. The work presented in this thesis demonstrates the usefulness of the change history, which is important to learn about the past changes and to improve our understanding.

Considering aggregated change request data from both project releases as illustrated in Figure 4.15 and Figure 5.17, the results indicate that the top five significant reasons according to the total number of requirements changes were:

- *Functionality enhancement* (21.8%)
- *Requirements Clarification* (15.7%)
- *Scope change/reduction* (12.4%)
- *Missing requirements* (11.2%)
- *Product strategy* (11.2%)

Overall, these five categories accounted for more than 70% of the total requirements changes evaluated.

Although project management were not surprised by some of these categories, such as *Functionality enhancement* and *Product strategy*, they found these findings very useful. These two reasons categories were expected by GDS as part of the business goals to maintain and improve the quality of software application and to address current market needs. However, the other significant reasons, such as *Requirements Clarification*, *Scope change/reduction*, and *Missing requirements*, were unexpected and mostly occurred during *Project Release_B*. These changes represented the dynamic process of the project development activities. The case study investigations reveal that a substantial number of agreed requirements had to be deferred to the next release as a result of reducing scope or because they could not be achieved in the current release.

Beside those common categories, GDS and perhaps other similar organisations can also consider other categories (see Figure 6.2), such as *Design improvement*, *third-party software influence*, *Defect fix*, *Obsolete functionality*, or *Design complexity*. These categories are related to the characteristics of product line software, where its design or functionality needs to be improved in each new release

as well as its interdependencies with current available third-party supporting software/tools.

Overall, these findings can be used to identify improvement opportunities for GDS processes. Interpreting these findings would lead the GDS management to define better strategies, particularly in the development of future requirements, procedures to capture or anticipate requirements change, and effective requirements engineering process.

6.2.3.3 Requirements Change Origins

Another element of the classification is the source of requirements change from where the changes are originated. This element is essential for assessing or examining change requests because doing this will guide the way organisations approve changes and implement them.

As discussed in Chapter 2, the review of the literature revealed that requests for changing requirements during or after the development lifecycle may come from various sources, such as customers, end-users, action items as a result of the review process or walkthroughs, or feedback from prototypes (Curtis et al., 1988, Christel and Kang, 1992, Costello and Liu, 1995). These suggest that the requests are commonly raised by external sources (customers or end-users) and internal sources (project management, development team, or support team).

Various sources of requirements change were identified in this case study, similar to those described in the literature. The summary of these sources is presented in Figure 6.3. There are similarities and differences between the attributes of the *Change Origin* identified in the literature review and in this case study. The area in which the *Change Origin* categories overlap represents the common sources of requirements changes identified, both in the current study and literature. The *Change Origin* categories: *Customer (Marketing) request* and *Action item from review/walkthrough* are in fact the most frequent sources of requirements changes identified in this study. It is important to notice that although customers are most likely to request requirements changes, in fact the action item from formal peer review activities is also the main source to requirements changes.

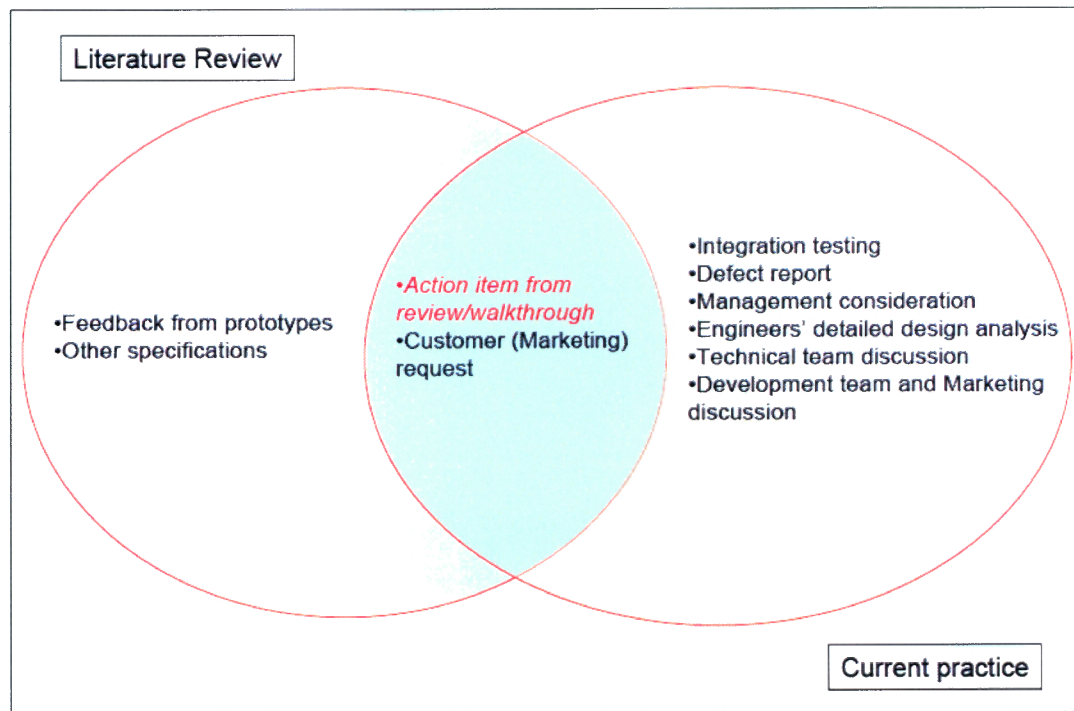


Figure 6.3. A summary of *Change Origins* derived from current research (literature) and current practice (this case study)

Aggregated change analysis from two releases reveals that the change origin *action items from peer reviews* accounted for more than 40% of change requests evaluated. This indicates that performing formal review/inspection on the work products can detect requirements defects early in the project lifecycle. This supports previous claims that formal requirements or software inspection is a cost-effective technique to discover requirements defects (Sommerville and Sawyer, 1997) and other defects present in software artefacts (Fagan, 1986, Boehm and Basili, 2001). In addition, the case study at GDS has demonstrated that implementing a formal peer review/inspection process can effectively discover defects.

Apart from the common requirements change origins (*Customer/Marketing request* and *Peer reviews*), this case study also identified other sources that could be considered exclusive to GDS organisation. They are as follows:

1. *Engineering Analysis* – which covers detailed design analysis;
2. *Discussions* between development team, marketing group, project management, and customer support;

3. *Management consideration* – which usually concerns product deliveries;
4. *Defect Report* – which records all defects reported by the customers or end-users after delivery;
5. *Product Integration Test* - defects identified during system integration test.

Understanding the various sources during change request activities can help GDS management or software engineers in general to anticipate the arrival of change requests, to better evaluate the importance of a change request and the time needed to process it. Since the change request process has multiple levels of change approval (as applied at GDS), the source types can be taken into account during the change approval, and also in prioritising CR implementation. A process improvement initiative has been undertaken at GDS in terms of assessing the levels of change request approval. The new change control process now has a mechanism to differentiate between the types of approval based on the sources and impacts of requirements change requests.

6.3 *Measuring Requirements Volatility*

As mentioned in Section 6.1.3 of this chapter, the types of change such as Add, Delete, and Modify are useful indicators for understanding the stability/volatility of requirements during the system development process. This section discusses requirements volatility measures and ways to represent them.

Measuring requirements volatility enables software managers to quantify the development and implementation of software requirements throughout the development lifecycle. It is part of requirements engineering activities that software managers must measure in order to control the project within its planned scope. This measure can also support the management of resources or effort, project risk, and quality.

Requirements volatility can be measured by counting the total number of requirement changes occurring over a period of time. It is calculated as a monthly rate of changes to requirements (Add, Delete, and Modify) throughout the project's life. As part of project tracking activities, organisations commonly record

requirements change data and store them in the project repositories. However, not all organisations see the benefits of using the change data to increase their understanding of requirements changes and to manage the change impacts.

This study clearly demonstrates the benefits of measuring requirements volatility. The metrics used can easily be applied in any organisation that has established a change control process and recorded the change data, such as total number of requirements additions, deletions, and modification, as well as total number of agreed requirements over a period of time. These basic RV metrics can then be represented as the cumulative number of requirements changes over time (Costello and Liu, 1995) or as percentage (Stark et al., 1999).

Collecting these metrics is not difficult. However, careful analysis of the CR documents is very important. In many cases during the analysis, there were inconsistencies between the quantities of changes stated in the CR documents and those recorded in the project repository. Thus, in order to overcome this problem, the researcher developed a spreadsheet to track and record the number of requirements changes every week. This spreadsheet is an elaboration of a simple *Change Request Register* (see Appendix F), which recorded the number of CR arrival, status of CR, dates (initiate and approved date), and the number of requirements added, deleted, and modified. In addition to this issue, the researcher also maintained weekly communication with the project managers and quality assurance lead. This activity, part of the case study method as described in Chapter 3, is important to confirm the validity of the data. Other relevant documents, such as specification documents, CR forms, and quality assurance reports, were also examined and cross-checked on a regular basis. With this approach the researcher maintained the integrity of the change data throughout the case study.

6.4 *Representing Requirements Volatility*

While measuring the rate of requirements volatility is important, graphical representation, such as using a bar chart or line chart, will provide additional information for analysing requirements volatility. Mapping requirements volatility over time with other change attributes, such as *Change types*, *Reason for change*, and

Sources of change, will further advance our understanding of requirements volatility problem and the identification of strategies to handle it.

Figure 4.10 (Chapter 4) and Figure 5.12 (Chapter 5) show the dynamic behaviour of requirements over a period of time (month) within the two software project releases. Each diagram shows the points where the peaks (high and low levels) of requirements volatility occurred. This form of graphical representation had not previously been used or presented in any of GDS's project reporting. Project management and some of the software engineers stated that they found these diagrams very useful in improving their understanding of requirements changes.

Although this study produced diagrams that illustrate requirements changes of the past projects, it is evident that measuring and representing requirements volatility provides strategic information for GDS management. This kind of analysis is particularly useful for organisations that develop product line software. Furthermore, this approach can help software managers improve their understanding of the dynamic behaviour of requirements and use the information to support management in making strategic decisions. For example, if changes occur in the early stages of the lifecycle, as shown in Figure 4.10, diagrams will help software managers to plan and allocate resources for implementing the changes. However, if changes still occur in the later stages of the lifecycle, as illustrated in Figure 5.12, they are likely to disrupt the planned schedule or budget, and software managers need to decide whether the changes should be implemented or deferred to the next release. The diagrams also suggest the points in the lifecycle where the changes are likely to happen, such as at the end of requirements analysis phase, during design phase or system integration testing. This kind of strategic information is essential, particularly for the organisations that embrace a process improvement program, where measurement and analysis is one of the essential support process areas for improvement within CMMI framework (Chrissis et al., 2003).

Change analysis and data mapping as presented in Chapter 4 can be followed by GDS management to collect metrics and information related to requirements changes. For example, mapping the rates of requirements volatility over time (or phases) combined with the requirements change attributes provides information on the high and low peaks of the RV level. It can lead to addressing the following

questions: “*Why did requirements still change at the later phases of development lifecycle?*” or “*Will the changes in later phases impact planned schedules?*” or “*Is the high rates of RV at the early phase a risk to the project?*” All these questions are related to the magnitude of requirements change and the case study findings have provided significant benefit for characterising and representing requirements volatility.

6.5 *The Cost of Requirements Volatility*

Our analysis reveals that change impact analysis and effort estimation are major problem areas. This section discusses the cost of requirements volatility which is based on (estimated) effort required to implement changes.

The result of the change effort analysis for *Project Release_A* data, illustrated in Figure 4.13 (Chapter 4), suggests that requirements changes for ‘*Functionality enhancement*’ purposes had the highest cost (286 hrs). This followed by other significant changes, ‘*Design improvement*’ (181 hrs), ‘*Defects fix*’ (170 hrs), ‘*Product Strategy*’ (165 hrs), and ‘*third party software/ tool*’ (69 hrs). These top five significant costs accounted for more than 90% of the total cost of requirements changes during the development of this release.

The analysis of change effort for *Project Release_B*, as demonstrated in Figure 5.15 (Chapter 5), indicates that 80% of the total cost of requirements change was consumed by changes related to ‘*Product Strategy*’ (342 hrs), ‘*Missing requirements*’ (242 hrs), ‘*Functional integration*’ (71.50 hrs), and ‘*Clarification*’ (63 hrs). These findings suggest that there were similarities and differences in the types of the change each project had experienced. For instance, ‘*Product Strategy*’-related changes are likely to occur for each release because they are associated with the market trends in product packaging, installation, or branding.

The simple analysis described above is useful for software developers and managers in assisting them to evaluate past and future changes. These case study results provide an insight into the cost of requirements volatility during the development lifecycle (especially for GDS). These findings can also be considered as a source of information to develop a better change implementation plan or to

prioritise the implementation of the changes according to their importance. In the context of the total cost, GDS should pay more attention to the changes that required more effort to implement them, such as '*Product Strategy*', '*Missing requirements*', '*Functionality enhancement*', and others described earlier.

Our analysis of the change effort data revealed that there are other attributes that should be considered when estimating effort. Pareto analysis on the change effort such as the one presented in Figure 4.13 (Chapter 4) and Figure 5.15 (Chapter 5) clearly indicates other factors might contribute to the estimated effort of change, such as *Change Types* (Add, Delete, or Modify). The analysis suggests that substantial effort was needed mostly to implement additional new requirements. This evidence highlights the consequences of adding new requirements at the later phases of development lifecycle.

Estimating change effort is not a simple task to perform, but it can be supported by change impact analysis. As part of effective requirement change management, understanding and examining the impacts of a proposed change can help software developers to assess the feasibility and reasonableness of the proposed change prior to change approval/rejection.

Having mentioned the values of change impact analysis and effort estimation process, the case study findings further revealed that the software developers or managers at GDS rarely performed this activity during the change process for the following reasons:

- the complexity of the software applications being developed;
- most developers who request a change are unable to identify other affected areas;
- incompleteness of the traceability data;
- no formal change impact analysis procedure was in place;
- pressure on project schedule and budget;
- motivation of the staff.

Thus, those reasons and the software developers' motivations to consolidate and conduct the change impact analysis during change request activities have contributed to their inability to provide realistic estimates of the change effort. As Wiegers (2003) and Bohner (2002) state, the ability to effectively perform change impact

analysis is dependent on the completeness of the traceability data. Without traceability information, the software developers may overlook a software component or work product that might be affected by adding, deleting, or modifying requirements. At GDS, the traceability link between software requirements and other software artefacts was clearly inadequate.

The change impact analysis procedure has immediate benefits for the developers in understanding the implication of approving a proposed change. The developers will be able to identify all affected areas and the finer-grain tasks needed to implement the changes, and then estimate effort for each identified task (Wiegers, 2003, Bohner, 2002). In this case study, we found that there was no formal change impact analysis established, followed, or integrated into GDS change control process. The change request initiator individually has to consolidate all change impact information from other members of development team before submitting the request. This situation often creates insufficient change request information on the potential affected areas caused by the change because the initiators are often reluctant to identify the detailed impacts.

In light of our case study findings, the change impact analysis should be well facilitated in the change request form. Improvement initiatives have been underway at GDS, such as updating the change request process document and change request form. As a result of this case study research, the form is now reconstructed to encourage the developers to perform impact analysis or consolidate information from various stakeholders, record all potential work products that the change might affect, and estimate the effort.

Although the effort data collected and presented in this case study is only based on the estimated effort data, the findings should be useful to the organisation and should encourage them to improve the way they record this data. The historical data, which can be focused on the individual change effort associated with the change types or reason categories, in the long term can become an effective method for estimating change effort.

6.6 *Strategies for Handling Requirements Volatility*

This section discusses the following research question: *RQ4b* What are the differences between the strategies outlined in the current theory and the strategies the organisation(s) apply in practice?

After an in-depth investigation into the requirements volatility issues, which covered its causes and consequences in details, this case study also identified the strategies used by GDS in managing requirements volatility. These include requirements change management practices, such as the change process established by GDS, the structure of the change request (CR) form, assigned roles and responsibilities, tracking the CR implementation, and other related issues. Recommended strategies proposed by previous researchers are also revisited and discussed in the next sections.

6.6.1 Recommended Strategies

There is no 'one size fits all' strategy to be used in handling requirements volatility. The proposed strategies can be viewed from various perspectives, such as perspectives from software development lifecycle models (Boehm and Papaccio, 1988), project management (Moynihan, 2000b, Ghosh P. P/, 2004), change management process (Harker and Eason, 1993), requirements engineering process, such as requirements modelling, requirements traceability, elicitation techniques (Sommerville and Sawyer, 1997, Van Buren and Cook, 1998, Hull et al., 2002) or organisational structures ((Boehm and Papaccio, 1988, Harker and Eason, 1993, Chudge and Fulton, 1996, Van Buren and Cook, 1998, Moynihan, 2000a, Hull et al., 2002, Ghosh P. P/, 2004). Other researchers have also suggested some strategies, such as reducing the rate of requirements volatility or making the changes less disruptive (Jones, 1996), improving quality of requirements specifications (Van Buren and Cook, 1998), understanding the procedures for capturing and changing requirements (Pfahl and Lebsanft, 2000), and also building good communication and trusting relationships with clients (Harker and Eason, 1993). Overall, these strategies

are useful; however, the adoption of these strategies is not a simple task for organisations because it will be dependent on the organisational complexity and policy.

Since the customers' needs and the business environment continually change, requirements volatility becomes a way of life in software development. In order to better handle these problems, once the nature of RV and its consequences are well understood, organisations can adopt the strategies proposed by various researchers or develop new ones that align with the organisations' needs. For example Van Buren and Cook (Van Buren and Cook, 1998) emphasised that effective elicitation techniques and tools could limit the number of missing requirements found during the design phase. This indicates the techniques and tools that software development team uses in eliciting, exploring, and analysing requirements early in the lifecycle are potentially useful to promote understanding of the requirements and explore design uncertainty, thus it led to the identification of change earlier.

6.6.2 Strategies in Current Practice

Besides identifying the nature of requirements volatility and its consequences, we also studied the way GDS controls and manages changes to requirements during the software development lifecycle. This organisation has implemented a process improvement program according to CMM/CMMI framework. The improvement initiatives were introduced to improve GDS software processes, such as the requirements engineering process, and project planning and tracking. Other processes, such as peer review process, requirements management process, and change control process are also in place.

The following discusses the relevant processes for requirements change requests and the characteristic of the two project releases that may have contributed to the way in which GDS handled the changes.

1. *A new software development methodology was adopted for the first time at GDS during the development of Project Release_B.*

When the new method was implemented, it changed the way GDS normally developed software product releases. The *waterfall model* was applied during *Project Release_A*. When *Project Release_B* started, an *iterative design and development methodology* was introduced. During this project's life, the development team members needed to adapt to the iterative approach, which is different from the previous method they used. This iterative approach allows them to develop design specification, code/unit test, and system integration through series of iterations, each iteration being more detailed. The iterative development approach is claimed to allow software development to evolve, manages requirements change better through allocated requirements and its iterations, and also provides earlier feedback from the correctness of the implementation (Jacobson, 1999, Leffingwell and Widrig, 2000). However, it should be supported by an effective change request process and procedure to handle rapidly changing requirements.

In our case study, the significance of this iterative approach to software development was not the main focus of the investigation. Thus, there is no conclusive evidence to demonstrate that the method worked well to address requirements volatility during this project. Furthermore, the iterative approach was used to improve the management of rapidly changing requirements; however, the case study findings do not support the claim that it made any significance difference as far as the rate of RV was concerned. There are three possible reasons why the rate of RV was not significantly different in *Project Release_B*:

- Lack of tool support (in facilitating and communicating small design changes)
 - Learning curve of staff involved (due to new methodology used and developing a new product)
 - Lack of rigorous process (the existing change process does not effectively process small rapid changes)
2. *Requirements engineering process improvement initiatives were introduced at the beginning of Project Release_A.*

At GDS, two main Requirements Engineering activities were conducted during the project lifecycle: requirements analysis and decomposition, and requirements testing (validation and verification).

- Requirements analysis and decomposition

During the requirements analysis activity, multiple levels of requirements were captured. Group sessions were held to analyse requirements for each feature or component. High-level requirements were analysed and decomposed into low-level requirements. The development team members, i.e. team leads from each feature/component, project manager, tester, SQA, product information team, were involved in the requirements capture sessions for low-level requirements. The customer representative was only involved during the high-level requirements analysis sessions.

The method used during the analysis of each requirement was to draw a context diagram to identify the interrelationships that may exist in the system design as a result of implementing the customer's requirements, and a MS Excel form (using a structured sentence template) was used to capture newly developed requirements. These requirements were structured according to: requirement description, requirement rationale, and test scenario. The development team found this technique was useful in developing new requirements and analysing their interrelationship with other existing systems [Damian, 2002].

- Requirements testing (validation and verification)

All captured requirements were subjected to peer-review once the requirements analysis phase was completed. Formal peer-review sessions (review meetings) with an inspection checklist were conducted for each feature or component. This peer-review process was also applied to the other work products, i.e. FS, DS, code, and test cases. The case study findings reveal that the peer-review process is effective in finding requirements defects and other defects in the early phases of development lifecycle.

3. *Requirements Management Tool*

An agreed set of requirements were baselined, documented in a requirements specification document, and recorded in the requirements repository using *Rational RequisitePro*. Various requirements attributes, such as requirement ID, functional types, rationale, test scenario, test cases ID, feature names, date, priority and iteration

number were also recorded. All requirements for the project releases were managed through *RequisitePro*. Other software used includes MS Power point and Microsoft Excel.

Recording the agreed set of requirements into the requirements database is a good practice in managing requirements. The tool supports the organisation in maintaining and facilitating changes to the requirements and tracing their link to the test cases.

4. *Requirements traceability*

Regarding *traceability* links, all low-level requirements were linked to the high-level requirements and marketing requirements (*backward traceability*). During the development of the *Project Release_A*, there was no *forward-from traceability* established between the requirements and design specification. However, the requirements were linked to their features or components on both project releases.

As GDS continues their process improvement program, the activities related to establishing traceability link between requirements and other specifications have also improved. This was shown by the traceability link created between requirements and functional specification and design specification during *Project Release_B*. It is acknowledged in the literature that requirement traceability is an important part of good requirements management (Gotel and Finkelstein, 1994, Wiegers, 2003). It plays an important role in managing requirements change, particularly when other requirements or specifications impacted.

However, no links were created between the changed requirements or change request and other impacted work products (i.e. FS, DS, or tcases). This link is useful in tracking the change implementation across multiple areas. As a result of this case study, recommendations have been made to GDS management for the improvement of the requirements traceability.

5. *The Change Control Process (CCP) used by GDS software developers was assessed*

Any change made to the requirements specification document has to be controlled via the Change Control Process. Requests for requirements changes were submitted to the relevant project manager who served as a moderator/approver during the change request review activity. The project manager was responsible for

facilitating, tracking, and monitoring the progress of a CR from initiation to completion. The CCP covered all changes to requirements, design specification, project plan, processes, procedures and work instructions. The results of the process assessment on *Project Release_A* and *B* by the researcher reveal that a CR form is used primarily as an operational tool to allow project manager and software developers track and communicate changes.

This process has been used by the GDS to control changes made to work products or project deliverables. However, it had not been updated in recent years (before *Project Release_B* started). Problems relating to the change process activities and change request (CR) form were identified during the case study investigation. For example, it took a long time to review/approve a CR, the CR description was insufficient to effectively analyse the need for a change, as effort data was missing for each affected area. These problems were confirmed by the pre-survey results. This led to an improvement opportunity for GDS, which resulted in the CCP as well as the CR form being restructured and updated. New tools (Change Synergy and DOORS) were then introduced to support the implementation of the new CCP, because the old tool could not handle the needs of the new process and the new software methodology that has been introduced.

Although GDS has implemented change management practices over the last few years, the case study findings reveal that their implementation of the CCP is not optimal. Findings from the pre-survey on the usage of the CCP and inspected documents indicate that the CR form and CR activities, such as CR initiation, implementation, and verification are still in need of improvement.

6.6.3 Strategies for handling RV

Effective strategies for handling requirements volatility problems should be developed appropriately for the organisational context. The problems that the organisation faces should be identified and examined in order to define the actions to be taken. Furthermore, available infrastructures should also be established to support strategies to address the problems.

As described earlier, a *classification of requirements change* and *change analysis approach* such as the one developed in this study (Chapter 4 and Chapter 5) should be used to improve our understanding of requirements volatility, its causes and consequences. At GDS in particular, the *classification of requirements change* should be used as a strategic tool to assist the project managers or development team members in analysing requirements volatility impacts and understanding the consequences of making changes. For example, requirements *change types* (Add, Delete, and Modify) can be used to assess the size of a change and its impact on the project's planned schedule. While the *Reason for change* category is useful for understanding the purpose of making a change and the problems that need to be resolved. Additionally, the *Source of change* category is important for assessing the impact of a change and taking improvement action. For instance, a request for change from a customer might have a ripple effect and if the change request is proposed in the middle of the development lifecycle (e.g. at code review time), a discussion between the customer and the development team should be conducted to effectively communicate the changes.

All findings presented in this thesis should be consolidated and formulated into effective strategies for handling requirements volatility at GDS in particular, and similar organisations in general. These strategies will assist them to improve their change related activities, such as change implementation plans, project risk assessment, and the estimation of change effort. As described in Chapter 4, the findings demonstrate the utilisation of change data. For example, if software developers captured impact analysis data as detailed in the CR forms, this data would enable the estimation of the effort needed to implement the changes or understand the cost of change more accurately. This is especially effective in developing product line software where the main baseline features remain stable from one release to the other and effort estimates of changes could be carried over from one release to the next with minor modifications.

GDS has established and followed a process for capturing and managing changes to requirements and other work products. This change control process is claimed to be the core of effectively managing change during software development project (Leffingwell and Widrig, 2000, Hull et al., 2002). The process assessment results,

however, reveal that there are some issues related to the change activities that need to be addressed by GDS management. At the end of the case investigation, a recommendation on revising the CCP and CR form was proposed to GDS. A new CCP and CR form have been developed and implemented. The following issues have now been addressed in the new CCP and CR form:

- detailed description of the proposed change, including CR types (A, B, and C);
- information of the change impacts and effort are mandatory;
- the inclusion of roles and responsibilities for each change participant;
- the time frames to review/approve a CR are explicitly articulated.

This improvement activity was derived from pre-survey findings on the change control process. For example, the change participants were concerned about the timeliness of responses from the reviewers (timeliness to approve CR and to implement/complete CR). One of the participants said *“CR takes too long to be approved. Management Leads/Technical Leads are sometimes unclear about the CR process and allow implementation to proceed even before the CR is approved ... the CR process needs to be streamlined – needs faster approval”*. The data also show that the average time to approve a CR was more than 55 days. This is a very long time to review or approve a CR. However, for GDS this is not surprising since the GDS policy recommends for many reviewers to be involved in the review stage.

There was no clear indication in the CCP regarding the appropriate reviewers for particular work products. This issue has now been addressed in the new version of CCP, which selects the reviewers of a CR based on change impacts. Thus different levels of reviewers are used for different types of CR. In addition, timeliness to review the proposed CR is limited to 10 working days, otherwise the CR is approved by default.

GDS software developers and managers also highlighted another issue related to the coordination of the change implementation and communications among the impacted areas. As a manually driven process, completing the documentation of the implemented changes very much depended on the implementer. It was found that some other impacted documents were not updated although the changes had been implemented. Coordination and communication during the requirements change

process is widely recognised to be a problem that needs to be resolved (Curtis et al., 1988, Harker and Eason, 1993). In order to address this problem, GDS management introduced a change request tool (TelelogicTM Change Synergy) which supports the new CCP and CR form. This should improve communications among the change participants because this tool automatically processes each CR that is submitted to the CR database. The tool also produces email notifications for all change participants who are involved in each change activity. The change participants can access the database anytime and monitor the completion of CR.

As comprehensively described and analysed in Chapter 4, the important findings reported in this case study confirm that:

- The organisation should deal with continuous requirements change, particularly in the later phases of development lifecycle.
- Classification is a strategic mechanism to analyse requirements change and understand its causes and consequences.
- Most changes to requirements were caused by changes in the business environment and technology.
- Various sources of change can be identified at certain phases of software development. Understanding these sources would lead to better strategies in anticipating future requirements change.
- Changes to requirements in later phases require substantial amount of effort or cost to implement, particularly for adding new requirements.
- The change control process is a significant process for requirements change management. Effective change control process should be structured according to the organisation context.
- Classification of requirements change should be part of the change process activities, particularly during CR review and analysis.
- The change control process should be supported by effective tools

6.7 Summary

In this chapter the key findings about requirements volatility have been discussed. The extent of requirements volatility and their characteristics were

empirically analysed and discussed. It was suggested that these two aspects of requirements volatility provide valuable information to increase understanding of the RV causes and impacts. It can also be used as basic measures to monitor the progress of the project.

Understanding factors that trigger the incidence of requirements change and response to the impacts caused by the change can be seen as effective strategies to manage requirements volatility. Another strategy is related to the implementation of a support process that helps organisations manage change requests in efficient way, i.e. the change control process. This process is central to the change management process that organisations should establish. Other processes, such as peer-review/inspection process, requirements capture and analysis process, should be established and followed by organisations. These processes and tools can facilitate better management of uncontrolled requirements changes.

Chapter 7: Conclusions and Future Work

7.1 Introduction

This thesis has addressed the important issue of requirements change management, and is concerned with the extent of requirements volatility during software development projects. The study has identified multiple dimensions of requirements volatility from a review of the literature and empirical findings from a longitudinal case study in an industrial setting. Following the presentation of analysis and synthesis of research data in previous chapters, this final chapter summarises the findings, presents conclusions, briefly describes limitations and outlines direction for future research on requirements volatility.

7.2 Research Objectives/Questions revisited

In this section the research questions that motivated the work are revisited and summarised.

RQ1. What are the characteristics of requirements volatility identified in the literature?

RQ1a. What are the sources of, reasons for, and types of changes that contribute to requirements volatility?

A review of the literature on requirements volatility issues was conducted to answer the above research questions. In the context of the characteristics of requirements volatility, two components were identified from the literature. These components were the *causes* and *impacts* of requirements volatility. The results

suggest that understanding the causes of RV and its consequences is essential in order to better manage the RV impacts.

RV problems are commonly caused by rapid changes in technology, customer needs, government regulations, market strategies, business rules, organisation complexity, and the increased knowledge of software developers about the software application being developed. These changes are perceived to trigger further changes to requirements during software development projects.

As reported in the literature, the causes of RV can be further characterised by requirements change attributes, such as the *types of*, *sources of*, and *reasons for requirements change*. Three *types of change*, *addition*, *deletion*, and *modification*, represent the higher levels of requirements change. The *sources of requirements change* or change origin vary across organisations and projects. The sources include internal sources (e.g. software developers, testers, and project managers), external sources (e.g. customers/end users, marketing group, or government regulations), action items from peer-review processes, and changes derived from other specifications. Another attribute is the *reason* for proposing a change. Various reasons were found in the literature including *requirements inconsistency*, *misunderstood requirements*, *external interface change*, *requirement error*, *obsolete requirements*, *scope change*, *clarifying requirement*, or *missing requirements*. Based on the research literature RV was then characterised by these change attributes.

RQ1b. What are the impacts of requirements volatility on software development projects?

It is widely reported in the literature that requirements volatility causes difficulties during software development projects. Frequent changes to requirements may have adverse impacts on many aspects of a project, such as project cost overruns, project schedule overruns, increased software defect density, increased rework (development effort), or adversely affecting testing readiness. The results suggest that organisations should consider the consequences of making changes to requirements and manage their adverse impacts on the project. Requirements will always change during the software development lifecycle. Understanding the causes of change will improve the management of requirements change impacts.

RQ2. What potential strategies are useful for managing requirements volatility?

Managing the impacts of RV is essential. Software engineering researchers have proposed various methods and strategies to handle the adverse impacts of RV. In general, the proposed strategies are intended to address three main issues:

- to identify the need for change as early as possible in the lifecycle,
- to minimising the occurrences of RV at later phases of the lifecycle, and
- to assist in the effective implementation of change.

The recommended strategies as outlined in the literature represent multi-dimensional perspectives. In the context of the software development lifecycle, researchers recommend evolutionary process models, such as iterative and incremental software development processes and prototyping. These approaches were developed to reduce risk and provide better solutions to user/customer needs. Software applications are developed incrementally in order to handle rapid changes to requirements or other software artefacts during the development lifecycle. For instance, when changes cannot be implemented during a particular iteration, then the changes will be deferred to a later iteration.

In the context of quality-oriented software process improvement, requirements change management and change control are part of the key process areas that organisations should focus on. Organisations should establish and implement a change control process to effectively manage the arrival of requirements change and to control the adverse impacts of RV.

The proposed strategies also suggest managing RV within software development phases. For instance, during the requirements analysis phase, identifying and capturing requirement changes early in the lifecycle is strongly recommended. Researchers introduced approaches, such as effective requirements elicitation techniques, conducting requirements reviews/inspections, establishing requirements traceability, or using early prototype approach to assist organisations identify the need for change and reduce the arrival of the late RV.

RQ3. How does an organisation manage requirements volatility?

RQ3a. What sources of, reasons for, types of changes and impacts of requirements volatility are most significant in practice?

A case study was carried out within a multi-site software organisation to address this research question. The case study findings were derived from two software project releases at GDS. The change request database was the main source of the case study data collection. Content analysis was used to analyse the change request data and to elicit requirements change attributes. Other sources of case study data include project documents (e.g. specification documents, processes and plans), observations, informal interviews, and card sorting exercises.

Based on the change request analysis of the two sets of software project data, the characteristics of RV were elicited. The three types of change described in the literature (*add*, *delete*, and *modify*) were also identified in this case study. The reasons for requirements change and the sources of change were also elicited. These reasons for and sources of change attributes of RV characteristics vary across the two software project releases reflecting their dynamic behaviour.

Reasons for requirements change presented in this thesis correspond to the needs for change, which are driven by factors such as *Product Strategy*, *Functional enhancement*, *Missing requirements*, *3rd party software/tool*, *Scope changed*, *Requirements clarification* and *Defects fix*. The results suggest that GDS or other similar organisations should pay more attention to these kinds of reasons in order to understand the importance of change and to develop strategies to handle its consequences and potential subsequent future changes.

As reported in the literature, RV is challenging and problematic during the software development lifecycle. In this case study, the immediate impact of RV on project development effort was investigated. Calculating the estimated effort required to implement the change has provided evidence that RV consumes substantial development effort. The findings suggest that when RV occurred in the later phases of the development lifecycle, more substantial effort (hours) was required to implement the later changes. Adding requirements at the later phases of the development lifecycle put the project at risk to complete on schedule.

RQ3b. What strategies does the organisation use to manage requirements volatility in practice?

Based on the observations of the software development projects and the inspections of the project documents, the strategies used by the development teams at GDS and their limitations are identified.

The main process to facilitate the incorporation of change during the software development lifecycle is the Change Control Process (CCP). This process was established and followed by the GDS development teams during the software development lifecycle. Changes made to the requirements specification document or other specification documents that may affect the requirements have to be processed and approved according to the CCP. This suggests that the CCP is an effective mechanism to facilitate the proposed change requests and to control their impacts on project documents and activities by prioritising, scheduling and deferring change requests.

GDS has established and implemented other processes that may help them to identify changes earlier, such as requirements capture and analysis processes and peer review processes. GDS has also pursued requirements engineering process improvement initiatives. Multidisciplinary teams, such as Marketing group (customer representative), GDS management, developers, testers, and product information team, were involved in the requirements capture and analysis sessions. At the end of the requirements analysis phase, the captured requirements were documented, reviewed, and baselined (software requirements specification document). Traceability link between the high-level requirements, the lower-level requirements, and test scenarios/cases were also established.

A review/inspection process was conducted at the end of each development phase. The findings suggest that this process was effective in finding defects earlier, particularly during the requirements specification reviews and the design specification reviews.

RQ3c. What are the current limitations of these strategies?

The limitations of the strategies employed by GDS were mainly in the utilisation of the change request data and CCP. The case study analysis points out that the CCP

did not fully represent the development team's current activities in facilitating the changes. In addition, the form used to propose or request a change was incomplete and lacked essential information (e.g. details of impacted areas, reason for change), which are important for change impact analysis and to communicate changes among development teams. These conditions contribute to the developers' ability to process proposed change requests effectively. Furthermore, all changes to requirements should be monitored and tracked continuously. Other limitations included the CCP being a paper-based process (no automated tool), and no traceability link between requirements and other downstream specification documents was in place.

RQ4. What gaps exist between current theory and current practice?

RQ4a. Are there any differences between the sources of, reasons for, types of change, and impacts of requirements volatility described in current theory and those identified in current practice?

This research questions was answered by analysing the RV related issues as outlined in the literature and the issues identified in practice at GDS. There were both differences and similarities in the characteristics of requirements volatility. For example, the same *Change types* (add, delete, and modify) were used in the literature and in current practice.

There were similarities in the *Sources of change*, for example, *changing customer needs* and *action items from peer reviews* were common sources identified from the literature and the current practice. There were differences in the reasons for change between the two fields. For instance, '*Design improvement*' and '*the influence of third-party software/tool*' were significant reasons for change identified in the current practice but were not identified through the literature. This difference could be driven by the characteristics of GDS's projects, which are aimed to develop software applications in a series of releases, maintaining and enhancing existing functionality to meet the customer needs.

In the context of RV impacts, the case study results confirm one of the impacts as described in the literature, that on development effort. Overall, RV during software development project consumes unexpected substantial development effort (hours) to implement changes. However, the assessment of an individual change suggests that a

lower rate of RV does not necessarily imply a lower project cost (hours). As the evidence presented in this thesis shows, a low RV rate in the later phases of the development lifecycle does not imply that the cost to implement the later changes is less. Thus, organisations should anticipate the later changes and minimise their adverse impacts on the project by restricting the project scope or keeping an effective CCP in place to better schedule and prioritise the change implementation.

RQ4b. What are the differences between the strategies outlined in the current theory and the strategies that organisations apply in practice?

Various strategies have been recommended from the literature to assist organisations in understanding the need for effective requirements change management and minimise the impacts of RV. However, the implementation of the strategies in practice is subjected to various factors:

- the organisations' policy and program to adopt a particular type of software development process (waterfall or iterative development process);
- the need to improve organisations' requirements engineering processes; and
- the use of automated tools to support the change management process.

There were differences between the strategies outlined in the literature and the strategies GDS applies in practice. For example, in order to effectively facilitate the incorporation of change, *performing impact analysis* and *the utilisation of the CCP for detailed change analysis, change prioritisation and scheduling* were outlined in the literature but were not implemented in practice (GDS). The results suggest that the approaches above are essential in assisting the development teams to better understand the changes and its consequences, however it is difficult to implement them if the organisation does not have processes and procedures in place and adequate tools to support them.

RQ5. If necessary, can current theory be extended to accommodate the imperative needs of current practice?

The findings of this research highlight the imperative needs of current practice (GDS), such as approaches to:

1. analyse the nature and impacts of changes to requirements during the software development lifecycle;
2. improve the developers' understanding of requirements volatility and its multiple aspects, i.e. causes, impacts, and strategies;
3. measure and represent the characteristics of RV;
4. estimate the cost of RV (change effort in terms of hours).

As discussed in Chapters 4, 5, and 6, this research points out the need to extend the current theory by proposing the approaches to analyse the nature of RV and its cost as mentioned above. Further implications for RE research are discussed in section 7.3.

7.3 Thesis Contributions Revisited

This thesis has made significant contributions to the field of requirements engineering and software engineering by demonstrating the strategic usage of change request data to understand the nature and magnitude of requirements volatility, to measure and represent requirements volatility over time and to highlight important results within a software development organisation.

In this section, the contribution of this research made up of several key elements are summarised as follows.

- An insight into multiple-aspects of requirements volatility was developed from various research disciplines. In Chapter 2, these aspects are discussed, paying particular attention to the terms used, the causes of and adverse impacts of requirements volatility, and the potential strategies that can be adopted by software development organisations. This review of the literature provides an analytic synthesis of a currently lacking comprehensive concept of requirements volatility phenomena.
- A case study research process involving a longitudinal case study in an industrial setting was designed and described in Chapter 3. This case study research provides an insight into methods for conducting the long-term case study research in software development practice.

- A requirements change classification was constructed from the change request databases for two software projects. In Chapters 4 and 5, the development of the requirements change classification is described and the actual results are presented. This classification can be used to identify and trace the problems, reasons for, sources of, and impacts of requirements changes. The requirements change classification also plays a pivotal role in the requirements change management, particularly in analysing the nature and magnitude of change and the cost of individual change.
- The card sorting technique was used to classify requirements change attributes from software practitioners' perspectives. The procedure for conducting the card sorting exercises is described and presented in Chapter 4. The results of these exercises were used to validate the preliminary classification developed earlier.
- A change analysis approach was developed to empirically analyse change request data and to demonstrate the benefits of documenting and collecting requirements change information. The approach was also used to develop requirements change classification.
- Graphical representations of requirements volatility such as those illustrated in Chapters 4, 5 and 6 are useful for representing the characteristics of RV. The results from data analysis demonstrate each aspect of the RV characteristics throughout the software development lifecycle including the total (estimated) cost of RV.

7.4 Implications for Research

The findings and contributions of this research have several implications for theory on requirements volatility related issues, particularly to extend the current theory for managing requirements volatility during software development project.

7.4.1 Requirements Change Classification

Classification, in general, is acknowledged as a basis for software measurement and as a strategic mechanism to empirically analyse changes throughout the software

development lifecycle. Using actual data from software projects, this thesis has presented a comprehensive technique for constructing requirements change classifications. The two approaches described in this thesis, the *change analysis approach* and the *card sorting technique*, represent repeatable methods that are described in sufficient detail in Chapter 4 and 5. These approaches could be used by other researchers to further understand RV issues, identify causes or reasons for requirements changes.

7.4.2 Measuring and Representing Requirements Volatility

This research confirms that the three types of requirements change, *addition*, *deletion*, and *modification*, are essential attributes to measure RV. Furthermore, this study has presented the extent of RV based on these change types from data for two software projects, discussed in Chapters 4 and 5. Representing the rate of RV throughout the development lifecycle in a graphical view is proven to be effective in increasing our understanding of RV issues and highlighting its causes. In addition, the extent of RV was elaborated in more detail, such as the mapping between requirements *Change Types* and *Reason Category* over time (development phases). This approach is used to 'drill down' to the causes of requirements change at a particular point in the lifecycle. The depth of analysis carried out and presented in this thesis clearly fills a significant gap in the current theory on RV.

7.4.3 Requirements Volatility and Change Effort

There is a lack of empirical evidence reported on the cost of RV or total change effort (hours) to implement changes to requirements during the software development process. This research has presented the utilisation of the requirements change classification to assist in identifying an individual (estimated) effort (hour), which is assigned to the elements of the classification, i.e. change types, reason for change, or sources of change. As discussed in Chapters 4 and 5, the mapping of RV rates and total estimated effort over time provides an indicative cost of RV. The

research findings suggest that changes taking place later in the lifecycle, require more effort to implement. However, a lower rate of RV does not necessarily imply a lower cost. This important finding clearly extends the current theory on the cost of RV during the development lifecycle.

7.5 *Implications for Practice*

The findings presented in this thesis have very significant implications for organisations that develop and maintain software. This research identifies key attributes that the organisations should pay more attention to such as approaches and specific elements in analysing, measuring, and representing requirements volatility issues.

7.5.1 Change Analysis Approach

The approaches presented in this thesis are useful in assisting software developers/managers or software analysts to understand the underlying causes of requirements change. They have been described and discussed in Chapters 4 and 5, and a simplified version of the approach is illustrated in Figure 7.1. The change analysis approach is repeatable and can be adopted easily by organisations to analyse change requests and generate valuable information to support strategic decisions.

The approach comprises three main stages: *Data Collection* (Stage 1), *Change Request Analysis* (Stage 2), and *Change Characterisation* (Stage 3). In order to perform an effective change request analysis (Stage 2), three main sources of data, i.e. Change Request forms, Release Documents, and the results of Interviews/discussions with project managers/software developers, are essential. The output of the change request analysis are twofold: the development of a requirements change classification (three elements: *Change Types*, *Reasons Category*, and *Change Origin*) and the identification of the causes of RV.

The fact that we have applied this approach successfully at GDS is a clear testimony on the effectiveness and usefulness of this approach in practice. We can improve our understanding on the underlying causes of change and its consequences.

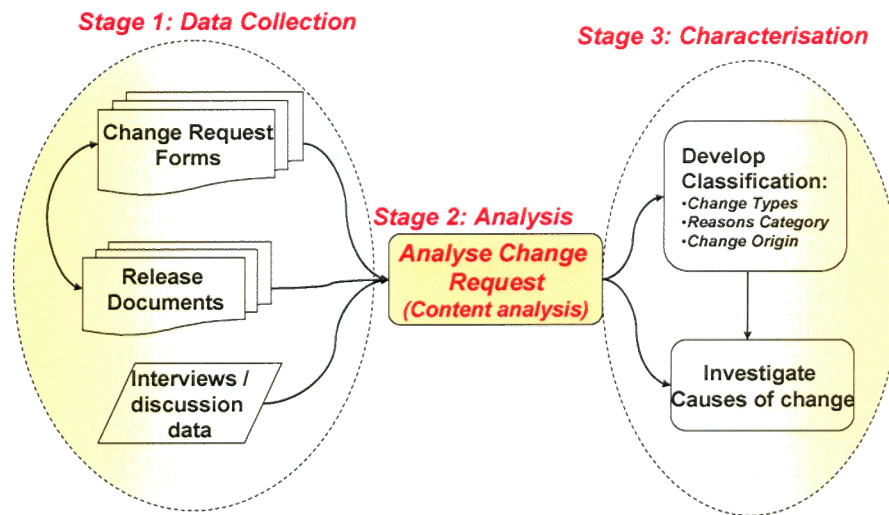


Figure 7. 1. Change Analysis Approach

7.5.2 Requirements Change Classification

A Requirements Change Classification is very useful in characterising the nature of requirements volatility during the software development lifecycle. This classification is described in detail and could be used by other software development organisations in their own environment to analyse changes and to gain a deeper understanding of the requirements volatility problems. It is easy to implement, simple to use, and has low cost to perform in the organisational setting.

The elements of the classification presented in this thesis (*Change Types*, *Reason Category*, and *Sources of Change*), are essential properties of change requests. If the change requests contain complete information about those properties, the classification will lead to an effective empirical analysis of RV issues. This analysis will enable software developers to identify the relationships between the elements and change requests. As demonstrated in this thesis, software development organisations should develop a requirements change classification using their project data and apply it within their organisation environment. The classification itself

evolves over time and should be refined from release to release (e.g. for product line software type). The requirements change classification should be stored in the project repository and managed as prescribed by the CMMI framework. In the context of the change process, the classification is useful to effectively track proposed change requests, monitor its potential impacts, and as a central point to prioritise and plan the change request implementation.

7.5.3 Measuring and Representing Requirements Volatility

Measuring RV is not difficult to perform when complete change request data is recorded and maintained. This research has calculated and displayed the extent of RV in different phases of software development lifecycle. This visualisation includes important strategic information for project managers, such as the rates of RV throughout the lifecycle, the change types and their associated reasons for change, the elapsed time to process requirements changes and total estimated effort needed to implement different types of requirements change. GDS engineers and other software development organisations will benefit from measuring, representing, and tracking requirements volatility. It will lead to better understanding of change and better scheduling of its implementation.

7.5.4 Requirements Volatility and Change Effort

This case study has analysed change effort using the classification developed and described in Chapters 4 and 5. Software developers and project managers should record estimated effort and actual effort to implement requirements change. In this research, the estimated effort data was used to calculate the cost of RV. As demonstrated in Chapter 4, recording estimated effort data according to the *Change Types* or *Reason Categories* provides analytical information for the effort estimation. The goal is to be able to address questions such as “*how much effort is needed to implement each requirements change type?*” GDS management will gain benefits from this historical data. However, the current information recorded and maintained

at GDS needs to be improved with the inclusion of more data on the change effort. Overall, recording change information (including in hours) is considered as good RE practice that can be used for future estimation purposes.

Software development organisations in general, and GDS in particular, should pay more attention to the types of requirements change that require substantial effort (hours). A combination of data from two releases suggests the top five significant costs are for the following reasons:

- Product Strategy (508 hrs),
- Functional enhancement (294 hrs),
- Missing requirements (243 hrs),
- Third-party software/tool (102 hrs), and
- Design improvement (181 hrs) and Defects fix (170 hrs).

This information is useful for software developers and managers in assisting them to evaluate past and future change implementation plans, change prioritisation or change analysis.

7.5.5 Change Process and Change Request Form

The Change Control Process (CCP) and its associated CR form are important factors in managing changes and communicating them among the development team, stakeholders, and customers. This research has pointed out that CCP and the CR form should be frequently updated to align with the organisations' business practices. Chapters 4, 5, and 6 discussed the limitations of the existing CCP/CR form and the importance of updating the CCP/CR form to address those limitations in order to better manage change requests. The CCP should distinguish the types of stakeholders involved in processing change requests, and the organisation should take into account the impacts of change in defining the levels of change approval. This study has also demonstrated the strategic use of the change information recorded in the CR form. This suggests that documenting change information, such as the type of change made, the reason for change, the sources of change, the impacts of change on various

work products, and estimated effort for each impact, is important and useful for collecting and storing historical data that can be used in future.

7.6 Strategies for Managing RV

The results presented in this thesis highlight the approaches used (by the GDS organisation in particular), to handle requirements volatility. Although there are many strategies proposed in the literature, the implementation of strategies in practice is subjected to many factors, such as the organisation policy and procedures. As observed and analysed in this case study, the findings indicate that managing requirements volatility during software development projects is mainly reliant on the implementation of a change control process and project managers' initiatives. Other processes, such as requirements capture and analysis (part of the requirements management process), and peer review processes, are also established and followed. These processes help the development team to identify defects and have the potential to identify changes earlier.

Combining research findings generated from the literature review and this case study, the following strategies are suggested for organisations to focus on.

7.6.1 Reactive Strategy

This strategy is aimed at handling changes made to requirements and to assist the effective integration of requirements change during software development lifecycle. The focus is on how to better manage the arrival of unexpected and expected changes to requirements, assist in improving understanding of changes, and provide mechanisms to communicate changes.

The importance of handling the changes that actually occur is widely discussed in the literature. Approaches within this strategy should cover such activities as requirements change evaluation, change impact analysis, tracking and monitoring change requests, documenting requirements change, and effectively communicating the changes.

The work presented in this thesis is mainly focused on approaches related to this strategy. The requirements change classification was developed to characterise requirements volatility based on the three elements described earlier. A list of attributes for each element was produced, particularly *Reasons for change* and *Sources of change*. This list provides valuable information for software developers and managers to improve their understanding on the requirements volatility issues. Furthermore, it enables them to identify volatile requirements associated with particular parts of software component and specific stages of the software development lifecycle.

Measuring and representing the level of requirements volatility is one of the effective techniques presented in this thesis. The *Change Types* are initially used to measure and represent the rate of RV throughout the software development lifecycle. Within these *Change Types* the change information contains the *Reasons for change* and the *Sources of change*. Thus this approach enables software developers/managers to identify which sources of change or which reasons for change need more attention.

The next use of the classification is in the context of collecting metrics on change effort (hours). This study demonstrates the technique used in recording the estimated change effort associated with the attributes of *Change Types*, *Reasons for change*, and *Sources of change*. This technique is part of the requirements change impact analysis that is widely recommended in the literature. In the context of the cost of requirements volatility, this technique can be used to evaluate the changes that require more effort and have a major impact on the project. In the long-term, this technique is also useful to estimate or predict effort for future changes.

This study also highlights the necessity of an automated change control process. The two surveys we conducted revealed that the automated change control process helps the project teams to better track the progress of CR and to communicate CR effectively among the team members or change participants.

The implementation of the techniques related to this *Reactive Strategy* is dependent on the organisations' policy to manage requirements volatility, which should continuously monitor the incidents related to requirements changes, such as

additions of new requirements, deletions and modifications of existing requirements, assessment of change effort, and re-scheduling project caused by these changes.

7.6.2 Proactive Strategy

This strategy is aimed at minimising the occurrences of requirements change, which covers techniques on how to identify requirements change early in the lifecycle and to reduce changes. The techniques include the adoption of effective requirements elicitation techniques, requirements analysis and modelling techniques, requirements management processes, and peer review processes.

As described in this thesis, the GDS organisation employed techniques for managing requirements volatility. Some of the processes followed were the requirements capture and analysis process (requirements development), requirements management process, peer review processes, and the change control process. These processes are not directly aimed at preventing the occurrences of requirements change, but they are part of GDS's process improvement program.

Requirements engineering researchers suggest that the *good quality of initial software requirements specification* (SRS) (which represents the agreement between customers/users, developers, and other stakeholders on a set of requirements) contributes to minimising rework at later phases. Throughout a software project's life, the SRS should be *baselined* and monitored for any changes that occur. GDS software developers had a process that guides them in capturing and analysing candidate requirements from customers, developers, and other stakeholders. This technique allows them to enhance their understanding on the current products, the sources of change, or software components that are likely to change. The SRS is typically baselined a couple of times during software project lifecycle.

Requirements management is an ongoing activity, where the agreed set of requirements is documented (recorded) and the status of requirements is monitored. Changes to requirements should also be recorded and tracked throughout the development lifecycle. The use of an automated tool is useful to support the requirements management activity.

The *Change Control Process* is a support process to control changes to software project documents or deliverables. It is aimed at helping the development team or project member to handle changes, to initiate, review, approve, and implement changes to requirement document and other specification documents. The usefulness of a new change control process employed by GDS as discussed in this thesis is clear evidence of the need to manage change.

Requirement traceability is a technique to establish the link between requirements and their sources, and the links between requirements and other downstream artefacts. This technique is also useful to explore and identify volatile requirements, design uncertainty, or future changes.

The *peer review* process is known to be a useful mechanism to identify defects. In the context of requirements volatility, peer review processes also contribute to the detection of early changes to requirements. As reported in this thesis, many requirements change requests resulted from peer reviews, particularly during the early phases of the lifecycle.

Overall, while the main focus of this thesis has been on the approaches within the *Reactive Strategy*, the findings can in fact also serve as inputs for *Proactive Strategies*. For example, this thesis highlights some important issues, such as:

(*Reactive strategy*) - The establishment of a traceability link between Change Request number and requirements document, and between requirements and downstream specification documents (design, code, or testing document) will improve the effectiveness of change control process and change impact analysis – *this is an input for improvement opportunity on requirements traceability technique (Proactive Strategy)*.

(*Reactive strategy*) – Understanding the sources of requirements change, e.g. changes that originated from Marketing Group (customer), developers, or bug reports, provides valuable information that leads to actions such as an improvement in the process of capturing customer requirements or identifying defects early in the lifecycle – *this is an input to adopt better requirements elicitation techniques (Proactive Strategy)*

7.7 Challenges, Lessons Learned and Future Work

In this section, the challenges faced by the researcher are highlighted and reflective remarks on the lessons learned are stated. This section then concludes with potential research for future.

7.7.1 Challenges

7.7.1.1 Change Request Form

Understanding the information provided with each of the proposed change requests for requirements is very important. Although GDS had established and implemented change management practices over the last few years, the change request form, which is the sole tool to raise a change to the project deliverables, had little information about the reason for a proposed change and the origin of change. The information was inadequate for analysing the importance or the reasonableness of the change to be made. This kind of information is necessary in analysing problems effectively.

7.7.1.2 Impact Analysis and Effort Estimate

The change information, such as change impacts or change effort estimation, was insufficient in most of the change request forms submitted to the projects. This information is also difficult to consolidate. There was no formal impact analysis performed and the developer (change initiator) was frequently unable to predict potential impacts of a change on some areas. This is possibly due to the complexity of software applications being developed.

Thus, collecting requirements change effort data in this situation is challenging. In order to resolve this problem, informal interviews with the change initiator and project manager were conducted to fill in the gaps in the form.

7.7.1.3 Traceability between Requirements and other Software Artefacts

Traceability is an essential technique to support requirements change management, particularly during change impact analysis. The potential impacts of requirements change on some areas can be assessed completely when requirements are traceable to the other software artefacts. In this case study we found that some changes were made to requirements without addressing the potential impacts on other areas. This is because no traceability link was in place to support the impact analysis.

Additionally, the traceability links between requirement change requests and the requirements specification document were not recorded. We found the challenge was significant during the verification of change implementation. Although this organisation used a requirements management tool for recording all requirements, change requests information was rarely captured in the requirements database.

7.7.2 Lessons Learned

7.7.2.1 RV Analysis

The data analysis on the two software project releases helped us to better understand the characteristics of requirements volatility and its costs. The rate of requirements change tends to decrease toward the end of the development lifecycle. However, we found that this does not necessarily imply the cost to implement the later change is less. This is possibly due to the magnitude of individual change that can be explained by the change attributes, such as *change types*, the *reason for change*, or the *sources of change*.

7.7.2.2 Missing Requirements

Overall analysis provides insight into the effectiveness of requirements engineering practices in general, and into requirements elicitation and analysis in particular. It was observed that the development team did not spend as much time

and effort in exploring and elaborating requirements earlier in *Project Release_B* as they did in *Project Release_A*. We observed that in *Project Release_B* there were many more missing requirements identified in later stages of its development lifecycle than those in *Project Release_A*.

Furthermore, the software development lifecycle models used were different in the two releases. *Project Release_A* used the *Waterfall* approach, and it seems that the RE effort was more focused and performed continuously for the entire release. However, in *Project Release_B*, an *iterative* software development methodology was used, in which RE effort was somewhat fragmented. This may have contributed to the increase in missing requirements, especially because at GDS, process and tool support for iterative development did not seem to be adequate. It should be noted that the new change control process was not implemented during *Project Release_B* to take advantage of the improvements made to the existing change process and change request form.

7.7.3 Future Research Directions

The findings derived from this longitudinal case study provide insights into established practices from a software development organisation. There are several outstanding issues that could be addressed by future work.

7.7.3.1 Requirements engineering best practice

The findings presented in this thesis highlight various important issues related to requirements change analysis, such as requirements traceability, requirements change impacts, and early prediction of change. This study provides an opportunity for future work to extend RE best practices in detecting changes to requirements earlier or combining requirements traceability technique and RV classification to better analyse the impact of change.

7.7.3.2 Requirements Volatility study for other domains or different types of organisation

This requirements volatility study represents the case of a single organisation (GDS). Further research should be conducted to investigate requirements volatility within different organisations or domains, such as safety and security systems, web systems, and banking and insurance companies. In addition, the requirements change classification constructed in this study should be validated in different organisational settings.

7.7.3.3 Cost/Effort Analysis

The cost of implementing RV has not been intensely explored. In this case study, the cost of RV is not well understood. It is associated with the estimation of effort (hours) to implement changes. This research highlights the importance of understanding the cost of change. The findings presented in this thesis provide valuable information for future research on cost estimation models.

Further research should also address the analysis of cost (effort) versus the magnitude/types of changes, or classifying changes based on where in the software development lifecycle they occur and their actual effort spent to implement them.

7.8 Reflections on the Research

Research is all about a systematic approach of inquiry aimed at discovering, interpreting, and revising facts. Throughout this research process I learned the meaning of research and through my curiosity I gained substantial knowledge about the issues under investigation. I learned to use a systematic approach in defining and refining research objectives/questions, choosing appropriate research methods to answer the research questions, analysing data, and reporting the research findings. The research process is a cyclic process of reading, understanding, asking questions, researching, reading, etc, which I found is explorative, interesting, and sometimes leads to unexpected results. Through the research process I also learned the

importance of accurate and reliable data, and interpreting and synthesising the findings.

Conducting case study research in an organisational setting is a challenge. I learned that an industry environment is significantly different from the academic research environment. There were many issues and distractions involved in the organisation and I had to stay focused on the issues that I was investigating. The case study plan is very useful to keep the research on track. I learned that conducting a case study in the industry environment is advantageous because one can learn a great deal from practice.

7.9 Concluding Remarks

The findings of this investigation add value to the knowledge of requirements engineering process in particular and the discipline of software engineering in general. The results presented in this thesis provide new insights on handling RV. The key findings that improve our understanding of the RV issues include *the rates of RV throughout software development process, RV characteristics, the cost of RV, and the development of change control process.*

The *Change Analysis Approach* used and *Requirements Change Classification* constructed in this thesis add significantly to the knowledge of software requirements change analysis technique. The attributes of the classification will likely evolve and can be adopted by other organisations to refine and use within their environment. The classification also provides a basis for other researchers who are interested in the requirements volatility issues to continue working towards modelling requirements volatility more holistically.

This thesis addresses the problem of requirements volatility and empirically analyses its characteristics to improve our understanding on the underlying causes of requirement volatility during software development lifecycle. The results draw attention to various aspects of requirements volatility and provide significant empirical evidence on requirements volatility problems in practice.

Bibliography

- ARMOUR, F., CHATHERWOOD, B. & BEYERS, C. (2001) A framework for managing requirements volatility using Function Points as currency. In the Proceeding of *European Software Control & Metrics Conference*.
- BELL, T. E. & THAYER, R. H. (1976) Software Requirements: Are they really a problem? In the Proceeding of *2nd International Conference on Software Engineering*. San Francisco.
- BOEHM, B. (1988) A spiral model for software development and enhancement. *IEEE Computer*, Vol.21, Issue 5, May, 61-72.
- BOEHM, B. W. (2002) Get ready for Agile methods, with care. *Computer*, Vol.35, Issue 1, 65-69.
- BOEHM, B. W. & BASILI, V. R. (2001) Software defect reduction top 10 list. *IEEE Software*, Vol.18, Issue 1, January, 135-137.
- BOEHM, B. W. & PAPACCIO, P. N. (1988) Understanding and controlling software cost. *IEEE Transactions on Software Engineering*, Vol.14, Issue 10, 1462-1477.
- BOHNER, S. A. (2002) Software Change Impact: An Evolving Perspective. In the Proceeding of *the International Conference on Software Maintenance (ICSM'02)*. IEEE Computer Society.
- BOWEN, P. L., BOWEN, P. L., HEALES, J. & VONGPHAKDI, M. T. (2002) Reliability factors in business software: volatility, requirements and end-users. *Information Systems Journal*, Vol.12, Issue 3, 185.
- BROOKS, F. P. (1987) No Silver Bullet: Essence and Accidents of Software Engineering. *Computer*, Vol.20, Issue 4, April, 10-19.
- BUBENKO, J. (1993) Extending the scope of information modelling. In the Proceeding of *4th International Workshop on the Deductive Approach to Information Systems and Databases*.

- BURNS, R. (1997) *Introduction to Research Methods*, 3rd (Ed.). Australia, Addison Wesley, Longman Australia Pty. Ltd.
- BUSCH, C., DE MARET, P. S., FLYNN, T., KELLUM, R., LE, S., MEYERS, B., SAUNDERS, M., WHITE, R. & PALMQUIST, M. (2005) Content Analysis. Colorado State University Department of English, <http://writing.colostate.edu/references/research/content/>, 21 July 2005.
- BUTCHER, G., BUTCHER, G. & SCHROEDER, C. (1999) A model for addressing software volatility in new system development. *Journal of Information Sciences*, Vol.118, Issue 1-4, 99.
- CHARETTE, R. N. (2005) Why Software Fails. *IEEE Spectrum*, September, 42-49.
- CHRISSIS, M. B., KONRAD, M. & SHRUM, S. (2003) *Guidelines for Process Integration and Product Improvement*, Boston, Pearson Education, Inc.
- CHRISTEL, M. & KANG, K. (1992) Issues in requirements elicitation. Pittsburgh, Carnegie Mellon University. Pittsburgh, Carnegie Mellon University.
- CHUDGE, J. & FULTON, D. (1996) Trust and co-operation in System Development: applying responsibility modelling to the problem of changing requirements. *Software Engineering Journal*, Vol.11, Issue 3, 193-204.
- COSTELLO, R. & LIU, D. (1995) Metric for Requirements Engineering. *Journal of Systems and Software*, Vol.29, 39-63.
- CRESWELL, J. W. (2003) *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*, Second edition. Lincoln, Sage Publications.
- CROTTY, M. (1998) *The foundations of social research: meaning and perspective in the research process*, Allen & Unwind.
- CURTIS, B., KRASNER, H. & ISCOE, N. (1988) A field study of the software design process for large systems. *Communications of the ACM*, Vol.31, Issue 11, 1268-1287.
- DAMIAN, D., ZOWGHI, D., VAIDYANATHASAMY, L. & PAL, Y. (2002) An Industrial Experience in Process Improvement: An early assessment at the Australian Center for Unisys Software. In the Proceeding of *International Symposium on Empirical Software Engineering (ISESE'02)*. Nara, Japan, IEEE.

- DAVIS, A. (2005) *Just Enough Requirements Management*, Dorset House Publishing, New York
- DAVIS, A. & LEFFINGWELL, D. A. (1999) Making Requirements Management Work for You. *CROSSTALK The Journal of Defence Software Engineering*, April.
- DAVIS, A., OVERMYER, S., JORDAN, K., CARUSO, J., DANDASHI, F., DINH, A., KINCAID, G., LEDEBOER, G., REYNOLDS, P., SITARAM, P., TA, A. & THEOFANOS, M. (1993) Identifying and Measuring Quality in a Software Requirements Specification. IN THAYER, R. H. & DORFMAN, M. (Eds.) *Software Requirements Engineering*. 2nd ed., IEEE Computer Society.
- FAGAN, M. E. (1986) Advances in software inspections. *IEEE Transactions on Software Engineering*, 12, 7, 744-751.
- FENTON, N. E. & PFLEEGER, S. L. (1996) *Software Metrics: A rigorous and practical approach*, 2nd (Ed.). International Thomson Computer Press.
- FERREIRA, S., COLLOFELLO, J., SHUNK, D., MACKULAK, G. & WOLFE, P. (2003) Utilization of process modeling and simulation in understanding the effects of requirements volatility in software development. In the Proceeding of *International Workshop on Software Process Simulation and Modeling*. Portland, USA.
- FORRESTER, J. W. (1971) *Principles of Systems*, Cambridge, Productivity Press.
- GHOSH P. P/, V. J. C. (2004) Globally distributed product development using a new project management framework. *International Journal of Project Management*, 22.
- GILLHAM, B. (2000) *Case Study Research Methods*, London, Continuum.
- GOTEL, O. & FINKELSTEIN, A. (1994) An analysis of the requirements traceability problem. In the Proceeding of *1st International Conference on Requirements Engineering (ICRE'94)*. Colorado Springs.
- HAMMER, T., HUFFMAN, L. & ROSENBERG, L. (1998) Doing requirements right the first time. *CROSSTALK The Journal of Defence Software Engineering*, December, 20-25.
- HARKER, S. D. P. & EASON, K. D. (1993) The change and evolution of requirements as a challenge to the practice of software engineering. In the Proceeding of *IEEE International Symposium on Requirements Engineering*.

- HARRIS, D., HEITMEYER, C., MCKENZIE, L. E., PARKER, J., PLACE, P., REID, B., STEPHEN, K., SUMRALL, G. & WOOD, B. (1991) Requirements engineering and analysis workshop proceedings. Pittsburgh, Pennsylvania, Carnegie Mellon University.
- HIGHSMITH, J. & COCKBURN, A. (2001) Agile software development: the business of innovation. *IEEE Computer*, Vol.9, September.
- HSIA, P., DAVIS, A. M. & KUNG, D. C. (1993) Status Report: Requirements Engineering. *IEEE Software*, Vol.10, Issue 6, November, 75-79.
- HULL, M. E. C., JACKSON, K. & DICK, A. J. J. (2002) *Requirements Engineering*, London, Springer.
- HYATT, L. & ROSENBERG, L. (1996) A software quality model and metrics for identifying project risks and assessing software quality. In the Proceeding of *European Space Agency Software Assurance Symposium*. Netherland.
- IEEE-610.12. (1991) IEEE-610.12. IEEE Standard Glossary of Software Engineering Terminology.
- JACOBSON, I. (1999) *The Unified Software Development Process*, Addison Wesley.
- JARKE, M. (1998) Requirements Tracing. *Communication of the ACM*, Vol.41, Issue 12.
- JARKE, M. & POHL, K. (1993) Establishing visions in context: Towards a model of requirements processes. In the Proceeding of *12th International Conference on Information Systems*. Orlando, Florida.
- JONES, C. (1996) Strategies for managing requirements creep. *IEEE Computer*, Vol.29, Issue 6, June, 92-94.
- KITCHENHAM, B. & PICKARD, L. (1995) Case studies for method and tool evaluation. *IEEE Software*, July, 52-62.
- KITCHENHAM, B. & PICKARD, L. M. (1998) Evaluating software engineering methods and tools. *Software Engineering Notes*, Vol.23, Issue 1, 24-26.
- KOTONYA, G. & SOMMERVILLE, I. (2002) *Requirements Engineering Process & Techniques*, John Wiley & Sons.

- KRIPPENDORF, K. (2004) *Content analysis: an introduction to its methodology*, 2nd (Ed.). Beverly Hills, California, Sage Publication.
- LAM, W., LOOMES, M. & SHANKARARAMAN, V. (1999) Managing requirements change using metrics and action planning. In the Proceeding of the 3rd European Conference on Software Maintenance & Re-engineering. IEEE.
- LAM, W. & SHANKARARAMAN, V. (1998) Managing change in software development using a process improvement approach. In the Proceeding of the 24th IEEE Euromicro Conference.
- LAM, W., SHANKARARAMAN, V. & JONES, S. (1998) Managing requirements change: A set of good practices. In the Proceeding of the 4th International Workshop on Requirements Engineering: Foundation of Software Quality. Pisa, Italy, NAMUR.
- LANE, M. & CAVAYE, A. L. M. (1998) Management of requirements volatility enhances software development productivity. In the Proceeding of the 3rd Australian Conference on Requirements Engineering (ACRE 98). Geelong, Australia.
- LEEDY, P. D. & ORMOND, J. E. (2001) *Practical Research: Planning and Design*, Seventh Edition. New Jersey.
- LEFFINGWELL, D. A. & WIDRIG, D. (2000) *Managing software requirements: A Unified Approach*, Addison-Wesley.
- LEHMAN, M. (1998) Software's future: Managing evolution. *IEEE Software*, January-February, 40-44.
- LEHMAN, M. & BELADY, L. (1985) *Program Evolution: Processes of Software Change*, Academic Press.
- LOUCOPOULOS, P. & CHAMPION, R. (1989) Knowledge-based support for requirements engineering. *Journal of Information and Software Technology*, Vol.31, Issue 3, 123-135.
- LUBAR, M., POTTS, C. & RITCHER, C. (1993) A review of the state of the practice in requirements modelling. In the Proceeding of the 1st IEEE International Symposium on Requirements Engineering. IEEE Computer Society.

- LUQI & NOGUEIRA, J. (2000) A risk assessment model for evolutionary software projects. In the Proceeding of *the 7th Monterey Workshop on Modelling Software System*. Italy, University of Genova, Italy.
- LUTZ, R. R. (1993) Analysing Software Requirements Errors in Safety-Critical, Embedded Systems. In the Proceeding of *International Symposium on Requirements Engineering, RE'93*. San Diego, IEEE.
- MACAULAY, L. A. (1996) *Requirements Engineering*, Springer.
- MACQUARIE DICTIONARY (2002) *The Macquarie Dictionary*, Victoria, Australia, The Macquarie Library Pty Ltd.
- MAIDEN, N. A. M. & HARE, M. (1998) Problem domain categories in requirements engineering. *International Journal Human-Computer Studies*, Vol.49, 281-304.
- MAIDEN, N. A. M. & RUGG, G. (1996) ACRE: Selecting methods for requirements acquisition. *Software Engineering Journal*, Vol.11, Issue 3, 183-192.
- MALAIYA, Y. & DENTON, J. (1999) Requirements volatility and defect density. In the Proceeding of *the 10th International Symposium on Software Reliability Engineering*. Boca Raton, Florida, IEEE.
- MARCINIAK, J. J. (2002) *Encyclopedia of Software Engineering*, New York, John Wiley.
- MILES, M. B. & HUBERMAN, A. M. (1994) *An expanded sourcebook: Qualitative data analysis*, 2nd Ed. California, Sage Publications, Thousand Oaks.
- MOYNIHAN, T. (2000a) Coping with 'Requirements-Uncertainty': the theories-of-action of experienced IS/software project managers. *The Journal of Systems and Software*, Vol.53, 99-109.
- MOYNIHAN, T. (2000b) 'Requirements-Uncertainty': Is it best formulated as a latent, aggregate or profile construct? *European Journal of Information Systems*, Vol.9, Issue 2, 82.
- NIDUMOLU, S. (1995) The effect of coordination and uncertainty on software project performance: residual performance risk as an intervening variable. *The Journal of Information Systems Research*, Vol.6, Issue 3, 191-219.

- NIDUMOLU, S. R. (1996a) A comparison of the structural contingency and risk-based perspectives on coordination. *Journal of Management Information Systems*, Vol.13, Issue 2, 77.
- NIDUMOLU, S. R. (1996b) Standardization, requirements uncertainty and software project performance. *Information & Management*, Vol.31, Issue 3, 135-150.
- NURMULIANI, N., ZOWGHI, D. & FOWELL, S. (2004a) Analysis of requirements volatility during software development lifecycle. In the Proceeding of *Australian Software Engineering Conference*. Melbourne.
- NURMULIANI, N., ZOWGHI, D. & WILLIAMS, S. (2004b) Using Card Sorting to classify requirements change. In the Proceeding of *the 12th International Conference on Requirements Engineering*. Kyoto, Japan, IEEE Computer Society.
- NURMULIANI, N., ZOWGHI, D. & WILLIAMS, S. (2005) Characterising requirements volatility: An industrial case study. In the Proceeding of *International Symposium Empirical Software Engineering*. Noosa - Australia, IEEE.
- NURMULIANI, N., ZOWGHI, D. & WILLIAMS, S. P. (2006) Requirements volatility and its impact on change effort: evidence-based research in software development projects. In the Proceeding of *11th Australian Workshop on Requirements Engineering*. Adelaide, Australia.
- ORLIKWOSKI, W. J. & BAROUDI, J. J. (1991) Studying information technology in organisations. Research approaches and assumptions. *Information Systems Research*, Vol.2, 1-28.
- PALMER, J., DUFFY, T., GOMOLL, K., GOMOLL, T., RICHARDS- PALMQUIST, J. & TRUMBLE, J. A. (1988) The design and evaluation of online help for Unix EMACS: Capturing the user in menu design. *IEEE Transactions on Professional Communication*, Vol.31, Issue 1, 44-51.
- PATTON, M. Q. (2002) *Qualitative Research & Evaluation Methods*, 3rd (Ed.). California, Thousand Oaks, Sage Publication.
- PFAHL, D. & LEBSANFT, K. (2000) Using simulation to analyse the impact of software requirement volatility on project performance. *Information and Software Technology*, Vol.42, Issue 14, 1001.
- POHL, K. (1997) Requirements Engineering: An Overview. IN KENT, A. & WILLIAMS, J. (Eds.) *Encyclopedia of Computer Science and Technology*. New York, Marcel Dekker.

- PRESSMAN, R. S. & INCE, D. (2000) *Software Engineering: A Practitioner's Approach*, 5th (Ed.). McGrawHill.
- ROSENBERG, L., HYATT, L., HAMMER, T., HUFFMAN, L. & WILSON, W. (1998) Testing metrics for requirement quality. In the Proceeding of the 2nd *Quality Week Europe*. Belgium.
- ROYCE, W. W. (1970) Managing the development of large software systems: concept and techniques. In the Proceeding of *Technical Paper of Western Electronic Show and Convention (WesCon)*. Los Angeles.
- RUGG, G. & MCGEORGE, P. (1997) The sorting techniques: A tutorial paper on card sorts, picture sorts and item sorts. *Expert Systems*, Vol.14, Issue 2, 80 - 93.
- SAVOLAINEN, J. & KUUSELA, J. (2001) Volatility analysis framework for product lines. *the Communication of the ACM*, 133-141.
- SOMMERVILLE, I. (1996) *Software Engineering*, 5th (Ed.). Addison Wesley.
- SOMMERVILLE, I. (2004) *Software Engineering*, 7th (Ed.). Harlow, England, New York, Pearson/Addison Wesley.
- SOMMERVILLE, I. & SAWYER, P. (1997) *Requirements engineering, a good practice guide*, 1st (Ed.). Wiley.
- STAKE, R. E. (1995) *The art of case study research*, California, Thousand Oaks, Sage.
- STARK, G., SKILLICORN, A. & AMEELE, R. (1999) An Examination of the effects of requirements changes on software maintenance releases. *Journal of Software Maintenance: Research and Practice*, Vol.11, 293-309.
- SUGDEN, R. C. & STRENS, M. R. (1996) Strategies, tactics and methods for handling change. In the Proceeding of *IEEE Symposium and Workshop on Engineering of Computer-Based System*.
- SUTCLIFFE, A. (2002) *User-centred requirements engineering: theory and practice*, London, Springer-Verlag.
- THAYER, R. H. & DORFMAN, M. (1997) *Software Requirements Engineering*, 2nd (Ed.). IEEE Computer Society.

- THE STANDISH GROUP (1994) The CHAOS Report. The Standish Group International, Inc.,
http://www.standishgroup.com/sample_research/chaos_1994.php.
- THE STANDISH GROUP (1998) CHAOS: A Recipe for Success.
- TIWANA, A. & KEIL, M. (2004) The one-minute risk assessment tool. *Communication of the ACM*, Vol.47, Issue 11, 73-77.
- UPCHURCH, L., RUGG, G. & KITCHENHAM, B. (2001) Using Card Sorts to elicit web page quality attributes. *IEEE Software*, July/August, 84 - 89.
- VAN BUREN, J. & COOK, D. (1998) Experiences in the adoption of requirements engineering technologies. *CROSSTALK, The Journal of Defence Software Engineering*, December, 3-10.
- WIEGERS, K. E. (2003) *Software requirements: practical techniques for gathering and managing requirements throughout the product development lifecycle*, 2nd (Ed.). Washington, Microsoft Press.
- YIN, R. K. (2002) *Case study research, design and methods*, 3rd (Ed.). Newbury Park, Sage Publications.
- ZAVE, P. (1997) Classification of research efforts in requirements engineering. *ACM Computing Survey*, 29, 4, 315-321.
- ZOWGHI, D. & GERVASI, V. (2003) On the interplay between consistency, completeness, and correctness in requirements evolution. *the Journal of Information and Software Technology*, 45, 993-1009.
- ZOWGHI, D. & NURMULIANI, N. (1998) Investigating requirements volatility during software development: research in progress. In the Proceeding of *3rd Australian Conference on Requirements Engineering*. Geelong, Australia.
- ZOWGHI, D. & NURMULIANI, N. (2002) A study of the impact of requirements volatility on software project performance. In the Proceeding of *the 9th Asia-Pacific Software Engineering Conference*. Gold Coast, Australia, IEEE.
- ZOWGHI, D. & OFFEN, R. (1997) A logical framework for modelling and reasoning about the evolution of requirements. In the Proceeding of *the 3rd IEEE International Symposium on Requirements Engineering*. Annapolis, USA.

ZOWGHI, D., OFFEN, R. & NURMULIANI, N. (2000) The impact of requirements volatility on software development lifecycle. In the Proceeding of *the International Conference on Software, Theory and Practice (ICS2000)*. Beijing, China.

ZOWGHI, D. & PARYANI, S. (2003) Teaching requirements engineering through role playing: lessons learnt. In the Proceeding of *the 11th International Requirements Engineering Conference*. Monterey Bay, California, IEEE Computer Society.

Appendix A: Ethics Committee Approval Document

7 August 2002

Dr Didar Zowghi
Faculty of Information Technology
Broadway Campus

Dear Didar

UTS HREC 02/69 - ZOWGHI, Dr Didar, (for NURMULIANI - PhD student) -
"An empirical study of requirements volatility and its impact on
software development life cycle"

Thank-you for your response to my letter of 26 June 2002. I have no hesitation in approving your application. Could you please forward a copy of the non-disclosure agreement between Nurmuliani and ACUS for filing purposes.

Your approval number is HREC 02/69A.

The National Statement on Ethical Conduct in Research Involving Humans requires us to obtain a report about the progress of the research, and in particular about any changes to the research which may have ethical implications. The attached report form must be completed at least annually, and at the end of the project (if it takes more than a year), or in the event of any changes to the research as referred to above, in which case the Research Ethics Officer should be contacted beforehand.

I also refer you to the AVCC guidelines relating to the storage of data. The University requires that, wherever possible, original research data be stored in the academic unit in which they were generated. Should you submit any manuscript for publication, you will need to complete the attached Statement of Authorship, Location of Data, Conflict of Interest form, which should be retained in the School, Faculty or Centre, in a place determined by the Dean or Director.

Please complete the attached (green) report form at the appropriate time and return to Susanna Davis, Research Ethics Officer, and Research Office, Broadway. In the meantime, if you have any queries please do not hesitate to contact either Susanna or myself.

Yours sincerely,

Associate Professor Jane Stein-Parbury
Chair
UTS Human Research Ethics Committee

Appendix B: Consent Form

CONSENT FORM

I _____ agree to participate in the research project "An Empirical Study of Requirements Volatility and its Impact on Software Development Life Cycle" being conducted by Nurmuliani and Dr Didar Zowghi, Faculty of Information Technology, University of Technology, Sydney, Po Box 123, Broadway, NSW 2007, Ph: 9514 4445 and 9514 1860.

I understand that the purpose of this study is to investigate requirements volatility that occurs during software development life cycle and its impact on software processes and product quality.

I understand that my participation in this research will involve no harm or risks. I will participate in requirements meetings that the researcher may observe and in a number of interviews, for no longer than an hour at a time

I am aware that I can contact Nurmuliani or her supervisor Dr. Didar Zowghi if I have any concerns about the research. I also understand that I am free to withdraw my participation from this research project at any time I wish and without giving a reason, that this withdrawal will not affect my employment at the company and that any data already collected will be destroyed.

I agree that Nurmuliani has answered all my questions fully and clearly.

I agree that the research data gathered from this project may be published in a form that does not identify me in any way.

Signed by

____/____/____

Witnessed by

____/____/____

NOTE:

This study has been approved by the University of Technology, Sydney Human Research Ethics Committee. If you have any complaints or reservations about any aspect of your participation in this research which you cannot resolve with the researcher, you may contact the Ethics Committee through the Research Ethics Officer, Ms Susanna Davis (ph: 02 - 9514 1279, Susanna.Davis@uts.edu.au). Any complaint you make will be treated in confidence and investigated fully and you will be informed of the outcome.

Appendix C: Project Release_A Change Request Datasheet

Change-ID	Change description	Reasons for Change	Origin	Reason Category	change type	Type of Requirements	Initiated date	Approved date	add	del	mod	Tot
CR_002	Rewording text for clarification	Restatement of original requirements for clarity	Marketing Group	Clarifying requirement	mod	High level Req	27 Feb 02	8 Apr 02			1	1
CR_003	Enhancing Performance test	A new high level requirement is required to address test coverage	Marketing Group	Performance Enhancement	add	High level Req	4 Mar 02	20 Mar 02	1			1
CR_004	Removing obsolete requirement	Functionality is no longer required for this release	Marketing Group	Obsolete functionality	del	High level Req	20 Mar 02	8 Apr 02			1	1
CR_005	Removing obsolete requirement	Functionality is no longer required for this release	Marketing Group	Obsolete functionality	del	High level Req	20 Mar 02	15 Apr 02			1	1
CR_014	Adding new high level requirement	A new high level requirement is needed to enable linking of all 'engineering infrastructure' requirements to a high level requirement	Engineering Analysis	Missing Requirements	add	High level Req	5 Apr 02	15 Apr 02	1			1
CR_015	Adding new high level requirement	New requirement is required to maintain functionality (upgrade facility) for the product releases	Project Management	Functionality enhancement	add	High level Req	8 Apr 02	20 May 02	1			1
CR_016	Deleting functionality that provides no benefit for the potential users	Based on further design work conducted (design refinement), it revealed that the functionality has no value to the potential users	Design Review	Obsolete functionality	del	Low level Req	8 Apr 02	15 Apr 02			5	5
CR_017	Adding new functionality to maintain a discontinued PCE function	new functionality is added as an alternative support to replace "PCE/GIW" that is not available for this release	Design Review	Functionality enhancement	add	Low level Req	9 Apr 02	24 Apr 02	9			9
CR_018	Deleting sub-feature (upgrade functionality) that can be handled by other existing functionality	Since current limitation on other hosts do not require upgrade of "D-ISAM", and it can be handled by developer test	Design Review	Redundant Functionality	del	Low level Req	10 Apr 02	24 Apr 02			4	4
CR_019	Removing sub-feature that is not needed	removing sub-feature in order to have all components on the same level of functionality, also since some components do not have the resources to migrate to newer level	Design Review	Redundant Functionality	del	Low level Req	10 Apr 02	1 May 02			2	2
CR_020	Adding new feature/ new requirements to maintain/managing functionality	New feature suggestion (NFS) is introduced to replace a partially decommissioned functionality in particular environment (LITE) for this release	Project Management	Functionality enhancement	add	Low level Req	10 Apr 02	7 May 02	12			12
CR_022	Adding new requirements to introduce Configuration utility in order maintain the connection and configuration details	Improved maintenance functionality or to create and maintain connection and configuration details	Design Review	Design improvement	add	Low level Req	12 Apr 02	21 May 02	1			1
CR_023	Several requirements are modified to prevent parameter/version checking functionality conflicts. One requirement is deleted since not needed anymore	Modify requirements for CS implementation to eliminate the need for Graphical Presentation GUID to be propagated through the system (to eliminate parameter/version checking conflicts)	Technical Team Discussions	Resolving conflicts	del	Low level Req	15 Apr 02	3 May 02			1	1
			Technical Team Discussions	Resolving conflicts	mod	Low level Req		3 May 02			7	7
CR_024	Modifying requirements that refer to Unix, which is not included for the 3R3 release	Unix is not included for HDBA in this release due to technical reason (interoperation with COTS)	Feature Proposal Review	Clarifying requirement	mod	Low level Req	16 Apr 02	12 May 02			3	3

Change-ID	Change description	Reasons for Change	Origin	Reason Category	change type	Type of Requirements	Initiated date	Approved date	add	del	mod	Tot
CR_025	Adding new requirements to address security functionality	New requirements are required for new functionality (developer security)	Functional Specification Review	Design improvement	add	Low level Req	16 Apr 02	9 May 02	2			2
CR_026	Deleting a requirement that is no longer required, and to be replaced by modifying other requirement to address specification changes in data transfer mechanism	Specification changes in data transfer mechanism	Feature Proposal Review	Obsolete functionality	del	Low level Req	18 Apr 02	13 May 02		1		1
CR_026			Feature Proposal Review	Functionality enhancement	mod	Low level Req		13 May 02			1	1
CR_027	Removing requirement that is qualified for Caldera Open Unix 8 since this platform has no Oracle support. Adding new reqs that 3R3 will be supported on Unixware 7.1.1 since the Caldera OU 8 is the next release of Unixware	There is no Oracle support for Caldera OU 8 operating system, however the functionality of this release continues to be supported on Unixware	Functional Specification Review	3rd party software influence	del	Low level Req	18 Apr 02	23 May 02		1		1
CR_027			Functional Specification Review	3rd party software influence	add	Low level Req	18 Apr 02	23 May 02	1			1
CR_028	Dropping sub-features from product scope	Resource constraints, engineers moving into next release, and other engineers work on other higher priority features	Project Management	Scope reduction	del	Low level Req	23 Apr 02	22 May 02		4		4
CR_030	New requirement is added to address specification deficiency (Reserved words)	Fixing bugs from previous releases	Defect Report	Defects Fix	add	Low level Req	6 May 02	30 May 02	1			1
CR_032	Deleting redundancy requirements; Modifying requirements text for clarification	Requirements redundancy, it's not essential or already exists in the previous release, and it's not requirement; Modify existing requirements to qualify EAE for the NT operating systems	Functional Specification Review	Requirements redundancy	del	Low level Req	13 May 02	30 May 02		3		3
CR_032			Functional Specification Review	Clarifying requirement	mod	Low level Req		30 May 02			1	1
CR_033	Rewording requirement to address specification changes related to version checking	Follow-up (CR_023), on version checking, GUID functionality removed	Functional Specification Review	Resolving conflicts	mod	Low level Req	10 May 02	30 May 02			1	1
CR_034	Rewording requirement to address specification changes (functionality not required)	Rewording requirements to address specification (Site Builder utility) changes	Technical Team Discussions	Design improvement	mod	Low level Req	10 May 02	31 May 02			1	1
CR_035	Amend requirement on User Maintained Database tables to address consistency checking	Runtime application does not need database consistency checking	Customer Support discussion	Clarifying requirement	mod	Low level Req	21 May 02	31 May 02			1	1

Change-ID	Change description	Reasons for Change	Origin	Reason Category	change type	Type of Requirements	Initiated date	Approved date	add	del	mod	Tot
CR_037	Modifying requirements in ROC/ADHOC feature to address changes in the screens capability (Graphical Interface Workbench); Create new requirements in ROC/ADHOC feature to address D2L and 1.2D utilities	Rewording several requirements to include all the screen from all host runtime systems can be accessed in GIW; Create new requirements in ROC/ADHOC feature to enable two functionalities (Developer and GIW) exchange screens definition	Detailed Design Analysis	Functionality enhancement	mod	Low level Req	24 May 02	21 Jun 02			9	9
CR_037			Detailed Design Analysis	Functionality enhancement	add	Low level Req		21 Jun 02	3			3
CR_038	new functionality is added to replace the related functionality that has been omitted	Additional functionality is needed as a result of omitting related functionality,	Design Review	Functionality enhancement	add	Low level Req	29 May 02	30 May 02	1			1
CR_039	Deleting the requirements for 'listbox data handling' on Developer/Builder environment	Deleting the requirements due to complex integration or inflexibility between two environments (Dev/Builder and GIW environments)	Detailed Design Analysis	Functionality enhancement	del	Low level Req	30 May 02	19 Jul 02		2		2
			Detailed Design Analysis	Functionality enhancement	mod			19 Jul 02			2	2
CR_042	Modifying/rewording requirement to include Business Integrator capability	Amend requirements to address a request from BI that the Business Integrator should be delivered on the Client Tools CD-ROM	Marketing Group	Product Strategy	mod	Low level Req	11 Jun 02	17 Jul 02			1	1
CR_045	Adding new requirements to manage error handling option	New requirements are required to handle errors correctly and to solve functionality inconsistency b/w current release and previous releases	Defect Report	Defects Fix	add	Low level Req	21 Jun 02	18 Jul 02	3			3
CR_046	Adding new feature/ new requirements to provide release media mechanism	Need to provide a mechanism for this release,	Marketing Group	Product Strategy	add	Low level Req	26 Jun 02	18 Jul 02	8			8
CR_047	New requirements are added to address functionality upgrade (migration)	new requirements are required to update (LCU) functionality for 3.3 releases, to keep customers stay within 3.x product	Detailed Design Analysis	Functionality enhancement	add	Low level Req	26 Jun 02	20 Aug 02	6			6
CR_048	Adding new requirements to address specification changes for VB Guidelines	new requirements to address/implement BI requirements on VB migration for CS feature	Marketing Group	Functionality enhancement	add	Low level Req	27 Jun 02	29 Jul 02	1			1
CR_050	Deleting requirement that is not available for particular platform	Requirement that support Windows XP Professional in Window Infrastructure is removed since it's only available for Windows Runtime	Technical Team Discussions	Erroneous requirement	del	Low level Req	8 Jul 02	29 Jul 02		1		1
CR_052	Addin new requirement to address new drawing functionality	new requirement to satisfy PREQ76, lines and boxes support	Marketing Group + Detailed Design analysis	Design improvement	add	Low level Req	8 Jul 02	13 Aug 02	1			1
CR_053	Adding new requirements to ensure Business Integrator Module, which was developed in the post 3.2 release, works with Developer test in the 3.3 release	New functionality has been created in this release to maintain functionality from previous release	Project Management	Functionality enhancement	add	Low level Req	8 Jul 02	16 Aug 02	2			2

Change-ID	Change description	Reasons for Change	Origin	Reason Category	change type	Type of Requirements	Initiated date	Approved date	add	del	mod	Tot
CR_054	Modifying requirements to updated browser functionality	Modify requirements to reflect the minimum browsers level that will be developed, tested, supported by CE Viewer	Technical Team Discussions	Design improvement	mod	Low level Req	16 Jul 02	13 Aug 02			2	2
CR_055	Rewording requirement text for functionality simplification	Simplify Multi-line display functionality to reduce effort and complexity	Detailed Design Analysis	Design improvement	mod	Low level Req	16 Jul 02	9 Aug 02			1	1
CR_061	Adding new requirement for functionality enhancement	New requirement is added to enhance the utility CheckDB functionality in handling tables and indexes	Detailed Design Analysis	Design improvement	add	Low level Req	5 Aug 02	20 Aug 02	1			1
CR_065	New requirement is added to reflect the removal of Corporate Identity e-actionsub-brand.	Change in Corporate Identity, but the products name will not be changed	Marketing Group	Product Strategy	add	Low level Req	13 Aug 02	22 Aug 02	1			1
CR_68	Adding new requirement to capture the need for the CE Viewer installation program to operate in silent mode, to be qualified in a Terminal Service environment, and to make it visible to test teams	Add new requirement to capture the need for the CE Viewer installation program to operate in silent mode, and to make it visible to test teams	Marketing Group + Detailed Design analysis	Design improvement	add	Low level Req	26 Aug 02	13 Sep 02	1			1
CR_69	Adding new requirements to improve remote configuration file handling functionality	To capture the need for the CE Viewer to be qualified in a Terminal Service environment	Marketing Group + Detailed Design analysis	Design improvement	add	Low level Req	26 Aug 02	13 Sep 02	1			1
CR_070	Adding new requirement/ functionality for Web environment	To improve WebServer environment deficiency in CE Viewer feature	Marketing Group	Functionality enhancement	add	Low level Req	28 Aug 02	13 Sep 02	1			1
CR_072	Adding new requirements to address changed packaging style and simplify licensing for Developer and Builder products	To simplify licensing for Developer aand Builder products (Builder licences will no longer required)	Marketing Group	Product Strategy	add	Low level Req	12 Sep 02	18 Oct 02	2			2

Appendix D: Card Sorting Feedback Sheet

A. Your Role

1. General Information

Please fill in the following table with the relative information

Name (optional)	
Position	
Date	

2. How many years have you acted in this role?

3. How many years have you worked on software application development?

B. Classification of Changes

What benefits might a classification of changes provide?

Do you think a classification of changes will be useful to you?

Do you have any suggestions in developing taxonomy of changes?

C. Card Sorting Method

What do you think of this Card sorting method in gathering information?

Do you have any other suggestions to improve this method?

Thank you for your participation

Appendix E: Participants' Card Sorting Result

Respondents	Role	# Category	Criteria	Category #1	Category #2	Category #3	Category #4	Category #5	Category #6	Category #7	Category #8	Category #9	Category #10
MH (Senior Manager)	SM1		5 reason for changes	Product Marketing or Packaging/Licensing or Branding	Removal from Product Scope (2 reasons: Resources and Technical (done elsewhere or not needed))	Modification caused during development or Discovered/Increased Knowledge	Hardware/Software Environment or Conformance to Industry Trends	Addition to overall Reqs set/ Realised we have missed Reqs					
HB (Systems Architect)	Eng1		4 the magnitude of effort involved	High Effort (New/Changed functionality in known complex area)	Medium Effort (New or changed reqs that have reasonable impact on effort)	Low Effort (Minor changes to reqs/functionality)	No Effort (Mainly Rework of Reqs or rewording)						
JW (Technical Lead)	Eng2		8 reason for changes	New Reqs (Internal)- arising from further design experience gained - New Reqs (missing)- identified via traceability review(s)	Software Environment-related to interoperability with COTS	Migration - related to maintaining/managing functionality as the product is released at known versions	Interdependencies - resolving reqs overlaps or redundant/repeated reqs	Testability - Modification to increase test coverage or ensure that test correctly address requirements	Installation/Media/Packaging - Related to the physical form in which the product is delivered	Clarification - Rewording to eliminate chances for misinterpreting the requirement	Scope reduction - eliminating reqs deemed to be of low or no value to customers		
AL (System Architect)	Eng3		5 reason for changes	Incomplete Reqs - arising from reqs capture/review	Architectural Incompleteness	External Influences/Customer (BI)/ 3rd party software	Unnecessary Reqs captured	Scope Reduction					
YP (Project Manager)	PM1		10 reason for changes	New/Add - due to missed reqs at initial definition of Product Reqs (these reqs should be captured in the beginning)	Reqs deleted - due to top level changes	Reqs deleted - due to wrong requirements	Delete functionality based on Management considerations - Budget/Schedule	Reqs addition due to detailed considerations	Reqs deletion due to detailed design discovery	Modify reqs due to low level design	Modify reqs due to 'other reqs attributes' eg. Test scenario/test spec triggered changes to	Requirements Clarification	Modify Reqs Attributes

Respondents	Role	# Category	Criteria	Category #1	Category #2	Category #3	Category #4	Category #5	Category #6	Category #7	Category #8	Category #9	Category #10
KK (Consulting Engineer)	SM2	4	reason for changes	Major Changes: New Reqs may affect commitments schedule	Minor Changes: rewording/ removing duplicates or redundant	Packaging/Branding/New major functionality	Medium Changes: Clarifications: normally code, PI, & Testing impact						
DC (System Architect)	Eng4	9	reason for changes	Scope reduction - resource availability	New/better functionality identified by engineering	New/better functionality identified by "customer"	Consistency of configuration/platform - supported hardware/software	Change Requirements to match design	Bug Fix	Test Scenario change (Reqs attributes)	Internal Consistency Internal Review	Change in external software	
IS (Engineering Manager)	ML1	9	General changes	New Requirements Added: Functionality upgrade, Different platform, Product Strategy/feature	Drop Reqs	Modify Reqs	Reqs clarification	Activity	Doc related (PI)	Operational issue related			
MC (Engineering Manager)	ML2	9	General changes	Client Tool related changes; Supported platform changes; Add/new non functional req	Requirements modified; Reqs modified; Database related	Infrastructure related changes	All other functional/Reqs changes	Sub-feature related changes					
DB Project Manager)	PM2	6	Schedule impacted	New Reqs with high impact on schedule (a lot of new work)	New Reqs with low impact on schedule (only little work added)	Modified Reqs with significant impact on schedule	Modified Reqs with little or no impact on the schedule	Neutral effect on schedule or reduced effect (I.e. less work than foreseen)					
Total		68											
Min		4											
Max		10											
AVG		6.8											

Appendix F: Participants' Feedback on Card Sort

MH transcripts:

Qb. Ranking criteria: the important of changes or making changes to the product

Qb1 Benefits of Classification: provide you with some sort of structure: you don't need to view all the changes are being the same or being equally important and give you method for categorising changes and once you can define things up into a nice neat little categories and decide the problem actually diminishes. It gives you a mean of controlling and managing and sizing changes.

Qb2 The important of taxonomy of changes: Yes. definitely, caused we tend to regard all CR or variations equally, whereas some more important than others. My appealing is the important of them vary is depending on **where do they come from or who they come from.**

Qb3. in the taxonomy should have syntax that everybody knows

Qc1 Card sorting method: it's good and force people to make decision

JW transcripts:

Ranking criteria: -negative impact on schedule

Qb1 Benefits of classification (taxonomy) might provide: if we can work it out mapping the impact for each category, then we can get a reliable way of trying to assess how much impact each requirement can have. It also can promote common understanding within groups of what the change means

Qb2 A taxonomy of changes will be useful to you?: Yes I think so. I think the taxonomy is great as long as it doesn't kind of hide the reality, two things can be rolled: -people start making assumptions about it and they misunderstand the software requirements changes, I think they know what the changes but actually they don't really. The other thing is some way it can stop you thinking in if someone already makes decision about software requirements changes unless you want to challenge that you get way with the changes.

Qb3 Suggestions development taxonomy: best way to find the commonality between people opinion about classification.

HB transcript:

Criteria for classification: -the magnitude of effort involved, for example, how much work was involved in removing requirements where certain number of effort is needed for major changes.

Qb1. Benefits of taxonomy of changes may provide:

- identify **risk** according to the group of changes, such as a potential high risk cost more than lower risk.

Qb2.

- I guess, my perspective I think we tend to think a needful in this term of now or lots of cases, b/w the categorise things by the risk factor

Qb3. suggestion for taxonomy: - my idea was have the impact of change, maybe the combination of things or maybe emerging various one and come up with the right taxonomy.

Qc1: what do you think Card sorting method: - seeing the information is presented in isolation, it makes probably put it in different perspective as a part of large document. It makes you think differently maybe if you're reading in at the basic fact of requirements, and it will give you different answer and I think card sorting is a nice way to get a fresh look at.

Qc2. suggestions: none! because this is the first time. It's a good method if you go with other information...

3R3 is the project with an incremental product improvement. While 4.1. has more than that like building new car..

DB transcript:

Criteria for classification: - the type of requirements change in term of new/addition, changed/modification, and drop/deletion.

Qb1. Benefit of change classification would be provided:

- ease of determining the change request acceptability, or
- easier to make decision such as to approve or reject request change.

Qb2. Taxonomy will be useful?: - Yes for the reasons above.

Qb3. Suggestion about taxonomy: - well actually those my five categories would probably help. Only the last group (drop requirements) that I didn't split them. Maybe sometimes dropping requirements may mean more work at the certain circumstances (I am not sure). Drop requirements with no impact on schedule or with some impact. Obviously, interdependency would be included in the taxonomy. Something with **interdependency across many areas** would be more likely to be high impact on schedule.

From my perspective, the taxonomy items should include the effort involved and the impact on schedule, also the consequences if the change request is rejected or is not implemented.

Also there is *something that not specified in the change request form such as we need to change this as a result of the reviews, ok fine, but what the consequences if we not change it.* Is is a big thing?, is it that something to be look nicer, etc.

Qc1. Card sorting method:- Well what do you mean by information. The thing that useful for me is to start thinking about categories. It forces me to put cards or CR into some sort of categories that you never thought before.

Qc2. I don't know (this is the first time). The card sorting was good as a practical to get my mind going.

AL transcripts:

Criteria: - scoping, lack of clarity in the architecture, insufficient knowledge in problem domain.

Qb1. Benefit of change classification would be provided:

- a means for to focus what question to ask about each requirement, requirements capture process in general or during the requirements capture process.
- It allows you asses the impact on the schedule.

Qb2. Taxonomy will be useful?

- Yes, I think so. Because when people reviewing CR they can look at the classification and it can give some ideas to what the impact of the changes. So the reviewers of CR can think about it in different way.
- The taxonomy in CR form?: I think that would make it easier to focus, for instance if we missed this something that we should be doing , the fact is something we concern reviewing the requirements.....

Qb3. Suggestion about taxonomy: I thin it's a good idea. Something comes up with appropriate labels that people can agree upon. A subset that has meaning to everybody.

Qc1.: Card sort: It's useful technique because you can separate or sort them easily.

Qc2. it can allow you divorce one issue to the rest. And you can rank them in term of importance.

YP transcripts:

Criteria: Change classification that based on causes where they should been discovered.

Hierarchy classification: top level, medium level, and low level.

Qb1. Benefit of change classification would be provided:

- it provides an early ability to examine where the changes are coming from,

- it provides a framework on which the based on your change for proper consideration, obviously gives you an ability to maintain matrix and analyse the data at the later on

Qb2. Taxonomy will be useful?: Yes, I think so. It would be useful to feed back into the process. i.e. You look at a different number of change in different area and you can say this area is weak especially when you are doing root cause, fixing up, and enhancing the process. "A cause based taxonomy"

Qb3. Suggestion about taxonomy: - First, for a lot of people having taxonomy of changes is very useful thing to actually pin the changer, although maybe need to be two level things, the first level says this is what you get is going to be and then you do root cause actually come to any changes to that classification. You might classify a change as requirements clarification but it ends up as missing requirements. Second, if the taxonomy can be a phase based, some phases could be classified.

Qc1.: Card sorts method: I think it's a great method. I've never used this before. It's a good way actually thinking through and trying to work out the effort. It's not only forced you to make decision, it also gives you an ability to make decision. Open card sorting is a better way to gather information in exploratory way.

QC2; gives a clear instruction (half page) to participants.

KK transcripts

Criteria for classification: reason for changes based on the functionality that we agree to develop (much more worried about something that very visible from outside that we agree to deliver)

Ranking criteria: making changes that affect the high level of requirements

Qb1.....(not finished)

DC questionnaire:

Criteria: reasons or causes of change

Qb1. Benefit of change classification would be provided:

- to help in analysing the importance of change or prioritizing in terms of new functionality
- to pay more attention to particular/importance change request that related to functionality

Qb2. Taxonomy will be useful?:- Yes,

Qb3. Suggestion about taxonomy: yes, particularly if we have a multidimensional classification, i.e. two dimensions: causes of change and the impact of changes.

MC transcripts:

Criteria: - common things such as general changes (add/del/mod) and specific one like functional or non functional changes

Qb1. Benefit of change classification would be provided:

- I guess you can find specific changes much quicker,
- The changes that are related together you can manage them in similar manner

Qb2. Taxonomy will be useful?:- not really, because there is no terminology changes anyway. It means different sort of changes with different type so it's not like when we've to schedule the changes one up to the other and then drop some of them. Basically whenever change is accepted we have to do it. Therefore, having classification doesn't help, I mean we know the changes then we have to do it.

Qb3: I guess what more important to us would be of whether we want the changes or not. In our change process, it starts with propose changes, review changes, and accept/reject changes. **Maybe before CR being approved, the classification maybe helps reviewers to make decision or accept/reject the request change.**

Classification should be incorporated into the Change Control Process during 'CR Review' activity.

Why we want the classification? Requirements classification is more related to component architecture. It a good idea in analysing changes with component architecture based.

Qc1. Card sorting method: there is other way for categorisation, sort it using spreadsheet.

IS transcripts

Criteria: - general changes (addition/deletion/modification) and then it can be break it up into more detailed changes (by reasons)

The priority will be based safe passage to support customers.

Qb1. Benefit of change classification would be provided:

- better organise,
- help us to ensure that fully understand the clarification of CR, rejecting/accepting the CR, the impact on the operation, the operation itself impacting the schedule, and getting organise,.. making sure that we always know ...amm the... to help a complete Change control on overall requirements for the particular project so in that way we can be 100% sure the product to be delivered.

Qb2. Taxonomy will be useful to you?:- of course, because it helps us better organise and it helps usto ensure that the project will be.... We cannot allow requirements be crept in because will impact us at the end to delivered the product. If we have controlled over the changes by classifying them, it make sure when getting new reqs and knowing its impacts, then we are able to reject or approve, at least we will have opportunities to say no that we cannot delivered because it impact on the schedule in this way or it doesn't fit our strategy.

Qb3. Suggestion on Taxonomy of change: - there always a room for improvement...in the post-mortem project

Qc1. Card sort method: It's a good exercise. It's quickly to sort out things and (in the statement) logically to break down things.

Qc2. suggestion: It's cool!!

Appendix G: Pre- Survey Questionnaire on Change Control Process

CHANGE CONTROL PROCESS EVALUATION SURVEY QUESTIONNAIRE

About The Survey

The purpose of this survey is to investigate random ACUS users' views and opinions about the current Change Control Process at ACUS (Change Control Process revision 00-00-12).

- The information you provide will help the company to evaluate and improve its change process. It will also contribute to the research being carried out by UTS researchers at ACUS.
- It will take approximately 10 minutes of your time to complete the survey.
- There is no right or wrong answers; we are seeking your opinions and perceptions about the current Change Control Process. Your opinion is important to us.
- **Remember THE SURVEY IS STRICTLY CONFIDENTIAL.** All responses will be aggregated; no individual will be identifiable from the aggregated responses.
- When you have finished, please send the completed survey by 30th of April to Nur at nur@it.uts.edu.au

Thank you for your assistance.

Section 1: THE CHANGE CONTROL PROCESS

Instructions: All respondents should complete this section

1. Please indicate your current Functional Area: test
2. How would you rate your understanding of the current Change Control Process?

☐ Excellent ☐ Very good ☐ Good ☐ Fair ☐ Poor ☐ Not sure
3. To what extent is the Change Control Process important for you in your work?

☐ Very important ☐ somewhat important ☐ not important ☐ Not sure
4. Please indicate the importance of the following items relating to the Change Control Process (select the appropriate checkbox for each item)

	Lowest	Low	Medium	High	Highest	NA
Reporting of problem/issue	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Evaluation of Change Requests	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Performing change impact analysis	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Track work in process changes	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Perform Change Request Review meeting	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Approval of Change Requests	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Develop tasks for change implementation	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Verify change against product deliverables	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

SECTION 2: USING THE CHANGE REQUEST FORM TO SUBMIT A CHANGE

1. Have you ever initiated or submitted a Change Request?

☐ YES

☐ NO (if no, please go to section 3)

2. Please indicate the extent to which you agree (disagree) with each of the following statements

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree	Not Sure
The current Change Request (CR) Form is easy to use	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The current CR form is easy to understand	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The current CR form covers all aspects related to a requested change	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Information is sufficiently well presented in the CR Form	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Timeliness of response from Reviewers is generally appropriate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The response from Reviewers is generally clear	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Training and support provided on how to submit Change Request is adequate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

SECTION 3: REVIEWING/APPROVING CHANGE REQUESTS

1. Have you ever reviewed any Change Requests?

☐ YES

☐ NO (if no, go to section 4)

2. Please indicate the extent to which you agree (disagree) with each of the following aspects when you reviewed the proposed change

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree	Not Sure
Issue/problem described in the CR Form is clear	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Issue/problem described in the CR Form is complete	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The identification of rationale to make the change is sufficient	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
There is adequate information in CR form to state the impacts of the proposed change if the change will not be introduced	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Information for the impacted areas of the proposed change is generally sufficient	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The impact analysis information is helpful in guiding you to approve or reject the proposed change	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The impact analysis information is beneficial in estimating work required	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The CR process makes it easy to resolve conflicts if Reviewers disagree in their review results	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

3. How would you rate the following aspects when you decide to reject or approve the proposed change?

	Excellent	Very Good	Good	Fair	Poor	Not Sure
The CR information to be used in assessing the impacted areas of the proposed change is	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The reasonableness of the proposed change stated in CR form is	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Your confidence in reviewing the feasibility of the proposed change based on the supported information in the current CR form is	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

SECTION 4: IMPLEMENTATION OF APPROVED CHANGES

1. Have you ever been involved in the implementation of approved Change Requests?

☐ YES

☐ NO (if no, go to section 5)

2. Please indicate the extent to which you agree (disagree) with each of the following aspects when implemented the approved change.

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree	Not Sure
The written information in the approved CR form is clear	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The information stated in the approved CR form is sufficient for change implementation	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The communication with other impacted parties about the change implementation is effective	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

SECTION 5: VERIFICATION OF IMPLEMENTED CHANGE

1. Have you been involved in the verification process of Change Requests implementation?

☐ YES (if yes, please answer the following question)
☐ NO (end of survey)

2. How would you rate the easiness of tracking the approved and implemented changes based on the information available in the CR form?

☐ Very Poor ☐ Poor ☐ Fairly Good ☐ Good ☐ Very Good ☐ Excellent

SECTION 6: ADDITIONAL COMMENTS

Please add any additional comments you may have on the current Change Control process and Change Request Form

Thank You for completing this survey

Appendix H: Pre-Survey Data Table

Data Table

Form Number				Reporting of problem/issue	Evaluation of Change Requests	Performing change impact analysis	Track work in process changes	Perform Change Request Review meeting	Approval of Change Requests	Develop tasks for change implementation	Verify change against product deliverables	Have you submitted a Change Request?	The current Change Request (CR) form is easy to use	The current CR form is easy to understand	The current CR form covers all aspects related to a requested change
1	Management	2	1	5	4	3	3	2	5		4	1	2	2	4
2	Engineer/Developer	3	2	4	3	3	2	2	4	4	3	1	2	2	3
3	Engineer/Developer	2	1	3	5	5	4	3	5	3	2	1	2	3	3
4	Engineer/Developer	3	1	3	2	2	2	2	3	4	4	2			
5	Engineer/Developer	3	2	3	5	5	4	3	3	4	3	1	2	3	2
6	BE/Tester	4	2	4	4	5	3	2	3	4	4	2			
7	Engineer/Developer	3	2	3	4	4	3	2	4	4	4	1	2	3	4
8	Engineer/Developer	3	1	4	3	3	3	4	3	4	3	1	3	3	4
9	Engineer/Developer	3	1	3	5	5	5	3	4	3	3	1	5	5	4
10	Engineer/Developer	4	2	4	3	4	3	4	4	4	5	1	3	3	2
11	Engineer/Developer	3	1	4	4	5	3	3	4	3	4	1	2	3	4
12	Engineer/Developer	4	1	5	3	2	2	1	4	3	4	2			
13	Engineer/Developer	3	2	5	2	2	4	2	4	3	3	1	2	3	3
14	Engineer/Developer	4	1	4	4		4	3		4	4	1	3	2	2
15	BE/Tester	2	1	4	4	4	4	3	4	4	4	1	3	4	4
16	Management	1	1	5	5	5	5	4	5	5	4	1	2	3	4
17	Engineer/Developer	3	1	4	2	3	2	5	5	5	5	1	2	2	2
18	SQA team	1	1	5	5	5	5	5	5	5	5	1	na	2	na
19	Engineer/Developer	3	2	4	4	4	3	4	3	4	4	1	2	2	2
20	Engineer/Developer	3	2	2	4	5	4	2	2	5	4	1	4	3	3
21	Management	4	1	4	4	4	4	3	5	5	5	1	3	3	4
22	Management	3	1	3	5	5	4	3	4	3	4	1	2	2	2
23	Management	1	1	3	5	5	3	2	5	4	4	1	3	3	4
24	Management	2	1	4	4	5	4	3	4	4	4	1	2	2	3
25	Engineer/Developer	4	3	3	5	5	5	5	5	5	5	1	2	2	na
26	SQA team	5	1	5	3	5	4	3	3	5	5	2			

(Continuing table)

Form Number	Information is sufficiently well presented in the CR form	Timeliness of response from Reviewers is generally appropriate	The response from Reviewers is generally clear	Training and support provided on how to submit CR is adequate	Have you reviewed Change Requests?	Issue/problem described in the CR form is clear	Issue/problem of rationale to make the change is sufficient	The identification of rationale to make the change is sufficient	There is adequate information in CR form to state the impacts of the proposed change if the change	Information for the impacted areas of the proposed change is generally sufficient	The impact analysis information is helpful in guiding you to approve or reject the proposed change
1	3	2	3	3	1	3	3	2	4	2	1
2	3	4	3	3	2						
3	3	5	2	4	2						
4					1	2	4	4	3	4	4
5	2	4	4	3	1	2	2	2	2	3	2
6					1	2	3	3	4	4	2
7	3	2	3	2	1	2	2	2	4	3	2
8	3	4	2	3	1	4	3	3	4	3	3
9	4	2	2	4	1	2	2	2	4	4	3
10	2	3	3	3	2						
11	4	3	3	4	1	4	4	3	4	2	3
12					1	3	3	3	3	3	3
13	2	4	4	2	1	2	2	2	3	3	3
14	3	4	3	4	1	3	3	4	4	4	3
15	5	3	4	5	1	3	5	4	5	3	1
16	2	3	2	3	1	2	2	2	3	2	4
17	2	3	3	4	1	2	2	2	3	4	4
18	na	na	5	na	1	5	5	5	na	5	na
19	2	3	3	3	1	3	3	2	2	3	3
20	3	2	2	4	1	2	3	2	4	4	3
21	3	4	2	5	1	3	na	2	4	3	2
22	3	4	2	2	1	2	2	3	4	2	2
23	3	4	4	5	1	2	3	3	2	4	2
24	3	4	3	4	1	3	3	3	2	3	2
25	2	4	4	3	2						
26					2						

[Continuing table]

Form Number	The impact analysis information is beneficial is estimating work required	The CR process makes it easy to solve conflicts if Reviewers disagree in their review results	The CR information to be used in assessing the impacted areas of the proposed change is	The reasonableness of the proposed change stated in CR form is	Your confidence in reviewing the feasibility of the proposed change based on the supported informa	CR Implementation	The written information in the approved CR form is clear	The information stated in the approved CR form is sufficient for change implementation	The communication with other impacted parties about the change implementation is effective	Have you been involved in CR verification?	The easiness of tracking the approved and implemented changes (verification)
1	1	4	2	2	2	1	4	4	3	1	6
2						1	2	4	4	2	
3						1	2	2	2	2	
4	3	4	5	5	4	1	4	3	3	2	
5	2	na	3	3	na	2				2	
6	3	3	4	4	na	2				1	5
7	3	2	3	3	3	1	2	2	4	2	
8	3	3	4	5	4	1	3	4	3	2	
9	3	3	4	4	3	2				2	
10						1	3	3	3	2	
11	4	3	2	3	3	2				2	
12	3	3	na	na	na	1	3	3	3	2	
13	3	4	3	3	3	1	3	3	3	2	
14	2	4	2	2	2	1	2	2	3	2	
15	na	4	3	5	4	2				1	5
16	2	3	4	4	4	1	2	3	4	1	5
17	2	2	3	3	3	1	2	2	2	2	
18	na	5	3	5	5	1	4	4	4	1	3
19	2	2	2	3	3	1	2	2	2	2	
20	3	3	3	3	3	1	3	3	3	2	3
21	4	4	4	3	3	2				2	5
22	2	4	2	3	3	1	2	2	4	1	4
23	4	4	4	3	3	1	2	3	4	1	5
24	2	2	4	4	4	1	3	3	3	1	5
25						1	3	4	na	2	
26										1	6

[Continuing table]

Form Number	Additional comments	Participants' Code	The CR information to be used in assessing the impacted areas of the proposed change is	The reasonableness of the proposed change stated in CR form is	Your confidence in reviewing the feasibility of the proposed change based on the supported information
1	It really is up to the person filling in the CR whether it is a good 'experience' or not in terms of change detail, rationale, etc. The form could be better in terms of impact section and implementation details	InRelmVe			
2		InIm			
3	<p>The existing C process is manual and in the past, CRs may be in the CM repository, but no action was performed on the CR. This could do with a automatic notification when changes do occur to the CR or when a CR gets created.</p> <p>It's not clear who types of reviewers are from the CC Process. I assumed that it included BI but this is not made clear in the process. Eg. for what artifact, who the targeted reviewers are. Only then can I anticipate how long the review will take, Sometimes, it takes forever which discourages the use of CRs.</p> <p>It's not clear who is responsible for filling up each section in the CR at which point in time. This type of information is best provided in a work instruction, but no work instruction exists for CRs</p>	InIm			
4		Relm			
5		InRe			
6	Well I know, even though I've been assigned to write tests to cover a lot of recent CR's, I have only seen 1 task in my time sheet/task list to do with a CR (and it ended up being deferred). All other CR's I reviewed are probably most likely already in the build, and I haven't been allocated time to cover any of them	ReVe			
7		InRelm			
8	<p>The implementation of the CR by all affected parties is not coordinated. This can lead to the product being 'broken'. CR process should target a build number.</p> <p>SILs are not always raised for each affected party. They should be and they should be linked and documented in the CR so everyone knows what and when changes went in for each affected party.</p> <p>I can guarantee you that if you wanted to find all the changes that went into the code base and document revisions for a CR, you would never be able to tell.</p> <p>Some CRs that are implemented late in the Project, nullify other CRs that have gone in previously (by magic). This does not seem to be tracked.</p>	InRelm			

[Continuing table]

Form Number	Additional comments	Participants' Code	The CR information to be used in assessing the impacted areas of the proposed change is	The reasonableness of the proposed change stated in CR form is	Your confidence in reviewing the feasibility of the proposed change based on the supported information
9	We need tools / dialogs not forms	InRe			
10		InIm			
11		InRe			
12		Relm			
13	It all very much depends on the person writing the change request as to whether the change is easy to understand or whether sufficient information is supplied. You can formalize a process as much as you like, but a formal process is never going to replace a person's ability to clearly articulate an idea. As well as a format change request process, engineers need training in writing technical documents in a clear and concise manner.	InRelm			
14		InRelm			
15		InReVe			
16	The questionnaire does not ask about the information regarding the change beyond that which is included in the CR form. The CR form currently does not provide the project managers with sufficient detail to plan and schedule the changes. A clearer indication of the impact will be more informative.	InRelmVe			
17		InRelm			
18		InRelmVe			
19		InRelm			
20	CR takes too long to be approved. MLs/TLs are sometimes unclear about the CR process and allow implementation to proceed even before the CR is approved (especially when under pressure to get work done). The use of CR to cover changes in design is inconsistent. e.g. There have been many situations in 'Project XENGN' where changes in the its product design causes changes in the installation. However, NO CR is raised since it takes too long to process. CR process needs to be streamlined - needs faster approval.	InRelm			
21		InRe			
22	Variability in CR quality based on initiator. Best are very good easy to evaluate, worst leave a lot to imagination	InRelmVe			
23	Looking forward to new process and supporting tools	InRelmVe			
24		InRelmVe			

[Continuing table]

Form Number	Additional comments	Participants' Code	The CR information to be used in assessing the impacted areas of the proposed change is	The reasonableness of the proposed change stated in CR form is	Your confidence in reviewing the feasibility of the proposed change based on the supported information
25	I have had limited exposure the change request process and only for simple cases. It was sufficient in these cases	InIm			
26		Ve			

Appendix I: Project Release_B Datasheet

Change ID	Change Description	Change types	Reason Category	Sources of change	Req Levels	Nof Lev3	Req Levels	Nof Lev4	Total Reqs	Initiate date	Approved date
CR_006	the requirement overlaps and is covered by other requirement	del	Redundant requirement	Requirements Review	Lev3	2			2	9-Apr-03	21-May-03
CR_006	changes to requirements text for clarification and add more information	mod	Clarification	Requirements Review	Lev3	8			8	9-Apr-03	21-May-03
CR_007	changes to requirements text to remove unnecessary information such as 'i.e.'	mod	Clarification	Requirements Review	Lev3	1			1	14-May-03	22-May-03
CR_008	add more information into the text of req for clarity	mod	Clarification	Requirements Review	Lev3	3			3	14-May-03	22-May-03
	add new reqs	add	Missing requirement	Requirements Review	Lev3	1			1	14-May-03	22-May-03
CR_009	requirements statement need to be amended for more details information and precise	mod	Clarification	Requirements Review	Lev3	2			2	19-May-03	22-May-03
9	the functionality placed unnecessary constraint on the user	del	Implied unnecessary constraint	Requirements Review	Lev3	1			1	19-May-03	22-May-03
CR_010	this functionality is beyond the scope what should be implementing in this release. This functionality is not available in the previous rel	del	Scope change	Eng & Management discussion			Lev4	1	1	2-Jun-03	4-Jun-03
CR_011	the requirement is already covered by a test scenario of other requirement	del	Redundant requirement	FS Review			Lev4	1	1	4-Jun-03	4-Jun-03
CR_012	to envisage a migration from the previous releases to the current release and include better application security, in which this functionality is not available in the previous releases	mod	Functionality enhancement	Engineering analysis	Lev3	3	Lev4	4	7	20-Jun-03	3-Jul-03
CR_012	to envisage a migration from the previous releases to the current release and include better application security, in which this functionality is not available in the previous releases	add	Functionality enhancement	Engineering analysis			Lev4	1	1	20-Jun-03	3-Jul-03
CR_012	to envisage a migration from the previous releases to the current release and include better application security, in which this functionality is not available in the previous releases	del	Redundant requirement	Engineering analysis			Lev4	6	6	20-Jun-03	3-Jul-03
CR_018	requirements cannot be fulfill due to 3rd party CM tool does not provide capability	del	3rd party software/tool influence	FS Review			Lev4	1	1	7-Jul-03	8-Aug-03
CR_018	reqs are deleted due to certain sub-functionality interferes with integrity management functionality	del	Resolving conflict	FS Review			Lev4	3	3	7-Jul-03	8-Aug-03
CR_027	Additional Requirement Mapping for Builder Deployment	mod	Traceability	FS Review			Lev4	2	2	14-Aug-03	21-Aug-03
CR_025	this functionality is problematic and likely beyond the scope of what could be achieved in this release and it's not available in the previous release	del	Scope change	Technical team discussion	Lev3	1	Lev4	3	4	29-Jul-03	18-Aug-03

Change ID	Change Description	Change types	Reason Category	Sources of change	Req Levels	Nof Lev3	Req Levels	Nof Lev4	Total Reqs	Initiate date	Approved date
CR_034	change requirements text to address the print functionality in the new environment	mod	Clarification	Technical team discussion			Lev4	2	2	14-Aug-03	10-Sep-03
CR_034	It's triggered by changing requirements text above	del	Redundant requirement	Technical team discussion			Lev4	2	2	14-Aug-03	10-Sep-03
CR_035	it's not captured in the product definition. PM: no one knows in the beginning that we need these requirements	add	Missing requirement	DS review			Lev4	13	13	15-Aug-03	12-Sep-03
CR_038	View/Diff/Merge functionality changed, need to reword existing req	mod	3rd party software/tool influence	DS review			Lev4	1	1	28-Aug-03	23-Oct-03
CR_038	New functionality (Report Difference and Merge are invoked, need to add new req	add	3rd party software/tool influence	DS review			Lev4	1	1	28-Aug-03	23-Oct-03
CR_044	As a result of reducing the scope of this release, these requirements were deferred	del	Scope change	Eng & Management discussion	Lev3	1	Lev4	3	4	18-Sep-03	16-Oct-03
CR_046	Remove Version Control "may" requirements from NGEN 1.0 Project that are deferred	del	Scope change	FS Review			Lev4	7	7	16-Sep-03	3-Oct-03
CR_047	The NLS functionality is not currently supported and is not required for safe passage. It's beyond the scope of what can be achieved in this release	del	Scope change	FS review			Lev4	2	2	19-Sep-03	29-Oct-03
CR_052	changes the priority of MDA studio from an optional feature to be a default feature	mod	Product strategy	Project Management	Lev3	1			1	15-Dec	19-Dec-03
CR_054	A functionality (VS menu) has no supported functionality available and it requires a lot of Model (sub system) work to fulfill it	del	3rd party software/tool influence	DS review			Lev4	1	1	14-Oct-03	23-Oct-03
CR_054	A functionality (VS menu) has no supported functionality available and it requires a lot of Model (sub system) work to fulfill it. Remove the 3rd sw party names from reqs text	mod	clarification	DS review			Lev4	18	18	14-Oct-03	23-Oct-03
CR_055	Defer reqs relate to animation, breakpoints, and OLTP (some functionalities are available in the previous rel)	del	Scope change	Eng + Marketing group			Lev4	11	11	14-Oct-03	17-Oct-03
CR_056	Requirement statement has too implementation specific about webform that will not be developed	mod	clarification	DS review			Lev4	1	1	14-Oct-03	20-Oct-03
CR_058	there is no applicable requirement to fulfill 'Painter' functionality (high level req)	add	Missing requirement	DS review			Lev4	1	1	28-Oct-03	31-Oct-03
CR_059	there is no applicable requirement to fulfill Export/import utilities	add	Missing requirement	DS review			Lev4	1	1	3-Oct-03	10-Nov-03
CR_060	Scripts installation	add	Product strategy	Marketing group	Lev3	1	Lev4	3	4	29-Mar-04	28-Apr-04
CR_061	there is inconsistency between the text of requirements and the text of test scenario. The test scenario is correct, change the word 'list' to 'set' without loss of functionality	mod	Testability	Detailed design analysis			Lev4	1	1	3-Oct-03	13-Nov-03

Change ID	Change Description	Change types	Reason Category	Sources of change	Req Levels	Nof Lev3	Req Levels	Nof Lev4	Total Regs	Initiate date	Approved date
CR_071	in the packaging plan, a requirement has been placed to merge client tool with developer- need to merge the client tool installation package under the developer installation package	add	Product strategy	Project Management			Lev4	13	13	8-Dec-03	29-Jan-04
CR_086	The SQL Editor functionality does not support the functions stated in the requiement	del	3rd party software/tool influence	Detailed design analysis			Lev4	1	1	11-Mar-04	15-Mar-04
CR_091	need to address high level requirements - 'evaluation copy of developer' functionality.	add	Missing requiremment	Detailed design analysis	Lev3	1	Lev4	11	12	22-Mar-04	29-Mar-04
CR_094	to implement this requirement will require lots of effort due to design and code complexity, the risk is very high to finish in the time frame	del	Design complexity	detailed design analysis			Lev4	1	1	5-Apr-04	8-Apr-04
CR_096	this requirement about a creation and initialisation a user's database, which is not useful for the customer, possiblity of functionality conflicts; need further design thought, and beyond customer expectation	del	Resolving conflict	detailed design analysis			Lev4	1	1	6-Apr-04	13-Apr-04
CN_16	there is no requirement available to cover the installation of RM feature	add	Missing requiremment	Project Management			Lev4	1	1	13-Feb-04	13-Feb-04
CN_03	Changing the heading of the requirement that was inaccurate (changing 'checker' to 'maintained')	mod	Clarification	Detailed design analysis	Lev3	1.00			1	9-Jan-04	9-Jan-04
CR_112	Requirements Modeller Integration with NGEN Deployment	add	Functional integration	Product Integration Test			Lev4	3	3	11-Jun-04	22-Jun-04

Appendix J: Survey Questionnaire on a new Change Request Process

CHANGE REQUEST PROCESS (CRP) EVALUATION

SURVEY QUESTIONNAIRE

About The Survey

Recently, an updated version of the Change Control Process has been introduced at the Australian Centre for Unisys Software (ACUS) as part of process improvement initiatives. In order to identify the limitations and benefits of the new implemented Change Request (CR) process for future improvement, it is necessary to conduct a post-survey on the usage of the CR process.

- The information you provide will help the company to evaluate its change process. It will also contribute to the research being carried out by UTS researchers at ACUS.
- It will take approximately 10 minutes of your time to complete the survey.
- There is no right or wrong answer; we are seeking your opinions and perceptions about the new Change Request Process. Your opinion is important to us.
- Remember **THE SURVEY IS STRICTLY CONFIDENTIAL.** All responses will be aggregated; no individual will be identifiable from the aggregated responses.
- When you have finished, please send the completed survey by 16 September 2005 to Nur at nur@it.uts.edu.au

Thank you for your assistance.

Section 1: THE CHANGE REQUEST (CR) PROCESS

1. Please indicate your current Functional Area (i.e. Project Manager, Engineer, Eng Manager, Tester, PI, etc):

2. How would you rate your understanding of the new CR Process?

☐ Excellent ☐ Very good ☐ Good ☐ Fair ☐ Poor ☐ Not sure

3. To what extent is the CR process important for you in your work?

☐ Very important ☐ somewhat important ☐ not important ☐ Not sure

4. Please indicate how important is each of the following items relating to the Change Request Process (select the appropriate checkbox for each item)

Change Request Process is useful for	Lowest	Low	Medium	High	Highest	NA
Reporting of problem/issue	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Evaluation of Change Requests	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Performing change impact analysis	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Tracking change progress	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Perform Change Request Review meeting	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Develop tasks or work requests for change implementation	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Verify change against product deliverables	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

5. The following questions are related to Training on CR Process and Change Synergy Tool. Please indicate the extent to which you agree or disagree with the following aspects of the training (select the appropriate checkbox for each item)

Training	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree	Not Sure
*Training on the overall view of CR process is adequate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
*Training on the use of Change Synergy tool is sufficient	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Training on how to submit Change Request is adequate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
*I gained a better understanding of the CR process and its tool through the training session	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

SECTION 2: USING THE CHANGE REQUEST FORM TO SUBMIT A CHANGE

1. Have you ever initiated or submitted a Change Request?
☐ YES
☐ NO (please go to section 3)
2. Please indicate the extent to which you agree (disagree) with each of the aspects of CR process and CR form

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree	Not Sure
The current Change Request (CR) Form is easy to use	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The current CR form is easy to understand	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
*The inclusion of CR type(i.e. type A, B, and C) is useful for identifying and analysing change impacts	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
*The inclusion of CR type(i.e. type A, B, and C) is useful for choosing the appropriate reviewers	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Timeliness of response from reviewers is generally appropriate	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The response or information from reviewers is generally clear	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

SECTION 3: REVIEWING/APPROVING CHANGE REQUESTS

1. Have you ever reviewed any Change Requests?
☐ YES
☐ NO (please go to section 4)
2. Please indicate the extent to which you agree (disagree) with each of the following aspects when you reviewed the proposed change

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree	Not Sure
Issue/problem described in the CR Form is usually clear	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Issue/problem described in the CR Form is usually complete	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The identification of rationale to make the change is usually sufficient	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
There is adequate information in CR form to state the impacts of the proposed change if the change will not be introduced	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Information for the impacted areas of						

the proposed change is generally sufficient	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The impact analysis information is helpful in guiding you to approve or reject the proposed change	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The impact analysis information is beneficial in estimating work required	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The CR process usually makes it easy to resolve conflicts if Reviewers disagree in their review results	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

3. How would you rate the following aspects when you decide to reject or approve the proposed change?

	Excellent	Very Good	Good	Fair	Poor	Not Sure
The CR information to be used in assessing the impacted areas of the proposed change is	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The reasonableness of the proposed change stated in CR form is	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Your confidence in reviewing the feasibility of the proposed change based on the supported information in the current CR form is	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

SECTION 4: IMPLEMENTATION OF APPROVED CHANGES

1. Have you ever been involved in the implementation of approved Change Requests?

☐ YES
☐ NO (please go to section 5)

2. Please indicate the extent to which you agree (disagree) with each of the following aspects when implementing the approved change.

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree	Not Sure
The written information in the approved CR form is clear	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The information stated in the approved CR form is sufficient for change implementation	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The communication with other impacted parties about the change implementation is effective	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

SECTION 5: VERIFICATION OF IMPLEMENTED CHANGES

1. Have you been involved in the verification process of Change Requests implementation?

- ☐ YES
☐ NO (please go to section 6)

2. How would you rate the easiness of tracking the approved and implemented changes based on the information available in the CR form?

☐Very Poor ☐Poor ☐Fairly Good ☐Good ☐Very Good ☐Excellent

***SECTION 6: CHANGE SYNERGY (CS) TOOL**

1. Please indicate the extent to which you agree (disagree) with each of the tool aspects.

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree	Not Sure
The CS Tool is easy to use	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Using the tool to process a change request saves time	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Using the tool improves schedule performance for implementing changes	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

***SECTION 7: OVERALL SATISFACTION**

1. Overall, you feel that the CR process meets your needs for requesting and processing a change.

☐Strongly agree ☐Agree ☐Neutral ☐Disagree ☐Strongly disagree ☐NA

2. Overall, are you satisfied with the CR Process and Change Synergy Tool?

☐Never ☐Rarely ☐Sometimes ☐Very often ☐Always

SECTION 8: ADDITIONAL COMMENTS

Please add any additional comments you may have on the current Change Control process, Change Request Form, or Change Synergy Tool:

Thank You for completing this survey

Appendix K: Post-Survey Data Table

Data Table

Form Number				Reporting of problem/issue	Evaluation of Change Requests	Performing change impact analysis	Tracking change process	Perform Change Request Review meeting	Develop tasks or work requests for change implementation	Verify change against product deliverables	Training on the overall view of CR process is adequate	Training on the use of Change Synergy tool is sufficient
1	engineer	3	2	3	5	3	3	4	4	3	2	2
2	engineer	3	1	2	4	4	4	4	3	4	3	3
3	engineer	4	2	2	5	2	3	3	3	3	4	4
4	Project manager	2	1	4	4	4	4	3	5	3	2	4
5	engineering manager	2	1	3	5	4	4	2	3	3	2	2
6	engineer	4	1	5	5	1	1	2	4	na	4	4
7	engineer	3	2	4	5	5	4	3	5	5	2	2
8	Architect	3	1	4	4	4	4	3	3	3	na	2
9	engineering manager	2	1	2	4	4	2	4	3	2	3	2
10	engineer	2	1	4	4	2	3	1	3	2	3	3
11	engineer	4	1	3	3	3	3	3	3	3	3	2
12	tester	3	2	2	5	5	3	3	4	3	3	4
13	engineer	4	2	na	3	3	3	3	3	3	3	4
14	PI/Education	5	1	na	5	3	5	4	5	4	5	5
15	Architect	3	1	3	4	4	3	4	4	4	3	2
16	engineer	3	1	na	5	5	5	5	5	5	2	3
17	engineering manager	2	1	3	5	4	4	3	3	3	2	3
18	engineer	3	1	1	3	3	1	na	1	1	3	4
19	engineer	5	2	2	4	4	4	4	4	4	4	4
20	engineer	3	2	2	3	4	3	2	4	4	3	3
21	engineer	3	1	4	4	2	4	3	4	4	3	3
22	PI/Education	4	1	3	4	4	5	5	5	5	5	4
23	Director	2	1	4	4	4	4	3	4	3	2	2
24	engineer	4	1	1	4	1	1	1	1	5	5	4
25	engineer	5	3	3	5	5	5	5	5	3	5	5
26	engineer	4	2	1	4	5	5	3	2	2	4	3
27	Project manager	2	1	3	5	5	5	5	4	5	4	4

[Continuing table]

Form Number	Training on how to submit Change Request is adequate	I gained a better understanding of the CR process and its tool through the training session	Have you submitted a Change Request?	The current Change Request (CR) form is easy to use	The current CR form is easy to understand	Timeliness of response from Reviewers is generally appropriate	The response from Reviewers is generally clear	The inclusion of CR types is useful for identifying and analysing change impacts	The inclusion of CR types is useful for choosing the appropriate reviewers	Have you reviewed Change Requests?
1	2	3	1	3	3	4	3	2	3	2
2	2	2	1	2	2	4	4	2	2	1
3	4	3	1	2	2	3	2	3	2	1
4	3	3	1	2	2	3	2	4	2	1
5	2	3	2							1
6	4	4	1	4	4	2	3	3	4	1
7	3	2	2							2
8	2	na	1	4	3	2	2	4	2	1
9	2	2	2							1
10	3	4	1	3	3	5	4	2	2	1
11	3	3	1	3	3	2	2	2	2	1
12	4	3	2							1
13	3	3	2							2
14	5	na	2							1
15	3	2	2							1
16	3	3	2							1
17	2	2	2							1
18	3	3	1	4	4	5	3	3	4	1
19	4	4	1	2	2	2	2	4	3	2
20	2	2	1	4	4	3	2	3	3	1
21	3	4	1	4	4	3	3	2	2	2
22	2	4	2							1
23	2	2	2							1
24	5	5	1	5	5	3	5	5	5	1
25	5	5	1	4	4	2	na	na	na	2
26	3	2	1	1	2	1	2	2	1	2
27	4	4	1	2	2	2	2	1	1	1

[Continuing table]

Form Number	Issue/problem described in the CR form is usually clear	Issue/problem described in the CR form is usually complete	The identification of rationale to make the change is sufficient	There is adequate information in CR form to state the impacts of the proposed change if the change	Information for the impacted areas of the proposed change is generally sufficient	The impact analysis information is helpful in guiding you to approve or reject the proposed change	The impact analysis information is beneficial is estimating work required	The CR process usually makes it easy to solve conflicts if Reviewers disagree in their review resu
1								
2	3	4	3	3	3	2	2	4
3	3	4	3	4	2	2	2	4
4	2	2	3	3	2	2	1	2
5	2	2	2	3	4	2	2	2
6	2	2	2	5	2	2	4	3
7								
8	5	5	5	5	2	na	na	5
9	2	2	3	4	2	2	3	2
10	2	3	3	5	4	3	5	4
11	2	2	2	2	2	2	2	2
12	2	2	1	3	2	2	2	4
13								
14	5	5	5	5	5	5	5	na
15	3	2	2	2	3	2	2	3
16	4	4	4	3	4	3	3	3
17	2	2	2	2	2	2	2	2
18	2	2	2	2	2	2	3	4
19								
20	2	2	2	3	3	2	3	3
21								
22	2	3	5	2	4	2	4	3
23	2	2	2	2	2	2	2	3
24	5	5	5	5	4	4	4	4
25								
26								
27	2	2	2	3	2	2	1	2

[Continuing table]

Form Number	The CR information to be used in assessing the impacted areas of the proposed change is	The reasonableness of the proposed change stated in CR form is	Your confidence in reviewing the feasibility of the proposed change based on the supported informa	Have you involved in the change implementation?	The written information in the approved CR form is clear	The information stated in the approved CR form is sufficient for change implementation	The communication with other impacted parties about the change implementation is effective	Have you involved in CR verification?	The easiness of tracking the changes made to the project artefacts
1				1	2	3	3	2	
2	4	3	3	1	3	3	2	2	
3	na	3	4	2				2	
4	2	2	2	2				2	
5	3	3	2	1	2	2	2	1	3
6	2	2	2	1	2	2	3	2	
7				2				1	3
8	4	4	4	2				2	
9	2	2	3	2				2	
10	4	4	4	1	3	3	3	2	
11	3	3	3	1	2	2	2	2	
12	2	2	2	1	1	2	2	2	
13				2				2	
14	na	na	na	1	5	5	4	2	
15	3	3	3	2				2	
16	4	4	4	2				2	
17	3	2	2	1	2	2	2	1	4
18	3	3	3	1	2	2	5	2	
19				1	2	2	3	2	
20	3	3	3	1	2	2	2	1	4
21				1	3	3	2	2	
22	4	3	5	1	4	4	4	2	
23	2	2	2	2				2	
24	5	4	4	2				2	
25				2				2	
26				1	4	3	3	2	
27	2	3	3	1	2	2	1	2	

[Continuing table]

Form Number	Change Synergy (CS) Tool is easy to use	Using the tool to process a change request saves time	Using the tool improves schedule performance for implementing changes	Overall, the CCP meets your needs	Overall, you are satisfied with the CCP and CS Tool
1	3	3	3	2	3
2	2	3	2	2	3
3	2	3	3	3	3
4	4	2	2	2	4
5	2	2	3	2	2
6	2	4	5	4	2
7	2	2	2	2	2
8	3	2	2	2	3
9	2	2	2	2	2
10	3	4	5	3	3
11	3	3	3	3	3
12	2	3	3	2	3
13	3	3	3	3	3
14	5	5	na	5	4
15	3	3	3	3	3
16	2	2	3	2	2
17	2	3	2	2	3
18	5	4	5	4	2
19	3	2	2	2	3
20	4	3	2	2	3
21	2	2	3	2	3
22	2	4	3	4	3
23	2	2	2	2	4
24	4	4	4	5	4
25	2	2	2	3	3
26	2	2	3	2	4
27	1	1	1	1	1

[Continuing table]

Form Number	Additional comments
1	the process needs to be streamlined for type C changes. It takes too long for example, to update a design for an approved section of the SDD. The process can also be delayed if key people are on leave at review time
2	<p>If when reviewing a CR, I had issues that need to be addressed, I have found it unclear how to communicate these issues (via Change Synergy) and how these issues get resolved and updated using tool.</p> <p>Also the automated notification emails are confusing - it is not very apparent from the email what action is required as a result of getting a notification email</p>
3	<p>Needs to be supplemented with conflict resolution process - we use the AF process. Not nearly enough technical detail goes into CRs alone to make a reasonable decision - Usually, CR form needs to be supplemented by supporting documentation and review processes (in practice, usually informal ones - sometimes if you are really lucky, you get an attachment) outside of the CR process, which does not usually get documented in the eventual CR. Technical reviewers tend to have a good understanding of the issues before reviewing the CR, but mainly through informal processes, and this understanding is not captured on the CR - most of the time I already know what the problem is, but don't record how we got there. Thus, assumptions, terminology and context of the CR is usually not recorded either, making it difficult to pass the CR to another engineer without a detailed TOI from initiator. This TOI should probably be derived from another existing process, and used to supplement the CR implementation process. Also not sure how the CRs link back to our originating marketing requirements - I guess most of the time, the CR is raised because the requirements are either ambiguous, missing or misinterpreted by us, so it'd be hard to pin it on a requirement in the first place. Overall, the CR process is a definite step forward, but there needs to be a recognition of its process environment to know where we came from and where we go to.</p>
4	<p>1- The CR Type values (A, B or C) suggest priority or importance, not type.</p> <p>2- There are issues: a) The tool is mostly OK to approve the CR, but the transferral from CS to the schedule is still an issue. b) The "work product" field is not being filled with the correct data, it mostly says "source code" without specifying whether it's R/T or Developer, or Model or what. c) It is impossible in some cases to just add a comment to the CR. I'll leave it at that for the moment.</p>
5	<p>When describing impacts, authors do this in different ways. Some put in a code, document updates and testing for one component in each impact. Other people put every change to every artifact as a impact. It is easier to schedule if each impact is a component code change or component document change, test, pi, education.</p> <p>It is confusing to people whether a CPP is required for the document updates or if they should use exclusive purpose.</p> <p>All in all, we are getting better at this, but with some additional improvements and up to date training we should get there.</p>
6	
7	

[Continuing table]

Form Number	Additional comments
8	<p>Tool: Not easy to search on a keyword of the CRs. Have trouble filtering out CPPs/CPs from CRs and vice versa.</p> <p>Process: The classifications of A, B and C need to be clearly identified in the Form or provide Help in the form for these classifications. Next, The criteria for creating a Change Request is unclear.</p>
9	
10	
11	
12	
13	
14	
15	
16	<p>The process itself is good. Usually the it is more the content of the change request that could be improved. I also find that a change request comes to me to approve and then there is pressure to approve it quickly even when it may not be complete. This may lead to CRs getting approved when they should not.</p> <p>The Change Synergy Tool is also good. The form takes a bit of getting used too. The placement of the attachments field below the Approval section is not the best. The attachments are part of the materials to be approved and so I feel they should be closer to the change request summary information.</p>
17	
18	<p>I dislike the Change Synergy tool. The interface isn't very user friendly. I've also had problems transitioning it back a state to update the description after a reviewer made a comment.</p> <p>For changes that are required to documentation, this process adds extra time. The standard time to allow for someone to review a CR is a week (although in practice this is usually much longer). Once it's approved, you then have to make the document changes, and get them reviewed which takes at least another week. So to update a baselined document, it takes minimum 2 weeks (usually a lot longer). It would be good if it could send out an email every few days to the people who have yet to review the CR.</p> <p>Our process seems to be written centered around the tool rather than the other way around. A good process is tool independent.</p>
19	
20	
21	
22	
23	
24	
25	

[Continuing table]

Form Number	Additional comments
26	It would be useful if some information like how to classify a CR, or how to select a mode was available with the CS tool so that one can get the CR raised correctly in the first attempt. Most of the times while raising a new CR one gets some or the other information incorrect which leads to cycles of CR updates between the initiator and CR administrators.
27	

[Continuing table]

Form Number	Participants' Code
1	InReIm
2	InReIm
3	InRe
4	InRe
5	ReImVer
6	InReIm
7	Ver
8	InRe
9	Re
10	InReIm
11	InReIm
12	ReIm
13	
14	ReIm
15	Re
16	Re
17	ReImVer
18	InReIm
19	InIm
20	InReImVer

[Continuing table]

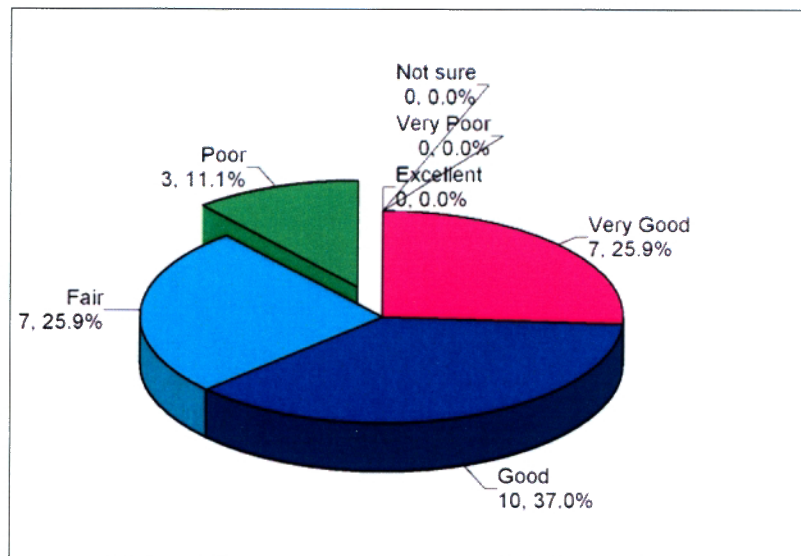
Form Number	Participants' Code
21	InIm
22	ReIm
23	Re
24	InRe
25	In
26	InIm
27	InReIm

Appendix L: Detailed Summary of Post-Survey Results

II. Findings: Post-Survey Questionnaire (extracted from the questionnaire)

1. General Overview of Change Control Process (CCP)

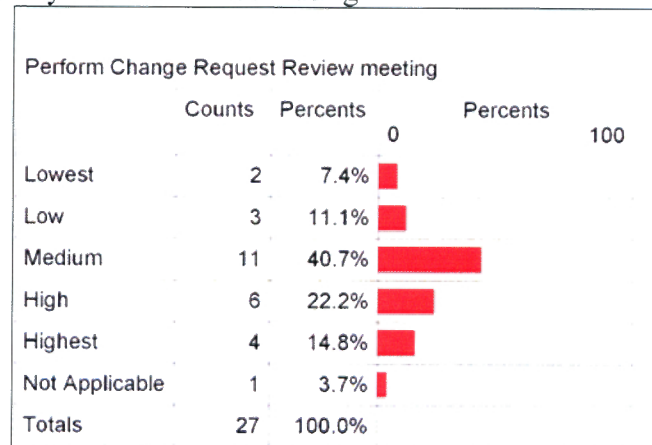
a. Participant's understanding on the current CCP



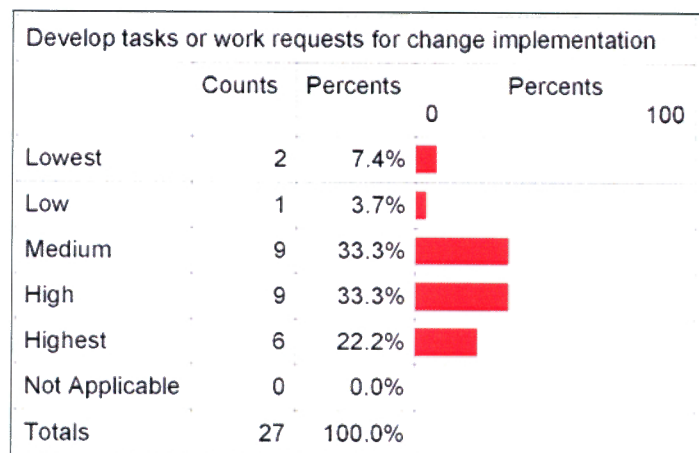
b. The priority of change impact analysis

Performing change impact analysis				
	Counts	Percents	Percents	
			0	100
Lowest	2	7.4%	<div></div>	
Low	3	11.1%	<div></div>	
Medium	5	18.5%	<div></div>	
High	11	40.7%	<div></div>	
Highest	6	22.2%	<div></div>	
Not Applicable	0	0.0%		
Totals	27	100.0%		

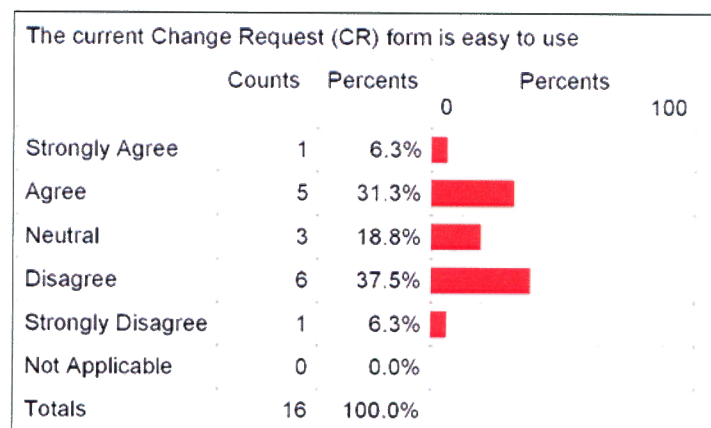
c. The priority of CR review meeting

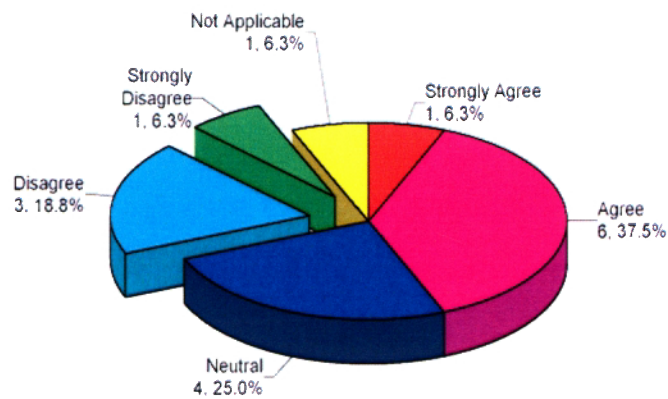


d. The priority of developing task for change implementation

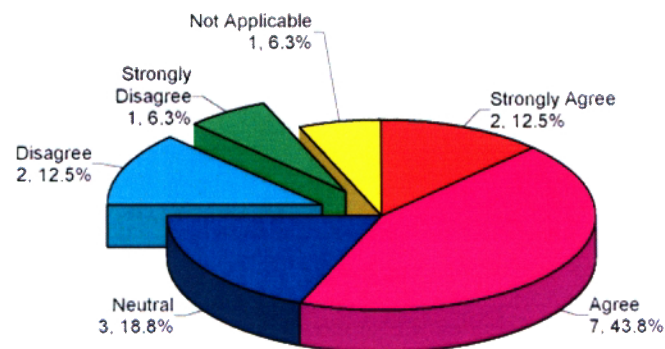


2. Initiator's Perspective





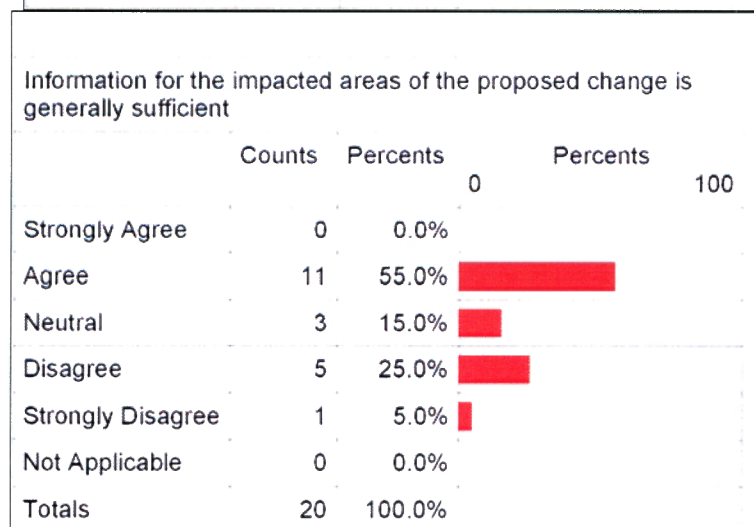
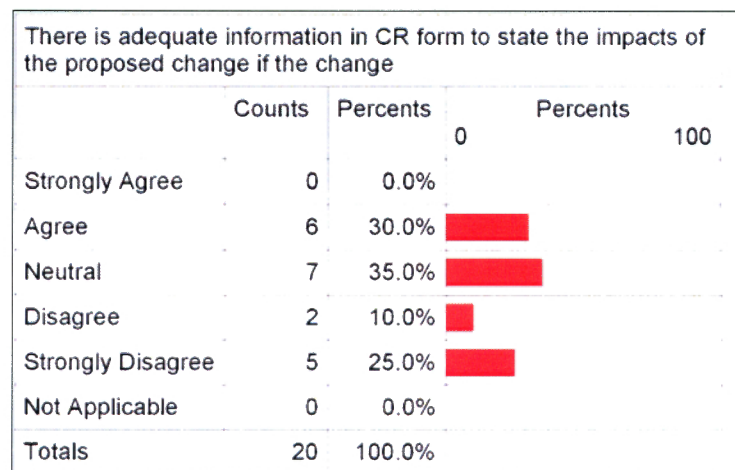
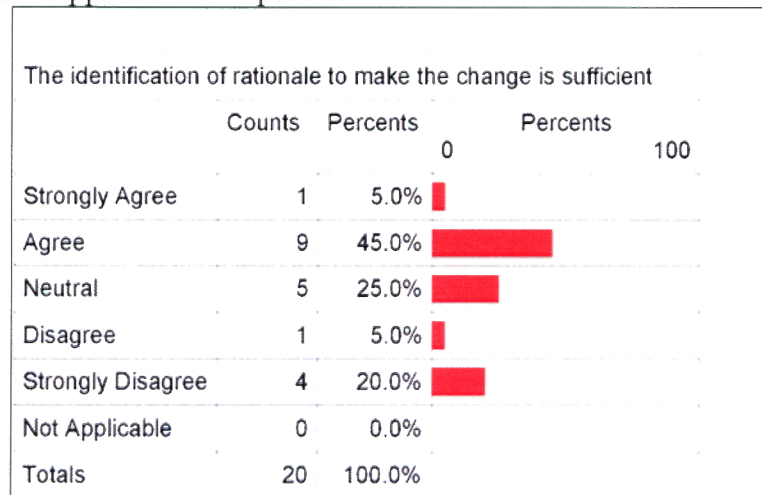
The inclusion of CR types is useful for identifying and analysing change impacts








The inclusion of CR types is useful for choosing the appropriate reviewers

Timeliness of response from Reviewers is generally appropriate				
	Counts	Percents	Percents	
			0	100
Strongly Agree	1	6.3%	<div style="width: 6.3%;"></div>	
Agree	6	37.5%	<div style="width: 37.5%;"></div>	
Neutral	5	31.3%	<div style="width: 31.3%;"></div>	
Disagree	2	12.5%	<div style="width: 12.5%;"></div>	
Strongly Disagree	2	12.5%	<div style="width: 12.5%;"></div>	
Not Applicable	0	0.0%		
Totals	16	100.0%		







3. Reviewer's/Approver's Perspective



The impact analysis information is helpful in guiding you to approve or reject the proposed change





	Counts	Percents	0	Percents	100
Strongly Agree	0	0.0%			
Agree	15	75.0%			
Neutral	2	10.0%			
Disagree	1	5.0%			
Strongly Disagree	1	5.0%			
Not Applicable	1	5.0%			
Totals	20	100.0%			

The impact analysis information is beneficial in estimating work required

	Counts	Percents	0	Percents	100
Strongly Agree	2	10.0%			
Agree	8	40.0%			
Neutral	4	20.0%			
Disagree	3	15.0%			
Strongly Disagree	2	10.0%			
Not Applicable	1	5.0%			
Totals	20	100.0%			

4. Implementer's Perspective

The information stated in the approved CR form is sufficient for change implementation

	Counts	Percents	0	Percents	100
Strongly Agree	0	0.0%			
Agree	9	56.3%			
Neutral	5	31.3%			
Disagree	1	6.3%			
Strongly Disagree	1	6.3%			
Not Applicable	0	0.0%			
Totals	16	100.0%			

The communication with other impacted parties about the change implementation is effective

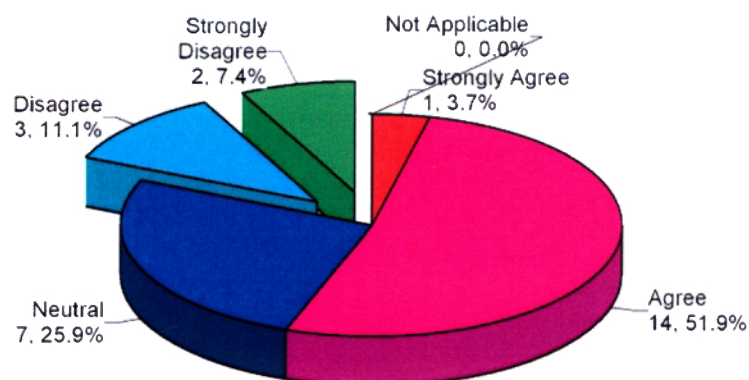
	Counts	Percents	0	Percents	100
Strongly Agree	1	6.3%			
Agree	7	43.8%			
Neutral	5	31.3%			
Disagree	2	12.5%			
Strongly Disagree	1	6.3%			
Not Applicable	0	0.0%			
Totals	16	100.0%			

5. Verifier's Perspective

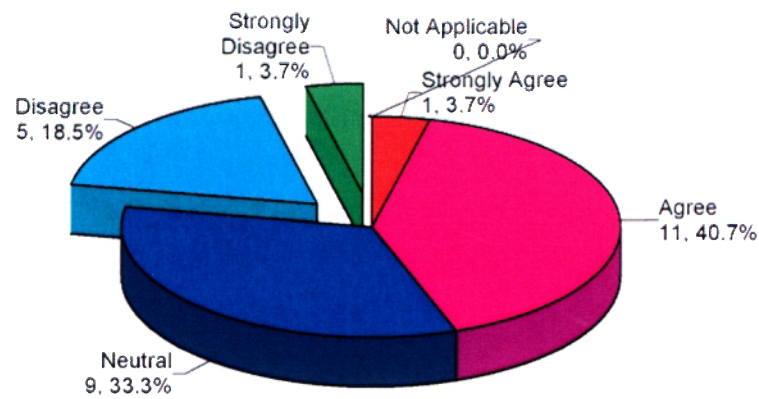
The easiness of tracking the changes made to the project artefacts

	Counts	Percents	0	Percents	100
Excellent	0	0.0%			
Very Good	0	0.0%			
Good	2	50.0%			
Fair	2	50.0%			
Poor	0	0.0%			
Very Poor	0	0.0%			
Not sure	0	0.0%			
Totals	4	100.0%			

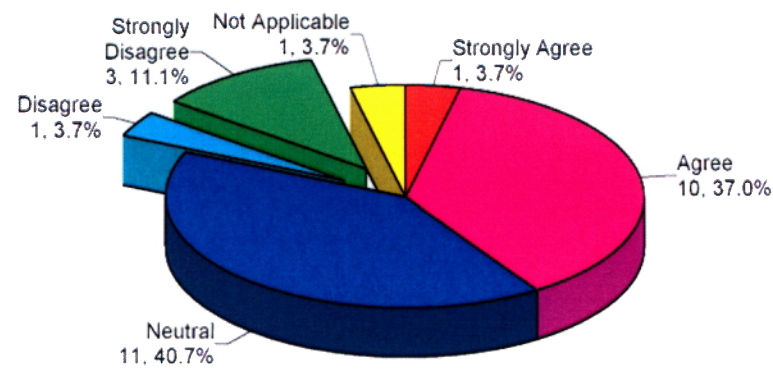
6. Change Synergy Tool



Change Synergy (CS) Tool is easy to use

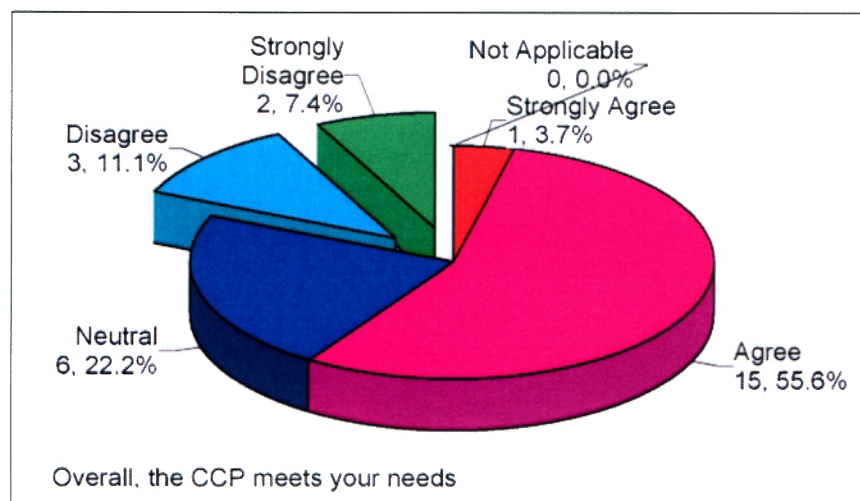


Using the tool to process a change request saves time



Using the tool improves schedule performance for implementing changes

7. Overall



Appendix M: Summary of All Requirements Change Request Data

Sources of change	Reason category	Change Types	Nof docs	Total Requir ements	Estimat ed Effort (hrs)	Elapsed time (days)
Design Spec review	3rd party software/tool inf	mod	5	1	8	56
Design Spec review	3rd party software/tool inf	add	5	1	8	56
Design Spec review	3rd party software/tool inf	del	4	1	13	9
Detailed Design Analysis	3rd party software/tool inf	del	3	1	2.5	4
Functional Spec Review	3rd party software/tool inf	del	1	1	1	9
Functional Spec Review	3rd party software/tool inf	add	3	1	67	18
Functional Spec Review	3rd party software/tool inf	del	2	1	2	32
Customer Support discussion	Clarifying requirement	mod	2	1	1	14
Design Spec review	Clarifying requirement	mod	4	18	20	9
Design Spec review	Clarifying requirement	mod	1	1	0.5	6
Detailed Design Analysis	Clarifying requirement	mod	5	1	15	0
Feature Proposal Review	Clarifying requirement	mod	1	3	0.5	32
Functional Spec Review	Clarifying requirement	mod	2	1	2	19
Marketing Group	Clarifying requirement	mod	1	1	0.5	16
Requirements Review	Clarifying requirement	mod	3	8	8	42
Requirements Review	Clarifying requirement	mod	3	1	10	8
Requirements Review	Clarifying requirement	mod	4	3	6	8
Requirements Review	Clarifying requirement	mod	2	2	2	3
Technical Team Discussions	Clarifying requirement	mod	2	2	1.5	27
Defect Report	Defects Fix	add	4	1	22	16
Defect Report	Defects Fix	add	5	3	148	50
Detailed Design Analysis	Design complexity	del	4	1	21	3
Design Spec review	Design improvement	add	2	1	1.5	27
Detailed Design Analysis	Design improvement	mod	5	1	21.5	21
Detailed Design Analysis	Design improvement	add	4	1	129	10
Functional Spec Review	Design improvement	add	2	2	8	36
Marketing Group + Detailed D	Design improvement	add	4	1	3	25
Marketing Group + Detailed D	Design improvement	add	3	1	8	50
Marketing Group + Detailed D	Design improvement	add	3	1	8	36
Technical Team Discussions	Design improvement	mod	2	1	1	7
Technical Team Discussions	Design improvement	mod	1	2	1	20
Technical Team Discussions	Errorneous requirement	del	4	1	2	26
Product Integration Test	Functional integration	add	3	3	71.5	11
Design Spec review	Functionality enhancement	add	1	9	1	35
Design Spec review	Functionality enhancement	add	4	1	86	25
Detailed Design Analysis	Functionality enhancement	mod	1	9	1	21
Detailed Design Analysis	Functionality enhancement	add	2	3	3	18
Detailed Design Analysis	Functionality enhancement	del	2	2	1	35
Detailed Design Analysis	Functionality enhancement	mod	2	2	2	29
Detailed Design Analysis	Functionality enhancement	add	4	6	77	40
Engineering analysis	Functionality enhancement	mod	1	7	1	13
Engineering analysis	Functionality enhancement	add	2	1	3	13
Feature Proposal Review	Functionality enhancement	mod	1	1	0.5	15
Marketing Group	Functionality enhancement	add	3	1	76	18
Marketing Group	Functionality enhancement	add	7	1	38	28
Project Management	Functionality enhancement	add	5	1	0.5	39
Project Management	Functionality enhancement	add	7	12	2	1
Project Management	Functionality enhancement	add	5	2	2	17
Requirements Review	Implied unnecessary constra	del	2	1	2	3
Design Spec review	Missing requirements	add	3	13	12	28
Design Spec review	Missing requirements	add	4	1	19.5	3
Design Spec review	Missing requirements	add	3	1	10	38
Detailed Design Analysis	Missing requirements	add	5	12	183.5	7
Engineering analysis	Missing requirements	add	1	1	0.5	27

Sources of change	Reason category	Change Types	Nof docs	Total Requirements	Estimated Effort (hrs)	Elapsed time (days)
Project Management	Missing requirements	add	4	1	2	0
Requirements Review	Missing requirements	add	4	1	15	8
Design Spec review	Obsolete functionality	del	1	5	1	23
Feature Proposal Review	Obsolete functionality	del	2	1	0.5	39
Marketing Group	Obsolete functionality	del	5	1	0.5	42
Marketing Group	Obsolete functionality	del	1	1	0.5	15
Marketing Group	Performance Enhancement	add	4	1	0.5	10
Marketing Group	Product Strategy	mod	3	1	17	17
Marketing Group	Product Strategy	add	1	8	2	21
Marketing Group	Product Strategy	add	6	1	93	28
Marketing Group	Product Strategy	add	4	2	53.5	24
Marketing Group	Product Strategy	add	4	4	39.5	30
Project Management	Product Strategy	mod	5	1	88.5	4
Project Management	Product Strategy	add	3	13	214	52
Design Spec review	Redundant Functionality	del	1	4	1	24
Design Spec review	Redundant Functionality	del	1	2	1	28
Engineering analysis	Redundant requirement	del	2	6	1.5	13
Functional Spec Review	Redundant requirement	del	2	3	2	36
Functional Spec Review	Redundant requirement	del	3	1	1.5	0
Requirements Review	Redundant requirement	del	3	2	3	42
Technical Team Discussions	Redundant requirement	del	2	2	1.5	27
Detailed Design Analysis	Resolving conflict	del	4	1	17.5	7
Functional Spec Review	Resolving conflict	del	2	3	1.5	32
Functional Spec Review	Resolving conflicts	mod	1	1	0.5	26
Technical Team Discussions	Resolving conflicts	del	1	1	1	22
Technical Team Discussions	Resolving conflicts	mod	1	7	1	55
Eng & Management discussion	Scope change	del	1	1	0.5	2
Eng & Management discussion	Scope change	del	3	4	7.5	28
Eng + Marketing group	Scope change	del	3	11	12	3
Functional Spec Review	Scope change	del	3	7	7.5	17
Functional Spec Review	Scope change	del	5	2	18	40
Project Management	Scope change	del	2	4	5	18
Technical Team Discussions	Scope change	del	1	4	1.5	20
Detailed Design Analysis	Testability	mod	1	1	0.5	41
Functional Spec Review	Traceability	mod	1	2	1.5	7