MANAGING AND ANALYSING SOFTWARE REQUIREMENTS CHANGES

Submitted by:

Shalinka Erandi Jayatilleke

M.Sc (Information Management)

BSc (Hons) (Computer Systems Engineering)

A thesis submitted in total fulfilment
of the requirements for the degree of
Doctor of Philosophy (by published work)

School of Engineering and Mathematical Science

College of Science, Health and Engineering

La Trobe University

Bundoora, Victoria, 3086

Australia

February 2018

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ACM	Associate for Computing Machinery
AIS	Actual Impact Set
CD	Compact Disk
CDM	Change Dependency Diagram
CEM	Change Event Manager
CEO	Chief Executive Officer
CIO	Chief Information Officer
CMP	Change Management Process
COCOMO	Constructive Cost Model
CW	Change Weight
DAA	Directly Affected Activities
DVD	Digital Video Disk
EIS	Estimated Impact Set
ER	Effort of Rework
GQM	Goal Question Metrics
GSD	Global Software Development
IC	Interaction Comparison
ID	Identity
IdAA	Indirectly Affected Activities
IEEE	Institute of Electrical and Electronics Engineers
IT	Information Technology
IW	Interaction Weight
LOC	Lines of Code
MI	Matched Interfaces
MisMI	Mismatched Interfaces
PERT	Program Evaluation Review Technique
RC	Requirements Change
RCM	Requirements Change Management
RCMP	Requirements Change Management Process

RDF	Resources Development Framework
RQ	Research Question
RUP	Rational Unified Process
SDD	System Design Diagram
SIS	Starting Impact Set
UCM	Use Case Maps
UCP	Use Case Points
UML	Unified modelling language

Abstract

Throughout the development of a software system, there is a tendency for new requirements to emerge and existing ones to change. The management of changing requirements during the requirements engineering process and system development is known as requirements change management (RCM). The literature suggests that unmanaged changes can cause budget and time overruns and lead to poor quality products, which will not satisfy customer requirements. In order to address some of the existing issues of RCM, this thesis aims to create a better understanding of RCM and presents a requirements change management process (RCMP) that encompasses change identification, analysis and rework which will be useful in providing a more rounded solution.

RCM has many aspects that have not been explored and/or understood in depth. A systematic review presented in this thesis brings together research relevant to RCM, providing a holistic picture that encompasses the causes of requirements changes, current issues, solutions provided and the existing knowledge gaps. In the RCMP, change identification is accomplished by a change specification method and a change classification method. The outcome of these methods results in less communication ambiguities and a better understanding of the need for the change. A method of requirement change analysis is developed to identify how requirement changes propagate through the existing system design and also to identify the system activities which are affected due to the changes. The third method presented as part of the RCMP is to analyse the extra work /rework required to implement a requirements change. The work presented gives a clear understanding of rework in the context of RCM with an assessment of the rework calculated using the interactions caused by the changes with the system activities and their connections.

To demonstrate the usefulness of the methods, several methods are applied to a running example which explains the application of the methods step by step. This allows for a better understanding of the mechanics of the methods. The demonstration is further extended by applying all the methods to a larger case study.

Statement of Authorship

This thesis consists primarily of work by the author that has been published / accepted or submitted for publication as described by the text.

Except where reference is made in the text of the thesis, this thesis contains no material published elsewhere or extracted in whole or in part from a thesis submitted for the award of any degree or diploma.

No other person’s work has been used without due acknowledgement in the main text of the thesis.

This thesis has not been submitted for the award of any degree or diploma in any other tertiary institution.

Signed: [illegible] Date: 12/02/18

Dedication

This thesis is dedicated to my loving parents, Sita and Emil Jayatilleke, sister and brother-in-law, Enoka and Chamath Abhayagunawardena and my husband, Alex Romashov (Sasha).

Acknowledgement

It has been a long journey and along the way there have been many people who have assisted me to reach this destination.

First and foremost, I would like to thank my principal supervisor, Assc. Prof. Richard Lai. Thank you for constantly striving to give me insight into the art of research. It was an absolute pleasure working alongside you and learning the ropes of postgraduate research. My project and my skills of researching benefited from your extraordinary wisdom and experience as a researcher and a supervisor. I also thank you for keeping me focused and for your flexibility as on a few occasions we worked together until mid-night. You were persistently patient, understanding and encouraging and worked hard to give me the support I needed as a PhD student, especially when life was rough for me. Your teaching, guidance and care, coaching, and mentoring have enabled me to achieve the best possible outcomes for my PhD studies. You will remain as my invaluable mentor.

I would also like to thank my co-supervisor, Karl Reed. Your words of encouragement and your faith in my success are greatly appreciated. Thank you for all the contributions you made during my candidature. My gratitude is extended to La Trobe University for awarding me a full Postgraduate Graduate Research Scholarship to study towards a doctoral degree. I’m thankful to Michele Mooney of the School of Engineering and Mathematical Sciences for her careful proof reading of my work at various stages of my PhD candidature. To all my friends and colleagues, who have been a source of joy and support, I am truly grateful.

Last but certainly not least, I am thankful to my family. I am indebted to my loving parents Sita and Emil Jayatilleke, for your encouragement, unconditional love and wishes that kept me going and made me determined to achieve this goal. To my sister Enoka and brother-in-law Chamath, without the two of you, I would have never started my journey of higher education and I will never forget that. Sasha, you get a special mention as my partner in life, for the love, support and being a pure source of happiness that put a smile on my face when I really needed it. Without you, this PhD could not have become a reality.

Research Dissemination

Journal paper published

1. **S. Jayatilleke and R. Lai**, "A systematic review on requirements change management," *Information and Software Technology*, vol. 93, pp. 163-185, 2018. DOI: 10.1016/j.infsof.2017.09.004.

Journal papers published online (awaiting print publication)

1. **S. Jayatilleke, R. Lai, and K. Reed**, "Managing software requirements changes through change specification and classification," *To appear in Computer Science and Information Systems,* 2018. DOI: 10.2298/CSIS161130041J.
2. **S. Jayatilleke, R. Lai, and K. Reed**, "A method of requirements change analysis", *To appear in Requirements Engineering*, pp. 1-16, 2017. DOI: 10.1007/s00766-017-0277-7.

Journal paper submitted

1. **S. Jayatilleke and R. Lai**, "A method of assessing rework for implementing software requirements changes", *Requirements Engineering*, 2018.

Conference paper

1. **S. Jayatilleke and R. Lai**, "A method of specifying and classifying requirements change," in *Software Engineering Conference (ASWEC)*, 2013 22nd Australian, 2013, pp. 175-180: IEEE.

Structure of thesis

This is a thesis by publication. La Trobe University doctoral thesis by publication guidelines state that the format of the thesis, based primarily on work completed during candidature, may be presented 'as a collection of articles or book chapters including at least one substantial integrating chapter, or a separate introduction, general discussion and conclusion that reveal the way the articles and book chapters are thematically linked'.

This thesis is structured according to these guidelines. For ease of reading, the publications have been incorporated into the main body of the thesis document as word documents.

Chapter 1: The introduction provides a foundation for the research that includes the background, research problem and motivation, aim and scope of the work, and contributions made.

Chapter 2: *A systematic review on requirements change management* - In this chapter, we present a systematic review on the current practices and challenges of RCM. The review is a result of 184 studies. The findings will enable readers to understand the current practices, benefits, challenges, and the risks associated in RCM. Through this review, developers as well as business personnel will be able to understand the importance of RCM and be informed of a plethora of ways to manage changes. This will lead to being better informed in making decisions on the associated requirements changes and can be used as a guide in selecting the methods of change management as appropriate for the situation. This chapter was published as a paper in the January 2018 issue of the Web of Science Journal, Information and Software Technology, Elsevier.

Citation	Aim
S. Jayatilleke and R. Lai, "A systematic review on requirements change management," Information and Software Technology, vol. 93, pp. 163-185, 2018. DOI: 10.1016/j.infsof.2017.09.004.	To understand what research work has already been achieved in the area of RCM and identify knowledge gaps to enable future research work.

Chapter 3: *Managing software requirements changes through change specification and classification* - In this chapter, an examination of the various methods used for change communication is carried out, as it is the first step in requesting a requirements change. Successfully completing this step will result in the elicitation of the correct goals in relation to the changes and knowing these goals will result in the successful implementation of the said changes. Therefore, in this chapter, a new way to communicate the required changes through specification and classification methods is established. The methods will minimise the communication ambiguities that hinder the process of understanding the changes at an operational level and will also facilitate guidelines for implementation. This chapter has been published online and will appear as a paper published in the June 2018 issue of the Web of Science Journal, Computer Science and Information Systems, ComSIS Consortium. A preliminary version of this paper was presented at the 2013 Australian Software Engineering Conference, published in IEEE.

Citation	Aim
S. Jayatilleke, R. Lai, and K. Reed, "Managing software requirements changes through change specification and classification," To appear in Computer Science and Information Systems, 2018. DOI: 10.2298/CSIS161130041J	To address the existing knowledge gap in the lack of research done in understanding and interpreting requirements changes.

Chapter 4: *A method for requirements change analysis* – This chapter explores the further analysis of the requirements changes identified in chapter 3. This method will provide the developers with increased understanding of how changes interact with each other as well as the processes in the system. The conflicts and/or dependencies identified through this method will facilitate better decision making on prioritizing changes, selecting the best possible implementation path and to realize the complexities associated in executing the change. This chapter has been published online and will appear as a paper published in the Web of Science Journal, Requirements Engineering, Springer.

Citation	Aim
S. Jayatilleke, R. Lai, and K. Reed, "A method of requirements change analysis," Requirements Engineering, pp. 1-16, 2017. DOI: 10.1007/s00766-017-0277-7.	To enable better decision making based on the further analysis of requirements changes and address the drawbacks of existing methods on impact analysis.

Chapter 5: *A method of assessing rework for implementing software requirements changes* – The work is initiated by defining rework in the context of RCM and then describes a method of analysing rework for implementing software requirements changes. The outcome of the methods introduced in chapter 3 and 4 are used in the initial stages of this method to produce a value that can be used as the assessment of the rework required to implement a change. This finding enables developers to have a better understanding of the rework required to compare the rework between the different options available for implementing a change and to identify the option that involves the minimal rework. This chapter has been submitted to the Web of Science Journal, Requirements Engineering, Springer.

Citation	Aim
S. Jayatilleke and R. Lai, "A method of assessing rework for implementing software requirements changes," Submitted to Requirements Engineering.	To understand the concept of rework in RCM and generate numerical values that can be interpreted to assess the rework required to implement a requirements change.

Chapter 6: *Conclusions and future work* – Provides the conclusion and the implication for future work. This chapter summarises the work conducted in the previous chapters and provides an evaluation of the work in terms of its strengths and limitations.

References: list of the previous work referenced in this thesis.

Appendices: list of appendices to support the work in this thesis.

Chapter 1

Introduction

1.1 Background

In the current environment, software systems are becoming increasingly more complex. As a result, the requirements of software systems are more prone to change. New requirements emerge and existing ones change throughout the development of software systems [1, 2]. New and/or modified requirements need to be integrated with existing ones, along with adaptations to the architecture and source code of the software system. Unmanaged changes can be quite detrimental to the successful completion of a software project and the need to put in place an effective change management process has become a necessity. In its most primary context, Sommerville [3] defines requirements change management (RCM) as a process of “managing changing requirements during the requirements engineering process and system development”. Requirements changes can be caused by changing user requirements and business goals and/or be induced by changes in implementation technologies.

The size and complexity of software systems make RCM costly and time consuming. A significant percentage of software system budgets goes to the operation and maintenance of software systems [4]. It is important that these changes are identified as early as possible in the development life cycle as correcting requirements errors late can cost up to 200 times as much as correcting these errors during the requirements phase [5]. Attention to upfront requirements activities has been said to produce benefits such as preventing errors, improving quality, and reducing risk throughout software development projects [6, 7]. Studies conducted by the Standish Group [8] found a striking 74 percent project failure rate, while 28 percent of projects were cancelled completely. The study suggests that the top factors of failure are related to requirements problems, including a lack of user input, a lack of a clear statement of requirements, and incomplete and changing requirements. Therefore understanding requirements changes and their impact on the rest of the development artifacts would be beneficial for software development projects.

Managing requirements changes can be problematic due to users' evolving needs, disagreement among customers or stakeholders on agreed requirements, errors in communicating (understanding) changes to the developers, changes in organization goals and policies, etc. A survey of 4000 European companies found that the management of customer requirements was one of the principal problem areas in software development and production [9]. Although there are several techniques that address the problem space of RCM, research has shown there are still many voids to deal with.

1.2 Research problems and motivation

The research problems addressed in this thesis focus on understanding the current state of research on RCM and the existing issues in managing requirements changes based on observations of the available literature. The formation of the research problems are also based on the drawbacks and limitations of existing techniques related to RCM.

RCM, although not a new concept, has many aspects that have not yet been explored and/or understood in depth. A plethora of research has been conducted in this area, yet there is no indication of any work that has collated the varied information relevant to RCM in a meaningful manner to provide a holistic picture regarding RCM and identify the existing issues and knowledge gaps. Prior to developing solutions for existing issues, it is important to uncover the causes of requirements changes. These causes tend to form how changes are managed and decisions are made concerning RCM. The lack of understanding of the research space of RCM hinders new efforts in identifying the issues in RCM as well as the development of new solutions.

A key issue in RCM is the lack of effective communication between business and IT staff in relation to change [10-13]. Clear communication plays an important role in the effective identification of requirements changes. In order to establish a common communication medium, several research studies have attempted a few different techniques such as taxonomies, classifications and card sorting [14-23]. These existing works have been evaluated as having several drawbacks and limitations, hence the need for a formal/semi-formal change identification technique still exists. The literature also suggests that there is little agreement and commonality between the existing methods, which makes it difficult to compare these methods and select an appropriate one for real-time application. Furthermore, most of the existing methods lack guidance in applying them to change management activities. Therefore, the requirements change (RC) communication ambiguities can lead to an end product that does not satisfy the customer requirements and/or is of poor quality.

Effective communication alone does not equate to the successful management of these changes. Requirements are usually not standalone entities but have complex connections and relationships with the other requirements of the system. As a result, a change administered to one requirement may affect many other requirements [24-28]. Therefore, the need to analyse and identify the impact of change is an essential characteristic in RCM. One of the popular techniques used in locating the impact of change is traceability. However, this technique has a few drawbacks and limitations. The lack of effective impact analysis techniques hinders efforts to understand a RC in depth, which then, in turn, could result in poor quality / incorrect implementation of the change.

Another aspect of RCM that has been overlooked is the additional work generated implementing a RC. This extra work becomes an important factor as part of the impact on the cost drivers and the duration of the project [5, 29, 30]. The extra work, referred to as rework, therefore becomes a key contributing factor for change effort estimation. Depending on the complexity of the changes, the amount of rework required can vary from some software module modifications to complete overhaul of a system. Therefore, the effort associated with change implementation activities will vary as well. It is evident from the literature that very little research work exists on analysing the rework required for requirements change implementation as well as for change effort estimation. Furthermore, the relationship between rework and change effort estimation has not been understood particularly well.

1.3 Aims and scope of the work

In many instances, successful software has to conform to the requirements of its customers and users. The cost of changing requirements increases exponentially as the development progresses through the project's phases [1, 6]. RCM is one of the core activities utilized to attain one of the main objectives of software development, which is to satisfy the evolving needs of customers at a reasonable cost and time. Most approaches to managing RCs can be classified as methods for change impact analysis, change cost calculation, change complexity analysis and change verification. Even though many such methods exists, software development projects and organizations still struggle in the battle to manage RCs.

The aims of this thesis are to create a better understanding of RCM and to develop a requirements change management process (RCMP) that encompasses several different characteristics that would be useful in providing a more holistic solution for the issues facing RCM. It is intended that this process will assist both the business and the development sides of a software project / organization. By analysing the literature related to RCM, and the techniques that are developed as part of the RCMP, and setting our research questions based on the unresolved issues therein, it is anticipated that the work in this thesis will enable the following:

1) a clearer understanding and a more complete view of RCM in terms of the causes of requirements changes, the current issues, the solutions provided and the existing knowledge gaps;

2) better communication of RCs from the business side to the IT side, providing a better understanding of the required change;

3) the identification of the conflicts and/or interdependencies between RCs and provide a decision criterion to analyse the requirements changes in more depth;

4) a clearer understanding of the rework needed in the context of RCM along with an analysis of the rework required in implementing a RC;

The usefulness of the proposed methods of the RCMP is illustrated by applying them to case studies. The research process, findings and contributions of this research are discussed in detail in the subsequent chapters of the thesis.

1.4 Contributions

In pursuit of the overall objective of the better management of requirements changes, this thesis makes several contributions to the knowledge on RCM. In this section, the major contributions of the research are described in brief.

A systematic review of requirements change management: Understanding the research space of RCM is initiated through a systematic review. It became evident during the review that there is a large body of information related to RCM, covering a wide range of issues, and yet no work has been done in collating this information into one piece of work that would be beneficial for researchers in the area. The outcome of the review resulted the identification of the following: (1) the causes of requirements changes; (2) the various process used for RCM; (3) the techniques used for RCM; and (4) the decision making related to RCM. Based on the various processes used, three main areas of RCM were identified: (1) change identification; (2) change analysis; and (3) change cost/effort estimation. The methods developed in this thesis are intended to fill the gaps identified during the review, and, of course, are "informed" by them.

Managing software requirements changes through change specification and classification: The RCMP begins with a combination of two methods for requirements change specification and classification. This two-stage process consists of communication of the requirements changes through the specification method and understanding the change through the classification method. The key features of the methods are as follows: (1) they eliminate ambiguities when communicating change between business and IT personnel at an operational level, to a better understanding of the reason for the change. As a result, developers are able to see the relevance of the change to the system, further helping the decision-making process; and (2) the clear guidelines provided by the change classification process not only provide a basic course of action for incorporating the change into the system but also determines (when possible) multiple routes to implementation. This may lead to a minimization of the conflicts between multiple change implementations. As a result, change elicitation using this method produces a better outcome in incorporating the change into a system.

A method of requirements change analysis: The changes elicited using the specification and classification methods will be stored in a change event log. These identified changes will then undergo further analysis. This three-stage process consists of application of the change analysis functions, calculation of the change difficulty, and application of the change dependency matrix. The key features of the method are as follows: (1) the change dependency matrix will help change implementers to identify conflicts and/or dependencies between multiple changes. The identification of such dependencies will make the process of determining the suitability of implementing the change possible. This will also help in deciding which route to take if there are multiple paths identified for the implementation of the change (through previous methods) as the less conflicted path can be easily identified; (2) as part of the change analysis, the method will calculate the difficulty level of implementing each change. The changes will be prioritized based on the results of the calculation. The difficulty level of the change will assist the developers, as it will indicate how rigorous the change implementation activity will be. The priority level will assist in determining the implementation order of the changes.

A method of assessing rework for implementing software requirements changes: This is the final stage of the RCMP method. The work in this section commences by defining the term rework in the context of RCM. This is a three-step process. In the first two steps, the method uses the findings of the previous two methods to identify the activities affected by the change. In the last step, an assessment of the rework is carried out using an interaction comparison and interaction weight. The key features of the method are as follows: (1) a numerical representation of the assessment of rework for all possible implementation options of a requirements change; (2) selection of a particular implementation option with lesser rework; and (3) comparison of the assessment of rework between multiple requirements changes.

Chapter 2
A Systematic Review of Requirements Change Management

2.1 Preface

Requirements change and its management has been a topic of interest with researchers and software development teams alike for a very long time. This interest is due to the importance requirements change management holds in relation to the success of a software project. We have discovered that the volatility of requirements and the improper management of their impact is a major contributor to project failure. The main purpose of this chapter is to identify what research work has been carried out to date and based on the information gathered; develop a comprehensive understanding of the research space. Furthermore, we use this review to identify the strengths and limitations of the existing methods in managing requirement changes as well as to uncover the research gaps that can potentially lead to future research work.

The chapter consists of a paper that investigates the research space of requirements change management. This investigation has led to establishing four key research questions. The findings of these research questions steer the formation of the rest of the thesis.

2.2 Publication

S. Jayatilleke and R. Lai, "A systematic review on Requirement Change Management," *Information and Software Technology*, vol. 93, pp. 163-185, 2018. DOI: 10.1016/j.infsof.2017.09.004.

Manuscript title	**Publication status**	**Nature and extent of candidate's contribution**	**Nature and extent of co-author's contribution**
S. Jayatilleke and R. Lai, "A systematic review of requirements change management". Information and Software Technology	Published in accordance with Information and Software Technology author guidelines.	**Eighty percent** contribution by the candidate. This included gathering information, drafting and revising the manuscript.	**Twenty percent** contribution by the co-author. This included discussions of the ideas expressed in the paper, critical review and submission to the journal.

Signed : [illegible] Date : 12/02/18

(S. Jayatilleke)

Signed : [illegible] Date : 12/02/18

(R. Lai)

A Systematic Review of Requirements Change Management

Abstract

Context: Software requirements are often not set in concrete at the start of a software development project; and requirements changes become necessary and sometimes inevitable due to changes in customer requirements and changes in business rules and operating environments; hence, requirements development, which includes requirements changes, is a part of a software process. Previous work has shown that failing to manage software requirements changes well is a main contributor to project failure. Given the importance of the subject, there's a plethora of research work that discuss the management of requirements change in various directions, ways and means. An examination of these works suggests that there's a room for improvement.

Objective: In this paper, we present a systematic review of research in Requirements Change Management (RCM) as reported in the literature.

Method: We use a systematic review method to answer four key research questions related to requirements change management. The questions are: (1) What are the causes of requirements changes? (2) What processes are used for requirements change management? (3) What techniques are used for requirements change management? and (4) How do organizations make decisions regarding requirements changes? These questions are aimed at studying the various directions in the field of requirements change management and at providing suggestions for future research work.

Results: The four questions were answered; and the strengths and weaknesses of existing techniques for RCM were identified.

Conclusions: This paper has provided information about the current state-of-the-art techniques and practices for RCM and the research gaps in existing work. Benefits, risks and difficulties associated with RCM are also made available to software practitioners who will be in a position of making better decisions on activities related to RCM. Better decisions will lead to better planning which will increase the chance of project success.

Keywords

Requirements change management; Agile; Systematic review

1. Introduction

Change is an intrinsic characteristic of the software engineering discipline compared to other engineering disciplines. In real-world scenarios, it is difficult to specify all the requirements for software as the need and the circumstance of the scenario is subject to change. Factors such as customer needs, market change, global competition, government policies, etc. contribute profoundly to the changing nature of requirements. The need for increasingly complex software is in high demand as organizations struggle to survive in a highly competitive market. Therefore, managing change in software development is not just important but crucial for the success of the final product.

Nurmuliani [31] defines requirements volatility as "the tendency of requirements to change over time in response to the evolving needs of customers, stakeholders, the organisation and the work environment". Requirements, in principle, are the needs and wants of the users and stakeholders of the system captured by an analyst through an elicitation process [3]. These requirements change (RC) throughout the system development and maintenance process, which includes the whole lifecycle of a system: requirement formation, analysis, design, evaluation and learning [1, 3, 16, 17, 31-41]. As this review progresses, we discuss in detail the factors that can cause these requirements changes. Therefore, requirements change management (RCM) can be defined as the management of such changing requirements during the requirements engineering process, system development and the maintenance process [3, 33, 42]. This definition of RCM is an adaptation of the definition provided by Sommerville [3] who states RCM is a process of "managing changing requirements during the requirements engineering process and system development".

Managing such evolving changes has proved to be a major challenge [38-41]. The consequences of unmanaged or improperly managed requirement changes can spell disaster for system development. These negative consequences can result in software cost and schedule overrun, unstable requirements, endless testing and can eventually cause project failure and business loss [31, 43-49]. Therefore, the proper management of change can be both rewarding and challenging at the same time.

The research area of RCM is of importance to many parties as requirements change is a constant factor. Many research studies on have been conducted on improving RCM and many more have been conducted to look for answers in the knowledge gaps found in the current research. The main motivation of this research paper is to bring together the plethora of research work done in the area of RCM into one location. This will enable software practitioners and researchers alike a reference point in acquiring knowledge on the current practices, benefits, risks and difficulties associated with RCM. As a result, they can form realistic expectations before making decisions on activities related to RCM. Better decision making will lead to better planning which will increase the chance of project success. An equally important reason to conduct this research is to identify the knowledge gaps in the area of RCM. Given that a lot of research work has been done in this area, we felt it is important for us as well as other researchers to understand the future of RCM. Although this is a widely researched area, there are many gaps still remaining that once recognized and remedied could assist organizations immensely.

2. Research questions (RQs)

To gain an understanding of current trends, practices, benefits and challenges in RCM, we formulated the following four questions;

RQ_1: What are the causes of requirement changes?

The motivation behind this question is to understand why requirement changes occur, which leads to the realization as to why this has been an evolving topic. To answer this question, we investigated various events and uncertainties that have been mentioned in literature. We also investigate whether there is any commonality between these events that would lead to a recognition pattern in predicting RCs.

RQ_2: What processes are used for requirements change management?

The motivation behind this question is to understand the various steps involved in managing RCs. To answer this question, we investigated the following: (1) recommendations for semi-formal methods of managing change; (2) formal process models available for RCM

RQ_3: What techniques are used for requirements change management?

The motivation for this question is to identify and understand the state-of-the-art techniques in managing major areas of the RCM process. To answer this question, we identify the main steps required to manage RC based on the answer to RQ_2 and then identify in the literature what techniques have been used in each of these steps.

RQ_4: How do organizations make decisions regarding requirements changes?

The motivation behind this question is to discover what factors are involved in making decisions regarding RCs at different organizational levels. To answer this question, we first identify the main levels of an organization and use the information available in the literature on RCM that can be mapped to each level.

3. Review approach

The systematic review was designed in accordance with the systematic review procedures and processes defined by Kitchenham [50, 51]. According to Kitchenham [50], there are 10 sections in the structure of a systematic review: 1. Title; 2. Authorship; 3. Executive summary or abstract; 4. Background; 5. Review questions; 6. Review Method; 7. Inclusion and exclusion of studies; 8. Results; 9. Discussion; and 10. Conclusion. The first 5 sections have been covered so far. The review method comprises four sections: 1. Data search strategy; 2. Study selection; 3. Data extraction; and 4. Data synthesis. This section comprises the review method and the inclusion and exclusion of studies. The results, discussion and conclusion are presented in the next section.

3.1. Study objectives

As noted earlier, the objective of this literature review is to thoroughly study the background and existing methods in RCM and thereby provide a critical analysis of the relevant research work and identify future directions for improvement.

3.2. Selected sources

In order to carry out a comprehensive analysis, search strings were established by combining the keywords through the logical connectors "AND" and "OR". The studies were obtained from the following search sources: IEEE, ACM, Science Direct (Elsevier), Springer, Wiley Inter Science, and Google Scholar. The quality of these sources guarantees the quality of the study.

3.3. Selected language

The English language is the most commonly used language in the world and most of the available research is written in English. Therefore, only papers which are written in English were selected for the literature review.

3.4. Data search

To answer the research question, we undertook the search using four steps;

Step 01 – Identify the fundamental areas to finalize the scope of the review.

Step 02 – Select key words / strings from the defined areas. Key words / strings were limited to seven (see Table 1).

Step 03 – Describe search expressions based on the first two steps i.e. [Expression = (A_1 OR A_2 OR A_3 OR A_4 OR A_5 OR A_6 OR A_7 OR A_8 OR A_9 OR A_{10} OR A_{11}) AND (B_1 OR B_2 OR B_3 OR B_4 OR B_5)].

Step 04 – Use the search expression in the libraries mentioned in the selected sources.

Category	Area	Keywords / Strings
A	Requirement Change Management	A_1 – Requirement change/volatility/creep
		A_2 – Requirement change difficulties
		A_3 – Requirement change management
		A_4 – Requirement change management models / Processes
		A_5 – Requirement change identification/type
		A_6 – Requirement change analysis
		A_7 – Requirement change factors/causes
		A_8 – Requirement change decisions
		A_9 – Change impact analysis
		A_{10} – Agile requirement change management
		A_{11} – Requirement change cost estimation

B	Nature of study	B_1 – Case study
		B_2 – Experiment
		B_3 – Surveys
		B_4 – Industrial
		B_5 – Literature reviews

Table 1: Categories and keywords

3.5. Study selection (Inclusion and exclusion of studies)

Once the research questions and the data search mechanism were defined, we started the process of selecting studies which fell under the defined scope and contained the keywords set out in the review process. As shown in category A of Table 1, the area of RCM has a lot of potential as change is a constant factor. As a result, our search yielded hundreds of research papers and studies. After screening these papers, we came to the conclusion that 28% (184) were relevant to the study.

Papers were excluded for a number of reasons related to format (editorial, seminar, tutorial or discussion), repetition, lack of peer review, lack of a focus on RC and RCM, redundancy and lack of quality. Several papers appeared in more than one research repository. We eliminated the repetitions and only considered one instance of a paper. Details on repeated articles do not provide any significant information, except the names of the articles which have been published by more than one publishing authority (e.g. IEEE, ACM). As a result, we do not mention the names of the repeated articles which were found during the study selection process. In the initial phase, the extracted papers were independently reviewed by both authors based on the inclusion and exclusion criteria. In the secondary phase, both authors compared their outcome of their selection and through discussion, came to agreement on the inclusion and exclusion of papers. The overall inclusion process comprised five steps, as shown in Table 2. Table 3 provides details of the reasons for the exclusion of 466 papers.

Analysis Phase	Inclusion Criteria	Number of Papers
1. Initial search	• Papers written in English • Available online • Contain search keywords and strings	650
2. Scrutinizing titles	• Only published in journals, conferences, workshops and books • Not an editorial, seminar, tutorial or discussion	573

3. Scrutinizing abstract	• Experiments, case studies, literature reviews, industrial and surveys	340
4. Analyzing introduction and conclusion	• Main contribution in the areas of search strings	230
5. Analyzing main contribution	• Reported significant contribution • Originality of work • Sole focus related to the theme of this review study	184

Table 2: Study selection process

Exclusion Criteria	No.
Paper format (editorial, seminar, tutorial or discussion)	95
Repetitions	43
Lack of peer review	75
Lack of a focus on RC and RCM	110
Redundancy	98
Lack of quality	45
Total	466

Table 3: Classification of exclusion

3.6. Data extraction

After completing the study selection process, we recorded basic information on each paper in data extraction form (refer to Table 4) to gather information on the causes of RCs, the study focus, RCM processes / models, RC identification, RCM techniques, reported challenges in RCM, decision making in RCM, study findings and knowledge gaps in RCM. The non-experimental models which presented a proposal without conducting experiments were also applied.

Aspects	Details
Study ID	Paper ID
Title	Title of paper
Authors	Names of authors
Publishers	Name of publishing authority
Publishing date	Date of publication
Causes of RCs	Factors that cause requirement changes
Study focus	Focus and perspective of paper
RCM processes / models	Processes / models listed for managing RC
RCM techniques	Techniques used for RCM (identification, impact analysis, cost estimation, etc.)
Reported challenges in RCM	Challenges and consequences associated with RCM
Decision making in RCM	Factors involved in decision making related to RCM
Study findings	Lessons learned from the paper
Knowledge gaps in RCM	Implications for future work

Table 4: Data extraction process

3.7. Data synthesis

Kitchenham [50, 51] states that there are two main methods of data synthesis: descriptive (qualitative) and quantitative. The extracted data were analysed using a qualitative method to answer our research questions, which leads to a descriptive data synthesis. One of the co-authors of this paper has published qualitative systematic reviews [52, 53] using similar techniques. The analysis used the constant comparison method [54] in comparing studies past and present in RCM. Using this method, we present the focus of the studies, the proposed methods, applicability to requirement change management, lessons learned from the studies and drawbacks and limitation of the studies.

4. Results for RQ_1: What are the causes of requirements changes?

It is anticipated that requirements will change during a project life cycle. Whilst this fact is a constant, delayed discovery of such changes poses a risk to the cost, schedule and quality of the software [32, 55-57] and such volatility constitutes one of the top ten risks to successful project development [56-58]. Pfleeger [59] recommends that a method needs to be developed to understand and anticipate some of the inevitable changes during the development process in order to reduce these risks. The identification of factors that cause or influence requirements uncertainty is a necessity. The recognition of such factors will support requirements change risk visibility and also facilitate better recording of change data [56, 57].

Change cause factors were collected using a key word search on academic papers, industry articles and books that deal with change management or requirement engineering. We used the search expressions A_1 OR A_2 OR A_3 OR A_5 OR A_7 (see Table 1).

Most literature extracted in this survey mentioned/indicated the reasons for requirement changes. However, it was deemed necessary to present these findings in a form that was meaningful rather than listing all the causes of RCs mentioned in the literature. Of the literature extracted, there were three studies that formally classify the causes of RCs. Weiss and Basili [60] divide changes into two categories: error correction and modifications. This classification appears to be simplistic and categorising all the identified change causes may not create an in-depth understanding. Bano et al. [61] classifies change causes also under two categories; essential and accidental. They further classify the change causes based on their origin: within the project, from the client organization and from the business environment. McGee and Greer [56, 57] use five areas/domains to classify change causes. For this survey, we use the classification presented by McGee and Greer as it has a more comprehensive categorization. The five change areas are: external market, customer organization, project vision, requirement specification and solution. Within the five change areas, they distinguish between two causes of change: trigger and uncertainty [56]. The difference between these two categories is that an event can cause a change without pre- or post-uncertainty. However, uncertainty cannot cause a change to occur without an event that is triggered to manage the risk of the uncertainty. The factors that were identified as causes of requirements change were sorted into five areas as follows:

(i) Change area: External market

In this category, the changes to the requirements are triggered by the events and uncertainties that occur in the external market which also include stakeholders. These stakeholders include parties such as customers, government bodies and competitors. Therefore, events such as changes in government policy regulations [10, 62, 63], fluctuations in market demands [10, 31, 63, 64] and response to competitors [41, 63, 65, 66] can be considered. Also, uncertainties such as the stability of the market [41, 67] and the changing needs of the customers [41] are also part of this category.

(ii) Change area: Customer organization

In this category, changes to the requirements are triggered by the events and the uncertainties that arise from a single customer and their organizational changes. Although the changes occur within the customer's organization, such changes have a tendency to impact the needs of the customer and as a result, impact the design and requirements of the software project. Therefore, events such as strategic changes within the organization [16], restructuring of the organization [10, 31, 62, 64, 68], changes in organizational hierarchy [41, 63, 69] and changes in software/hardware in the organization should be considered. The stability of the customer's business environment can create uncertainties that may lead to changes and these are also part of this category.

(iii) Change area: Project vision

In this category, the changes to the requirements are triggered by changes in the vision of the project. These changes are in response to a better understanding of the problem space from a customer point-of-view and the emergence of new opportunities and challenges. Events such as improvements to business processes [3, 63], changes to business cases due to return on investment [16], overrun in cost/schedule of the project [62, 64], identification of new opportunities [62] and more participation from the stakeholder [10] should be considered. Uncertainties, such as the involvement of all stakeholders [63, 68-70], novelty of application [63, 71], clarity in product vison [10, 63, 70, 72], improved knowledge development team in the business area [69, 71], identification of all stakeholders [68, 70], experience and skill of analyst [29, 63, 69, 72], size of the project [3, 69, 73] can also cause changes under this category.

(iv) Change area: Requirement specification

In this category, changes in the requirements are triggered by events and uncertainties related to requirement specification. These trigger events are based on a developer's point-of-view and their improved understanding of the problem space and resolution of ambiguities related to requirements. Events such as increased understanding of the customer [3, 62, 63, 73, 74], resolution of misunderstandings and miscommunication [41, 75, 76] and resolution of incorrect identification of requirements [31] can be considered as change triggers. Uncertainties, such as the quality of communication within the development team [16], insufficient sample of user representatives [16], low staff morale [29], quality of communication between analyst / customer [63, 67, 69, 73], logical complexity of problem [69, 71, 73], techniques used for analysis [3, 29, 62-64], development teams' knowledge of the business area [69, 71], involved customers' experience of IT [71], quality of requirement specification [16], and the stability of the development team [16] can contribute towards change under this category.

(v) Change area: Solution

In this category, changes in the requirements are triggered by events and uncertainties related to the solution of the customer's requirements and the techniques used to resolve this. Events such as increased understanding of the technical solution [16], introduction of new tools/technology [10, 33, 41, 62, 63, 66, 68, 77] and design improvement [41, 62, 75] should be are considered as change triggers. Technical uncertainty and complexity can also be considered under this category as a cause of change [16].

The five change areas listed above can be mapped to the classification proposed by Bano et al. [61]. The terms *essential* and *accidental* were initially introduced by Brooks [6]. According to Bano et al. [61], change causes under the essential category are those that are inherent in nature and cannot be controlled i.e. "fluctuating market demand" cannot be controlled or avoided by the development team or the organization. In comparison, accidental causes can be controlled and avoided i.e. "overrun in cost/schedule of the project" can be avoided or at least controlled by putting better techniques and mechanisms in place. Being able to categorize change causes under these two categories has added benefits in managing RCs. With essential causes, the focus should be to deal with their impact and therefore use techniques that will reduce time and effort for their management. With the accidental causes, the focus should be to use techniques that avoid such occurrences. Table 5 shows how these five categories in McGee and Greer's classification [56] can be mapped to Bano et al.'s classification [61] of essential and accidental categories.

Bano et al.'s Classification [61]	McGee and Greer's Classification [56]		
Essential	External market	Customer organization	
Accidental	Project vision	Requirement specification	Solution

Table 5: Comparison between classifications

Key findings of RQ$_1$

Given that RC is an inevitable occurrence in any development project, it is beneficial to identify which factors can cause these changes. The knowledge gained through such findings will enable all stakeholders of a project to better manage the changes when they occur, develop systems based on the changes, and anticipate certain changes. Based on the discussion formulated for RQ$_1$, the following are the key findings:

1) The factors that cause RCs can be divided into two categories: change trigger events; and uncertainties.

2) In reality, it is difficult to determine whether change happens as a result of one or both. In a practical sense, it is not important that the causes of the changes are divided into these two categories, as long as they are identified.
3) These identified changes can be categorised into five areas: external market; customer organization; project vision; requirement specification; and solution.
4) These five areas were identified by observing the characteristics of the change events and the uncertainties discussed in the literature. For example, any change factor that was part of the external environment of the organization, such as competitors, government regulations, etc. was categorised as the external market.
5) These five areas can be divided into two categories: essential and accidental. Based on this division, development teams can be proactive in managing such changes.
6) Based on the location in the life cycle of the software project, the above information can be meaningful for anticipating what factors may cause change and as a result will lead to better planning that will ensure a better success rate for the project.

5. Results for RQ$_2$: What processes are used for requirements change management?

In order to answer RQ$_2$, the following sections discuss various processes suggested for managing RC and the process models that are dedicated for RCM. We used the search expressions A$_3$ OR A$_4$ OR A$_6$ OR A$_9$ OR A$_{10}$ (see Table 1) to extract the relevant literature.

5.1 Semi-formal methods available for requirements change management

Change is considered to be an essential characteristic of software development and successful software has to be adapted to the requirements of its customers and users [33, 78, 79]. Thus RCM has become a significant activity, which is undertaken throughout the development of the software and also during the maintenance phase. Given the significance of this activity, it is unlikely that change management is undertaken in an ad-hoc manner. According to Sommerville [3], the process of RCM "is a workflow process whose stages can be defined and information flow between these stages partially automated". Having a proper process for RCM is linked with both improvement in the organizational processes and the success of software projects [33, 34, 80]. We have identified four (i - vii) academic works that refer to establishing semi-formal methods for managing change.

(i) Proposal: Leffingwell and Widrig [81]

This is a five-step process for managing change. The process is as follows:

1. Recognize that change is inevitable, and plan for it.
2. Baseline the requirements.
3. Establish a single channel to control change.
4. Use a change control system to capture changes.
5. Manage change hierarchically.

The process begins with a change management plan which recognizes that change is unavoidable. Requirements are therefore baselined for change control and any proposed RC is then compared with the baseline for any conflicts. In the third step, a change authority or change decision maker is established. For small projects, this would be a project manager while for larger systems, the responsibility would be handed to a change control board. In both cases, the decision is based on impact analysis. In the decision-making process, it is recommended that input from various stakeholders, such as customers, end-user, developers, testers, etc. should be taken into consideration. To be able to make an informed decision, the impact analysis should capture the effect of the change on cost, functionality, customers and external stakeholders. Also to be considered is the destabilization of the system, which can occur due to the implementation of the change. The decision which is taken should be communicated to all the concerned parties. The fourth step refers to establishing a system that can be used to capture the changes effectively. This could be either paper-based or electronic. The ripple effects of the change are to be managed in a top-down order.

Limitations of the proposal:

According to [81], this process should enable software practitioners to identify changes that are “both necessary and acceptable”. However, it is not mentioned in this work what steps are to be taken to decide if a particular change is both necessary and acceptable. Similarly, no specific details are given as to how to calculate the impact on cost, functionality, customers and external stakeholders. In this sense, these steps only form a basic understanding of what needs to occur in handling a change.

(ii) Proposal: El Emam et al. [80]

This process focuses on the preliminary analysis of change management. Two inputs are considered in order to conduct this process, the technical baseline and any comments made by stakeholders, such as customers, end-users, the development team, etc. The decision-making process involves a change control board as this change management process is prescribed for large systems. The technical baseline is essentially the system requirement specification document. The change management process has the following four phases:

1. Initial issue evaluation
2. Preliminary analysis
3. Detailed change analysis
4. Implementation

In the first step, the comments gathered from the stakeholders are validated and entered into a database as change requests. If a change request addresses a problem that is within the scope of the technical baseline, and has not been addressed before, a change proposal will be generated. In the second step, an analysis plan is formulated which describes the problem of the change proposal in detail. If this plan is approved by a change control board, then many potential solutions will be developed, from which one will be selected for implementation. This solution then needs to undergo further approval. In the third step, the solution approved by the preliminary analysis report is further analysed against the technical baseline to determine the impact on the system in detail and the changes required. In the last step, the technical baseline is modified according to the change proposal and the change request is closed.

Limitations in the proposal:

The use of these steps is limited to large projects. Furthermore, it is not clear on what basis the different alternative solutions are assessed and what exactly is the decision-making process in the second step. Given that this process is conducted at an initial stage of the development process, there is no access to the code. Therefore, a possibility exists that these changes may cause issues at a code level.

(iii) Proposal: Kotonya and Sommerville [82]

The authors emphasize the importance of having a formal process for change management to ensure the proposed changes continue to support the fundamental business goals. They [82] indicate that such a process ensures that similar information is collected for each proposed change and that overall judgements are made about the costs and benefits of such changes. A three-step change management process is proposed in [82] as follows:

1. Problem analysis and change specification
2. Change analysis and costing
3. Change implementation

In the first step, a problem related to a requirement or a set of requirements is identified. These requirements are then analysed using the problem information and as a result, requirements changes are proposed. In the second step, the proposed changes are analysed to determine the impact on the requirements as well as a rough estimation of the cost in terms of money and time that is required to make the changes. Finally, once the change is implemented, the requirement document should be amended to reflect these changes and should be validated using a quality checking procedure.

Limitations in the proposal:

The cost estimation carried out in the second step has a component of seeking customer approval. The information which is lacking at this stage is the decision factors that are considered by the software practitioners and the customers in order to approve or disapprove a proposed change. The negotiation process with customers in relation to accepting or rejecting a proposed change as indicated in [82] is based on cost and there is no indication that the risks associated with implementing the change were considered.

(iv) Proposal: Strens and R. Sugden [35]

The change analysis process introduced in [35] is based on two analysis methods, namely sensitivity analysis and impact analysis. According to [35], sensitivity analysis is used to predict which requirements and design areas have the highest sensitivity to changes in requirements while impact analysis is used to predict the consequences of these changes on the system. The main outcome of this analysis is to reduce the associated risks in accepting and implementing RCs. The process is as follows:

1. Identify the factors which are the cause of change.
2. Identify those requirements which are highly affected by the change (this information is acquired from the previous history of requirements or intuition).
3. Identify the consequences of these changes - impact analysis
4. Undertake change analysis on other requirements, design, cost, schedule, safety, performance, reliability, maintainability, adoptability, size and human factors.
5. Decide on and manage changes.

Limitations in the proposal:

It is important to perform change analysis, however there is no clear explanation as to how the impact analysis is to be carried out for the elements mentioned in step four and how these factors will be "equated". It is also difficult to determine the ripple effect of the changes, given that there is no identification of the implementation part and the test documents to be modified.

(v) Proposal: Pandey et al. [83]

The authors propose a model for software development and requirements managements. There are four phases in this process model: requirement elicitation and development, documentation of requirements, validation and verification of requirements and requirements management and planning [83]. The management of RCs are controlled by the requirement management and planning phase. However, according to the full process model, the activities of this phase are interrelated with the other phases. The process is as follows:

1. Track the changes of the agreed requirements.

2. Identify the relationship between the changing requirements with respect to the rest of the systems.
3. Identify the dependencies between the requirements document and other documents of the system.
4. Decision on the acceptance of the change(s).
5. Validation of change request.
6. Maintain an audit trail of changes.

Limitations in the proposal:

Although a comprehensive set of steps is described, the paper does not discuss specific schematics in executing these steps. Dependencies are considered but there is no indication of further impact analysis. It is not clear how decisions will be made in terms of accepting or rejecting a change as the impact analysis phase is not clearly discussed. There is also no indication of consideration of the cost or risks associated with implementing the change.

(vi) Proposal: Tomyim & Pohthong [36]

The method introduced by [36] for RCM uses UML for object-oriented development. The authors justify the use of UML due to the complexity of the many views and diagrams it produces, thereby adding more complexity in managing change. Therefore, a need arises for a process to manage the changes better using UML. The business model used in this method consists of two procedures: systems procedure (SP) and work instructions (WI). The SP explains the business operation from the beginning of a task until the end of the business process. The WI explain the way to operate any single task step by step. The method comprises the following steps:

1. Identify the change request.
2. Identify the related SP and WI.
3. Analyse the impact on the system and report on the impacted artefacts.
4. Make a decision based on the impact.

Limitations in the proposal:

The paper provides several sets of diagrams that represent the activities carried out but does not provide details of the execution of the steps. A decision on the implementation of the change is solely based on the impact analysis. This may be problematic if change priorities and costs/effort elements are not taken into consideration.

(vii) Proposal: Hussain et al. [84]

The method proposed by [84] is based on the need to manage informal requirements changes. Such requirements are internally focused, potentially subversive to the development process and therefore harder to manage [84]. According to the authors, there are many reasons for informal changes, some of which are: prematurely ending requirement engineering activities [85]; attempting a requirements 'freeze' earlier than usual in a project [81]; as a consequence of work hidden by managers to get something developed by making ad hoc decisions and bypassing time consuming formalities [86]; additions made without the consideration of delay in the schedule and project cost [87]; and failure to create a practical process to help manage changes [81]. Therefore, the authors suggest that there is as much a need for a method for managing informal requirement changes as for formal requirement changes. The method comprises the following steps:

1. Identify informal requirement change.
2. Analyse the impact of change.
3. Negotiate the change with stakeholders.
4. If accepted, decide on whether to include in current phase or next.

Limitations in the proposal:

The process is not very different from formal change management techniques. The negotiation component after the impact analysis is a slight variation from the norm, however it does not explicitly explain how the negotiation is done. The main component considered for negotiation is the impact analysis. However, the proposed method does not disclose how the impact analysis is conducted and what is considered for the impact analysis i.e. affected components, cost, effort, etc.

5.2 Formal process models available for requirements change management

The processes introduced above are not formalized models for managing RC. This section introduces several RCM process models. These models facilitate communication, understanding, improvement and management of RCs. Typically, a process model includes activities, who is involved (roles) and what artifacts are to be used [37, 88].

The activities of a change management process model are the actions performed during the RCM process that have a clearly defined objective, such as determining the change type which is a part of change identification [3, 89, 90]. The identification of the roles in these process models define the responsibilities attached to each role. For example, if the role of the customer is defined by the process model, this means the responsibilities need to be shared with the customer's organization and its representatives. The artifacts are documents and parts of the product created, used and/or modified during the process [3, 89, 90]. By identifying these artifacts as part of the RCM process, this makes the management of change more efficient due to the early detection of what documentation is going to be affected by the change.

Based on the information given in [42] and by individually studying several change management process models, ten such models [1, 45, 81, 91-95] were selected from the literature. Table 6 compares these models based on their activities, roles and the artifacts used. There are certain limitations to these models, which are detailed in Table 7.

5.3 Agile methods available for requirements change management

One of the most important aspects of agile methods is that change is a built-in aspect of the process [96]. Since software development is done in small releases, agile methods tend to absorb RCM into these small iterations. The processes for managing change can neither be categorised as semi-formal nor formal. Because of the frequent face-to-face communication between the development team and the client, the main reported changes in requirements are to add or to drop features [97, 98]. The clarity gained by clients helps development teams to refine their requirements, which results in less need for rework and fewer changes in subsequent stages [98]. There are several agile development models used, the most popular being Extreme Programming, Scrum, RUP, Lean, Plan-driven methods, Iterative &

Incremental model and the General Agile model [99]. Regardless of the agile style of development used, the underlying processes have an inbuilt capacity to manage requirement change. We were able to extract 10 such processes that deal with RCM as follows:

1. *Face-to-face communication [97, 98, 100-102]:*

   This is a frequent characteristic activity between the client and the development team [97, 100, 101]. There is minimal documentation using user stories which does not require long and complex specification documents. The frequency of this activity helps clients to steer the project in their own direction as the understanding of needs tend to develop and requirements evolve [98, 102]. Therefore, the possibility of dramatic and constant changes is reduced and the changes that do arise are easily communicated due to the frequent communication between all the stakeholders.

2. *Customer involvement and interaction [96-98, 101, 103]:*

   In relation to some of the change cause factors listed in RQ1, there are several elements to the involvement of the customer organization. In agile methods, there is a need to identify customers or representatives from the client organization for frequent collaboration to ensure that requirements are appropriately defined [103] [101]. As discussed above, this leads to a better understanding of the system requirements and makes the inclusion of changes less complicated.

3. *Iterative requirements [97, 101, 102]:*

   Unlike traditional software development, requirements are identified over time through frequent interactions with the stakeholders (face-to-face communication) [101]. The frequent interactions make this an iterative process. This allows the requirements to evolve over time with less volatility [97]. This gradual growth of requirements leads to less requirement changes and far less time spent managing such changes.

4. *Requirement prioritisation [98, 101-103]:*

   This is a part of each iteration in agile methods [98]. In each iteration, requirements are prioritised by customers who focus on business value or on risk [101, 103]. In comparison, traditional requirements engineering is performed once before development commences. Iterative requirement prioritisation helps in RCM by comparing the need for the change with the existing requirements and then placing it an appropriate priority location for implementation. As this is done frequently, understanding the need for the change and its priority becomes a much easier process.

5. *Prototyping [97, 101, 104]:*

   This is a simple and straightforward way to review the requirements specification with clients, so that timely feedback is obtained before moving to subsequent iterations [104]. This assists in RCM by identifying what new additions are required and what existing requirements are to be changed or removed. This reduces complex and/or frequent RCs in subsequent iterations.

6. *Requirements modelling [105, 106]:*

   One technique used in requirement modelling in agile methods is goal-sketching, which provides goal graphs that are easy to understand [106]. This activity is also iterative and the goals are refined during each iteration [105]. This helps in RCM by creating unambiguous requirements that have a clear purpose, reducing the need for change during subsequent iterations.

7. *Review meetings and acceptance tests [101, 107]:*

   During review meetings, the developed requirements and product backlogs are reviewed to ensure user stories are completed. Acceptance tests are similar to a unit test, resulting in a “pass” or a “fail” for a user story. These tests increase the collaboration of all the stakeholders as well as reduce the severity of defects. One of the reasons for RC is defects in the end product. This practice effectively reduces the need for changes due to such defects.

8. *Code refactoring [108]:*

   This process is used for revisiting developed code structures and modifying them to improve structure and to accommodate change [109]. This practice deals with requirement volatility in subsequent stages of agile development [108]. Therefore, in terms of RCM, the method allows flexibility in handling dynamically changing requirements.

9. *Retrospective [101, 102, 110]:*

   This process comprises meetings which are held after the completion of an iteration [110]. These meetings often review the work completed so far and determine future steps and rework. In terms of RCM, this provides an opportunity to identify changes.

10. *Continuous planning [102]*

    This is a routine task for agile teams where the team never adheres to fixed plans but rather adapts to upcoming changes from customers. In RCM, this facilitates changing requirements in the later stages of the project.

Agile development, different to traditional software development encourages change in every iteration. The iterative and dynamic nature of this development method promotes constant feedback and communication between the stakeholders. Therefore, the management of changes is continuous during the iterations. We have identified some of the challenges that are inherent in traditional methods of RCM that can be resolved by agile methods. This is discussed in Table 8. Whilst agile methods seem to have a very efficient way of managing change, we were able to identify some practical challenges in some of the techniques discussed above. The challenges are presented in Table 9.

Areas of change management	Model elements	Process models									
	Activities	Leffingwell & Widrig [81]	Olsen [91]	V-Like [92]	Ince’s [92]	Spiral [92]	NRM [1]	Bohner [95]	CHAM [93]	Ajila [94]	Lock & Kotonya [45]
Change identification	Plan of change	Y							Y		
	Problem understanding			Y		Y		Y			
	Determine type of change								Y		
Change analysis	Change impact on functionality	Y		Y						Y	Y
	Manage change hierarchy	Y									
	Solution analysis			Y		Y			Y		Y
Change effort estimation	Change impact on cost	Y							Y		Y
	Estimate effort								Y		
	Cost benefit analysis										Y
Other	Negotiation process	Y							Y		Y
	Update document	Y			Y				Y		
	Change implementation		Y	Y	Y		Y	Y		Y	Y
	Verification		Y				Y		Y		Y
	Validation			Y	Y		Y			Y	Y
	Document impact, cost and decisions										Y
Artifacts		Baseline, Vision document, Use case model, software requirement specification	N/A	Modification report, Problem statement	Problem statement, Change authorization note, Test record	Implementation plan, Release plan	N/A	N/A	N/A	N/A	Vision document, Use case model, software requirement specification, problem statement, change request form
Roles		Customer, developer, end user, change control board	N/A	Maintenance organization	Customer, Developer, Change control board	N/A	N/A	N/A	Customer, Developer, End user	N/A	Customer, Developer, End user

Table 6: Comparison of RCM process models

Model	**Limitations**
Leffingwell & Widrig [81]	Implementation of change is missing. Verification is not available and therefore not able to ensure the stability of the system post-change. Documentation in the form of change requests and decisions are also missing which contributes to poor management and future decision making.
Olsen [91]	Does not explicitly mention if there is any update to documents to keep track of the changes and also, there is no indication of the artifacts used and who is involved in the management process.
V-Like [92]	Two key elements are missing, cost estimation and impact analysis.
Ince's [92]	The decision-making process is unclear. Verification is not done.
Spiral [92]	Similar to Ince's model, there is a lack of decision making and no verification. Does not mention who needs to be involved in this process.
NRM [1]	Activities are at a very abstract level. Given that no artifacts and roles are mentioned, it is difficult to make use of this model in practice.
Bohner [95]	A key element that is missing is the analysis of impact, which is a major part of the decision-making process.
CHAM [93]	Although cost and effort is estimated, there is no analysis of impact on functionality which is an important factor for decision making. The artifacts to be used are also not mentioned.
Ajila [94]	There is no estimation of cost or effort. Artifacts and roles are also not mentioned.
Lock & Kotonya [45]	No aspect of change identification, which is critical in understanding the change.

Table 7: Limitations of RCM process models

Challenges in traditional RCM approaches	**Solutions provided by Agile approaches**
Communication gaps and lack of customer involvement causing ambiguous requirements	Frequent face-to-face communication, customer involvement, and iterative requirements
Changes that occur due to over scoping which is a result of communication gaps and changes after finalizing project scope	Continuous customer involvement, iterative requirements, and prototyping
Change validations	Requirement prioritisation through iterative processes, prototyping, and review meetings and acceptance tests

Table 8: Challenges in traditional RCM resolved by Agile approaches

Agile technique	**Challenges**
Face-to-face communication	The frequency of the communication depends on the availability and willingness of the team members. Customers may not be familiar with this agile technique and could be wary of it.
Customer involvement	Failure to identify needed/correct customer representatives can lead to disagreements and changing viewpoints.
Requirement prioritisation	A focus only on business value when prioritising requirements/changes can be problematic as there can be other factors to consider.
Prototyping	Problems may occur if there a high influx in client requirements at a particular iteration.
Code refactoring	Can generate code wastage, which increases the project cost.
User stories and product backlog	This is the only documentation used in agile methods as minimal documentation is a characteristic. This becomes a problem when there is a communication lapse or project representatives are unavailable. It is also problematic when requirements must be communicated to stakeholders in distributed geographical locations.

Budget and schedule estimation	Due to the nature of incorporating RCs in subsequent iterations, it is not possible to make upfront estimations, which can result in budget and schedule overruns.

Table 9: Challenges in Agile RCM

Key findings of RQ$_2$

Similar to any other activity in the software development process, RCM has also been described in related work as an activity that needs to be carried out in defined steps. Based on the discussion that formulates the answer for RQ2, the following are the key findings:

1) Academic work has identified that it is important to establish a process for managing change where establishing and practicing a defined process for RCM is attached with benefits, such as the improvement of organizational processes and an increase in the predictability of projects.
2) In terms of traditional software development, two different approaches were investigated, namely: 1) recommendations for semi-formal methods of managing change; and 2) the formal process models available for RCM.
3) With semi-formal methods, it became evident that different academic work took different approaches and elements, and recommended different steps for managing change, which resulted in no consensus on the elements.
4) However, based on the activities on which the elements focused, we were able to identify three areas of management: change identification; change analysis; and change effort estimation.
5) These three areas were then applied to the ten formal process models of RCM found in the literature. Using this classification, we were able to identify certain commonalities between the process models, as illustrated in Table 6.
6) The formal process models have three distinct sections: activities – the actions / steps taken in managing change; roles – the stakeholders involved in carrying out the activities; and artifacts – the documents needed in some of the activities (see Table 6).
7) We were also able to identify the limitations in both the semi-formal methods as well as the formal models.
8) Given the popularity of agile development in the recent past and present, several processes were identified that deal with RCM. Through this identification, we were able to discuss how agile methods can address some challenges in traditional RCM and also the challenges in agile RCM.

6. Results for RQ3: What techniques are used for requirements change management?

The information gathered in RQ_2 will be used to formulate a framework to answer this question. Examining the processes introduced in RQ_2 as a whole, we have identified three key areas of a practical approach to managing change. Figure 1 illustrates these areas i.e. change identification, change analysis and change cost estimation. It is important to understand how these areas can be practically implemented and what best practices are available in an organizational setting. As shown in Figure 1, none of these areas are standalone. They need to communicate with each other in terms of updates and verifications. The reason for this is that each area has the ability to feed information to another area. For example, although change analysis can be undertaken once the change has been identified, the cost estimation may provide additional information for the analysis step that may not have been identified previously. A good RCM process does not have steps that are stand alone, rather they are interconnected with information following to and fro from the steps. We used the search expressions A_4 OR A_5 OR A_6 OR A_7 OR A_8 OR A_9 OR A_{10} OR A_{11} (see Table 1) to extract relevant literature.

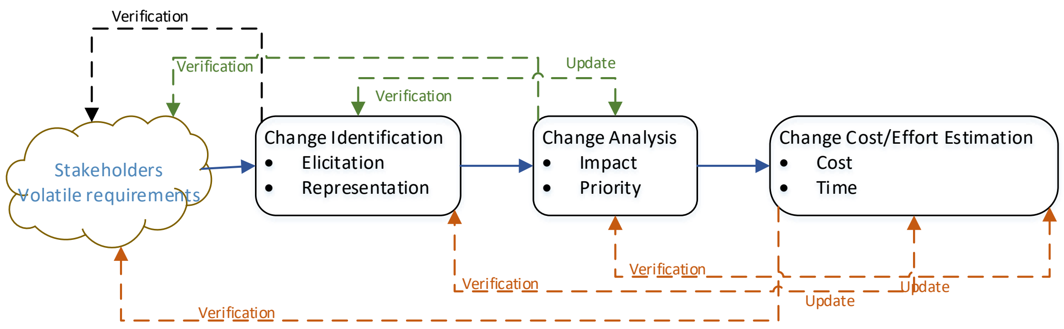


Figure 1: Change management process

6.1 Change Identification

Change identification stems from several processes identified in RQ_2 [80-82]. This step is important for the rest of the management process as the steps to follow will be based on the correct identification of the problem space as well as the change requirement. According to Figure 1, the change management process starts with change identification. Within this identification, there are two major activities, i.e. change elicitation and change representation.

In order to ensure the correct elicitation of changes, the change requirements need to be identified.

The correct elicitation should then lead to identifying further details of the change and if possible, where in the system the change has to be made. This signifies the representation part of the identification step. In most situations, the personnel involved in this step will need to have continuous communication with the stakeholders in order to verify that identification is done correctly, as illustrated in Figure 1. Through the literature, we identified two methods of change identification: taxonomies and classification. The following sections describe these two methods and several other methods that do not fall under these categories.

a) Through taxonomies

1) Research analysing change uses a plethora of techniques in order to build a taxonomy that can be used to identify changes as well as their impact. One such mechanism is the use of requirement engineering artifacts, such as use cases. The research done by Basirati et al. [14] establishes a taxonomy of common changes based on their observation of changing use cases that can then be used in other projects to predict and understand RCs. They also contribute to this research space by identifying which parts of use cases are prone to change as well as what changes would create difficulty in application, contributing also to the impact analysis of change.

2) The taxonomy developed by Buckley et al. [15] proposes a software change taxonomy based on characterizing the mechanisms of change and the factors that influence software change. This research emphasizes the underlying mechanism of change by focusing on the technical aspects (i.e. how, when, what and where) rather than the purpose of change (i.e. the why) or the stakeholders of change (i.e. who) as other taxonomies have done. This taxonomy provides assistance in selecting tools for change management that assist in identifying the changes correctly.

3) McGee and Greer [16] developed a taxonomy based on the source of RC and their classification according to the change source domain. The taxonomy allows software practitioners to make distinctions between factors that contribute to requirements uncertainty, leading to the better visibility of change identification. This taxonomy

also facilitates better recording of change data, which can be used in future projects or the maintenance phase of the existing project to anticipate the future volatility of requirements.

4) Gosh et al. [17] emphasize the importance of having the ability to proactively identify potentially volatile requirements and being able to estimate their impact at an early stage is useful in minimizing the risks and cost overruns. To this effect, they developed a taxonomy that is based on four RC attributes i.e. phases (design, development and testing), actions (add, modify and delete), sources (emergent, consequential, adaptive and organizational) and categories of requirements (functional, non-functional, user interface and deliverable).

5) The taxonomy established by Briand et al. [18] is the initial step in a full-scale change management process of UML models. In their research, they establish that change identification is the first step in the better management of RCs. The classification of the change taxonomy is based on the types of changes that occur in UML models. They then use this taxonomy to identify changes between two different versions of UML models and finally to determine the impact of such changes.

b) Through classification

There are many benefits of using a classification, the main benefits being to manage change to enable change implementers to identify and understand the requirements of change without ambiguity [19, 111]. The classification of RC has been studied in various directions. Table 10 lists the different directions that have been the subject of academic studies.

Direction	**Parameters**	**Comment**
Type [17, 19-23]	Add, Delete, Modify	The most common way of classifying change.
Origin [10, 17, 112]	Mutable, Emergent, Consequential, Adaptive, Migration	Derived from the places where the changes originated from.
Reason [12, 19, 20]	Defect fixing, Missing requirements, Functionality enhancement, Product strategy, Design improvement, Scope reduction, Redundant functionality, Obsolete functionality, Erroneous requirements, Resolving	Helps determine the causes of change and understand change process and related activities.

	conflicts, Clarifying requirements, Improve, Maintain, Cease, Extend, Introduce	
Drivers [113]	Environmental change, RC, Viewpoint change, Design change	Helps change estimation and reuse of requirements.

Table 10: Direction is change classification

c) Other change identification methods

1) Kobayashi and Maekawa [1] proposed a model that defines the change requirements using the aspects where, who, why and what. This allows the system analyst to identify the change in more detail, resulting in better impact identification as well as risk and effort estimation. This method consists of verification and validation and can be used to observe the RCs throughout the whole lifecycle of the system.

2) The change identification method usually has a pre-established base upon which its semantics are built. Ecklund's [114] approach to change management is a good example of this. The approach utilizes use cases (change cases) to specify and predict future changes to a system. The methodology attempts to identify and incorporate the anticipated future changes into a system design in order to ensure the consistency of the design.

d) Change identification through agile methods

Unlike traditional requirement engineering methods, agile software development welcomes changes in various stages [98]. As discussed in RQ2, changes can be identified in several different phases of the development process. Table 11 presents the different phases of agile development that contribute to the identification of RCs, the challenges faced and solutions suggested by literature. The techniques given in the table have been described in detail in RQ2 (see section 5.3).

Agile technique	**Challenge(s)**	**Solutions**
Face-to-face communication [97, 98, 100-102]	The success rate of the change identification at this stage is dependent on customer availability. However, this dependency is often unrealistic and a challenge as confirmed by studies [101, 115]	In practice, teams have surrogates or proxy customers to play the role of real customers [103] or use the "onsite developer" by moving a developer representative to the customer site [116].

Iterative requirements [97, 101, 102]	Can create budget and schedule overruns as initial estimations will always change when requirements are added or removed during the iterations [101].	Inayat et al. [98] suggest frequent communication to identify as many requirements as possible at early iterations to keep these overruns to a minimum.
Prototyping [97, 101, 104]	Given that this is a review phase of development, the client may have a large number of changes to be included based on the prototype. This can create schedule overruns [98].	This can be mitigated somewhat, through frequent communication and high customer involvement and interaction in stages prior to prototyping [98].
Review meetings and acceptance tests [101, 107]	Similar to the challenges of prototyping where there could be an influx of changes [107]. Also, if the product backlog is not maintained in detail, finding information related to changes made during the iterations will also be challenging.	Denva et al. [103] suggest maintaining a detailed artefact called delivery stories, in addition to user stories. These help developers make the right implementation choices in the coding stage of a sprint.
Retrospective [101, 102, 107]	If there are many changes identified in completed user story at this stage, there will be a considerable amount of rework to be done, causing budget and schedule overruns [98].	Increased customer involvement and interaction in the stages prior to completion of a user stories is essential [98].

Table 11: Change identification through agile methods

6.2 Change Analysis

Once a change has been identified, it needs to be further analysed to understand its impact on the software system so that informed decisions can be made. One of the key issues is that seemingly small changes can ripple throughout the system and cause substantial impact elsewhere [117]. As stated in the literature, the reason for such a significant impact is that the requirements of a system have very complex relationships [24-28]. Therefore, the way to realise this is to undertake change impact analysis, which according to [118] is defined as "the activity of identifying the potential consequences, including side effects and ripple effects, of a change, or estimating what needs to be modified to accomplish a change before it has been made". Change impact analysis provides visibility into the potential effects of the proposed changes before the actual changes are implemented [117, 118]. The ability to identify the change impact or potential effect will help decision makers to determine the appropriate actions to take with respect to change decisions, schedule plans, cost and resource estimates.

a) Traceability issues and solutions

Given that the complex relationships between requirements are the key reason for impact analysis, most methods for impact analysis use requirement traceability as their focal point. Requirement traceability is defined as "the ability to describe and follow the life of a requirement in both a forward and backward direction (i.e. from its origins, through its development and specification to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases)" [119]. Although traceability has been defined by many scholarly articles, the above definition was selected as the most comprehensive because it describes both pre- and post-traceability and is used by many other scholarly articles [120-129] for the same purpose.

Although traceability is one of the best ways to track the impact of RCs, many scholarly works discuss the challenges in maintaining traceability. Tables 6 and 7 detail the issues in traceability and the solutions that have been provided. The solutions in Table 12 have not been verified by industry while the solutions in Table 13 have.

Scholarly work	**Issues in traceability**	**Solution (Not verified by industry)**
Arkley & Riddle [130]	Requirement traceability does not offer immediate benefit to the development process.	Traceable development contract.
Cleland-Huang, Chang, Christiensen [131]	Informal development methods, insufficient resources, time and cost for traceability, lack of coordination between people and failure to follow standards.	Event-based traceability
Cleland-Huang, Zemont & Luasik [132]	Lack of coordination between team members. Developers think that traceability costs more than it delivers. Excessive use of traceability generates more links which are not easy to manage.	Traceability for complex systems frameworks.
Cleland-Huang, Settimi, Duan & Zou [133]	Manual construction of a requirement traceability matrix is costly.	Dynamic retrieval methods are used to automate the generation of traceability links
Gotel & Morris [134]	Requirements change by user. Less appropriate information is available for making decision with requirements.	Media recording framework.
Ravichandar, Arthur & Pérez-Quiñones [127]	Problems associated with tracing back to their sources.	Pre-requirements traceability technique.

Table 12: Traceability issues and their solutions (not verified)

Scholarly work	**Issues in traceability**	**Solution (Verified by industry)**
Blaauboer, Sikkel & Aydin [135]	Adopting requirement traceability into projects.	Increase awareness and adapt organizations to include requirement traceability.
Cleland-Huang [136]	Failure to trace non-functional requirements e.g. security, performance and usability	Goal centric traceability evaluated by an experiment
Gotel & Finkelstein [119]	Some problematic questions are identified as challenges: Who identifies a requirement and how? Who was responsible for the requirement to start with and who is currently responsible? Who is responsible for change(s) in requirements? What will be the effect on the project in terms of knowledge loss if key employees quit?	Framework of contribution structure.
Heindl & Biffl [124]	Cost related to requirement traceability.	Value-based requirements tracing tested through a case study.
Ramesh [137]	Organizational, environmental and technical factors.	Best practice given.
Verhanneman, Piessens, De Win & Joosen [129]	Requirement management challenges in industry projects e.g. inadequate impact analysis and lack of information transfer.	Requirement management tools like DOORS and RequisitePro.

Table 13: Traceability issues and their solutions (verified)

It is important to note that the solutions proposed might not be suitable for all types of organizations, however, some basic guidelines can be outlined.

i. The identified issues can act as a guideline to understand the challenges that might arise when creating and maintaining traceability and therefore improve the predictability of the traceability issues.

ii. The cost of traceability for a specific project will be concentrated on that project whilst its benefits (value) will span over and beyond the said project. The downside of this outcome is that it may hinder the motivation of a project team to work with traceability as the benefits are not realized immediately and therefore could be the cause of many of the challenges identified in Table 6 and 7.

b) Use of Traceability and other methods for impact analysis

According to Figure 2, there are three sets of objects that can be impacted by a change: starting impact set (SIS), estimated impact set (EIS) and actual impact set (AIS).

- SIS is the set of objects that are thought to be initially impacted by the change
- EIS is the set of objects estimated to be impacted after further analysis

- AIS is the set of objects that are actually modified as a result of the change

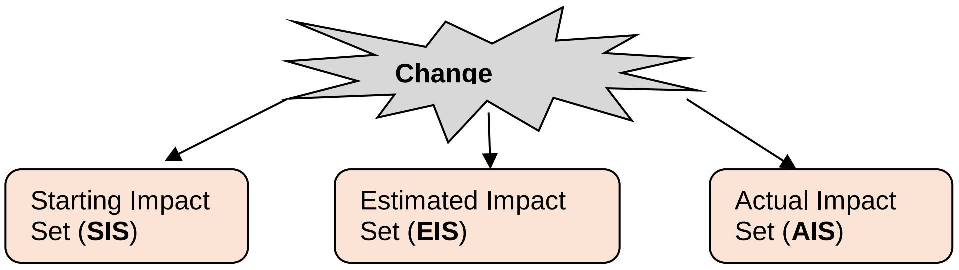


Figure 2: Change impact object sets

This is a concept introduced by Arnold and Bohner [138]. We identified in the literature several impact analysis techniques that use traceability and non-traceability methods. These methods were subject to the concept introduced by [138] to identify which set of objects are analyzed and are detailed in Tables 14 and 15. This finding benefits software practitioners in selecting a potential method for change analysis based on the set of objects on which they want to focus. Table 14 details solutions that use traceability techniques to analyse RC while Table 15 details solutions that use other techniques.

Scholarly work	**Title of work**	**Solution (Using Traceability)**	**Impacted objects**
Antoniol et al. [139]	Identifying the impact set of a maintenance request	The tracing is done at a coding level where the text in the maintenance request is mapped to development code components corresponding to the change request.	SIS
Li et al. [140]	Requirements-centric traceability for change impact analysis	The method uses an interdependency graph and traceability matrix to assess the impact at a requirement specification level.	SIS, EIS and AIS
Ibrahim et al. [141]	Integrating software traceability for change impact analysis	The method provides a holistic traceability solution that involves both high level and low level software models ranging from requirements to code.	AIS
Göknil et al. [142]	Change impact analysis based on formalization of trace relations for requirements	The method deals with a requirements metamodel with well-defined types of requirements relations, which are used to define change impact rules for requirements. These rules help identify the impacted requirements.	EIS and AIS
Von Knethen [143]	Change-oriented requirements traceability. Support for evolution of embedded systems	The approach consists of three parts, a conceptual trace model for embedded systems, rules to establish traces and analyse impact and a tool for semi-automatic impact analysis and consistency checking.	SIS and AIS

Table 14: Techniques used for impact analysis – Traceability methods

Scholarly work	**Title of work**	**Solution (Using Non-Traceability methods)**	**Impacted objects**
Kobayashi & Maekawa [1]	Need-based requirements change management	The method captures RC using the 4Ws: where, who, why and what. The solution mainly consists of verification and validation activities.	SIS
Ali & Lai [144]	A method of requirements change management for global software development	The method consists of three stages: understanding change, analyzing these changes and finally making decisions regarding the change based on the analysis.	SIS
Hassine et al. [145]	Change impact analysis for requirements evolution using use case maps	Method uses slicing and dependency analysis at the use case map specification level to identify the potential impact of RCs on the overall system.	SIS
Briand et al. [18]	Impact analysis and change management of UML models	The method uses a UML model-based approach where the UML diagrams are first checked for consistency. The impact analysis is carried out using a change taxonomy and model elements that are directly or indirectly impacted by the changes.	SIS and EIS
Hewitt & Rilling [146]	A Light-Weight Proactive Software Change Impact Analysis Using Use Case Maps	The method seeks to predict impact of changes at a specification level. The method focus on extracting information from Use Case Maps (UMC) that can be used for proactive change impact analysis at the specification level.	SIS

Table 15: Techniques used for impact analysis – Non-Traceability methods

c) *Predicting requirements changes*

Another aspect of analysing change is to proceed beyond the existing change impact and to use historical data, design diagrams, codes, etc. to predict where change may occur and identify their impact. Based on this concept, we were able to extract literature that discusses the prediction of RCs, their possible impact on the systems and how the change may propagate through the system. These findings are important in order for development teams to foresee how to be prepared for RCs, make better decisions and better implement such changes. We present the prediction methods and their limitations in Table 16.

Title	**Solution**	**Limitations**
1. Learning from Evolution History to Predict Future Requirement Changes [147]	Method uses historic information to develop a metrics that measures the evolution history of a requirement. Based on the metrics, the method proposes to reduce the impact of requirements evolution by attempting to predict requirements that are prone to change in the future.	Can only be applied to projects that have historic data. Some important requirements changes may be neglected by the prediction method.

2. Managing Changing Compliance Requirements by Predicting Regulatory Evolution [148]	Method uses an adaptability framework which helps requirements engineers to identify: why requirements change (rationale); how requirements change (classifications); and which portions of a proposed rule are most likely to change when the final rule is issued (heuristics). The framework allows engineers to focus primarily on analysing and specifying compliance requirements from the more stable areas of the laws, while the less stable areas of the laws are clarified during the final rulemaking.	The study uses two case studies from the healthcare industry and therefore the findings and applicability remain limited to the healthcare industry.
3. Mining the Impact of Object-Oriented Metrics for Change Prediction using Machine Learning (ML) and Search-based Techniques (SBT) [149]	This method is used to identify the probability of classes that would change (change proneness of a class) in the subsequent release of software. The study develops a relationship between Object- Oriented metrics and the change proneness of a class. The method evaluates the effectiveness of six SBT, four ML techniques and the statistical technique - Logistic Regression (LR) on change proneness prediction data and compares their results.	Findings and applicability limited to object-oriented environments.
4. Using Early Stage Project Data to Predict Change-Proneness [150]	This paper presents a feasibility study undertaken to test the validity of a hypothesis that data from requirements and design activities may also prove to be useful in predicting change proneness. A metrics is developed for quantifying requirements and design activities. Next, values are generated for these metrics from a real-world case study and finally a comparison is made with the actual number of changes detected.	Method can only be applied if the project has requirements and/or design information available. Clearly, this creates a limitation for approaches such as agile methods that have limited documentation.
5. Predicting the Probability of Change in Object-Oriented Systems [151]	This is a probabilistic approach to estimate the change proneness of an object-oriented design by evaluating the probability that each class of the system will be affected when new functionality is added or when existing functionality is modified. The goal is to assess the probability of how each class will change in a future generation.	Previous versions of a system must be analyzed to acquire internal probability values creating scalability problems for large systems. Cannot be applied in the initial stages of the development process (e.g. at the design level).
6. Using Bayesian Belief Networks to Predict Change Propagation in Software Systems [152]	The approach seeks to predict the possible affected system modules, given a change in the system. The method is composed of two steps: extracting information and predicting changes. In the first step, the authors extract the system elements' dependencies and change history. In the second step, the Bayesian Belief Networks are built using	Can only be applied to methods that have historic data and documentation.

	the extracted information and then predictions are produced using probabilistic inference.	

Table 16: Methods of predicting requirements changes

d) Change analysis using agile techniques

In agile development, requirements engineering activities are not explicit. Partially, this is due to the fact that there are less distinct boundaries in agile development than in traditional software development [153]. Therefore, similar to change identification, the analysis of RCs in agile development is not restricted to a particular phase of the development but a mixture of techniques is used that occur iteratively. The agile techniques discussed in RQ2 (section 5.3) are detailed in Table 17 to show how change analysis is carried out in agile development.

Agile technique	**How change analysis is done**
Iterative requirements [97, 101, 102]	The requirements related to a user story are not identified at the beginning of a project. Requirements are built on iterations, which allow stakeholders to gain a better understanding of what is required, therefore analyse, and understand the need for changes.
Requirement prioritisation [98, 101-103]	In each of the iteration, the identified requirements are prioritised. This means that any changes that occur during the iterations will be compared to existing requirements and will be assigned a place in the hierarchy of implementation. The iterative nature of this activity ensures the priority of requirements remain current.
Prototyping [97, 101, 104]	This allows the agile team to review the requirement specifications with clients to obtain feedback. The process will highlight issues with the changes identified so far and will prompt the development team to find better solutions.
Testing before coding [97, 101, 102, 154]	The development team writes tests prior to writing functional codes for requirements. This promotes identification test failure, which can be a form of validation of the changes that have been applied during the iterations.
Requirement modelling [105, 106]	A technique used in modelling in agile approaches is goal-sketching [106]. The outcome is an easy-to-read goal graph, which allows all stakeholders to refine the goals, making them well defined. Changes that are introduced in the iterations can be mapped to goals and this can help with decision making in the implementation of changes.
Review meetings and acceptance tests [101, 107]	The developed requirements and product backlogs are reviewed to identify if user stories have been completed. In terms of change analysis, this evaluates if changes have been implemented correctly and satisfy the end goal.
Regression testing [155]	Regression testing is done in agile methods to make sure that the newly incorporated changes do not have side effects on the existing functionalities and thereby finds the other related bugs. This is a form of change validation in terms of change analysis.

Table 17: Change analysis using agile methods

Two of the documents used in agile development that are worth mentioning are user stories and product backlog, which form a critical part of the change analysis process. User stories are created as the specification of the customer requirements. They facilitate better communication and unambiguous understanding between all stakeholders [103]. User stories are made up of three components: a written description, conversations, and tests [156]. They are meant to reduce the need for constant requirement change and also act as a reference point to check if changes are implemented to satisfy the client requirements. Product backlog keeps track of the details of all the developed requirements. This is one of the documents that can be used to keep track of all the requirements changes [101].

6.3 Change Cost/Effort Estimation

Software cost/effort estimation is referred to as the process of predicting the effort required to develop a software system [157, 158]. It is noteworthy that although effort and cost are closely related, they are not a simple transformation of each other [157]. Effort is often measured in person-months of the development team whilst cost (dollars) can be estimated by calculating payment per unit time for the required staff and then multiplying this by the estimated effort [157]. Cost estimation is usually carried out at the beginning of a project but as we have demonstrated, changes to the system can occur at any stage of the project. Therefore, there is a need to estimate the additional cost for implementation of the change.

There are some basic factors to be considered when estimating, regardless as to whether it is for the entire project or just for a change. The first step in cost/effort calculation is the calculation of the size of the software, which is considered to be the most important factor affecting estimation [157]. Therefore, it is essential to understand the popular software sizing methods used and their suitability for estimating the cost/effort of implementing requirements changes, as shown in Table 18.

Sizing Technique	**Feature**	**Suitability for change cost/effort calculation**
Lines of Code (LOC) [157, 159]	Based on the number of lines of the delivered source code of software. Programming language dependent. Widely used sizing method.	Exact LOC can only be obtained after the completion of the project and is therefore not suitable for changes at the early stage of the design. Also depends on expert judgement and can compromise reliability.

		Can be used for changes that occur towards the latter part of the development process.
Software science [160]	Based on code length and volume metrics. Code length is the measurement of the source code program length and volume is the amount of storage space required.	There have been disagreements over the underlying theory and therefore reliability is questionable [161, 162]. Not suitable for changes in the early phase (reason as above). Possibility of using this in the latter stages, yet the measure has received decreasing support [157].
Function points [163]	Working from the specification, systems functions are counted (inputs, outputs, files, inquiries, interfaces) These points are then multiplied by their degree of complexity.	Use of the specification makes it suitable to analyse changes in the early phase of development. Equally suitable for changes in the latter stages.
Feature point [164]	Extension of function points to include algorithms as a new class.	Similar usability as function points and suitable systems with little input/output and high algorithmic complexity.

Table 18: Popular software sizing techniques

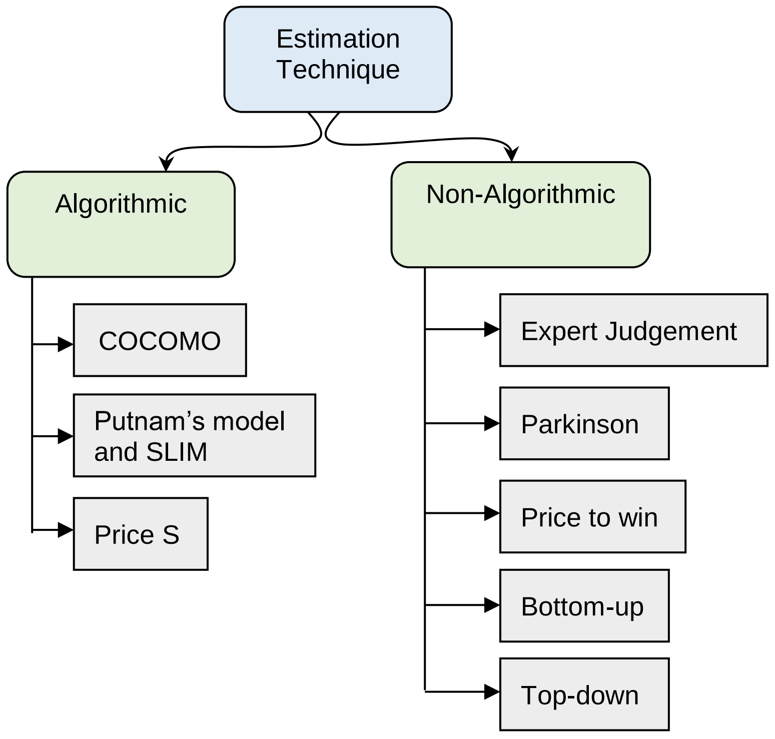


Figure 3: Costing Techniques

There are many methods described in the literature that are popular techniques for estimating cost/effort. As presented in Figure 3, we considered the more frequently used estimation methods in traditional software development and they can be classified into two categories:

algorithmic and non-algorithmic [157, 165]. Algorithmic models can be quite diverse in the mathematical expressions used. It is important to remember that these algorithmic models need to be adjusted to suit the local environment. Regardless of the technique used, none of the methods discussed in this section can be used off-the-shelf.

One of the key findings in this section is to identify the appropriateness of these methods for estimating the cost/effort of implementing RCs. Tables 19 and 20 describe several popular estimation techniques that belong to these two categories and their suitability for change cost estimation.

Category	*Non-Algorithmic*		
Technique	**Features**	**Challenges**	**Suitability for change cost/effort estimation**
Expert judgment	Based on one or more experts using their experience and techniques such as PERT or Delphi for estimation.	Dependency on experts, where human error is a major risk and there can be bias.	Can be suitable since the method is fast and can easily adapt to diverse circumstances. But the limitation carries a lot of risk.
Parkinson	Cost is determined (not estimated) by the available resources rather than an assessment of the entire situation.	Can provide unrealistic estimations and does not promote good software engineering practice.	Given the limitations far exceed its functionality, it cannot be recommended.
Price to win	Estimated to be the best price to win a project. Estimate is based on customer budget.	Not good software practice as software functionality is not considered. Can produce large overruns.	Software functionality is a key factor in change cost estimation and therefore is not suitable.
Bottom-up	Each component of the system is estimated separately and the result is combined to produce the overall estimate. Based on initial design.	Requires more effort and can be time consuming.	Can be suitable for changes in the latter phase. Not suitable for changes in the early phases as it requires detailed system information.
Top-down	The opposite of the bottom-up approach. This is an overall estimation based on global properties. Total cost can be split among the various components.	Less stable as the estimation does not consider different components.	Useful for changes in the early stages. Changes in the latter phases require more detailed costing and therefore it is not suitable.

Table 19: Popular estimating techniques – Non-Algorithmic

Category	*Algorithmic*		
Technique	**Features**	**Challenges**	**Suitability for change cost/effort estimation**
COCOMO	Uses power function models where $Effort = a \times S^b$ S is the code size and a, b are functions of other cost factors.	Not suitable for small systems.	Exact code size can only be obtained at the completion of a project and therefore may not be suitable for changes at early stages.
Putnam's model and SLIM	Equation used $S = E \times (Effort)^{1/3} {t_d}^{4/3}$ where S is LOC, t_d is delivery time, E is environment factor (based on historical data)	Based on information from past projects and may not be suitable for the current environment.	Although generally suitable for changes in cost estimation, dependency on historical data can make the accuracy questionable.
Price-S	This is a proprietary estimation model. Uses an estimate of project size, type and difficulty and computes cost and schedule.	Because it is company specific, it may not suitable for all environments.	Not suitable for change cost estimations due to limitations.

Table 20: Popular estimating techniques – Algorithmic

Effort estimation is more challenging in the agile context as requirement changes are embraced through multiple iterations of development. In line with the previous two sections, we consider the techniques used in agile development for effort estimation. Table 21 details the techniques, the challenges and the suitability for change cost/effort estimation.

Category	*Agile*		
Technique	**Features**	**Challenges**	**Suitability for change cost/effort estimation**
Expert judgment [166, 167]	Developers look to past projects or iterations, and draw on their own experiences to produce estimates for the user stories.	Dependency on experts, where human error is a major risk and there can be bias.	Can be suitable since the method is fast and can easily adapt to diverse circumstances. But the limitation carries a lot of risk.
Planning poker [168, 169]	Once the user stories have been understood, all the team members of the agile team make independent estimates and reveal their estimates simultaneously. The lowest and highest estimates need to be justified by their estimator. The group continues the discussion in order to decide on a collective estimate, possibly by conducting one or more additional rounds of individual estimating.	If the estimation process is unstructured, factors such as company politics, group pressure, anchoring, and dominant personalities, may reduce estimation performance.	Similar suitability as expert judgment but is still dependent on the skill and experience of the team members.

Use Case Points (UCP) [170, 171]	Once the use cases are identified based on the user stories, UCPs are calculated based on the number and complexity of use cases and actors of the system, non-functional requirements and characteristics of the development environment. The UCP for a project can then be used to calculate the estimated effort for a project.	UCP method can be used only when the design is done using UML or RUP.	Can be suitable for an early stage change estimation of the development process. Changes in the latter phases require more detailed costing and therefore it is not suitable.
Story points [172-174]	Story point is a measure for relatively expressing the overall size of a user story or a feature. A point is assigned to each user story. The value of the story point is dependent on development complexity, the effort involved, the inherent risk and so on.	Story points create lots of vagueness to the agile process. For every team, story size could mean different things, depending on what baseline they chose. If two teams are given the same stories, one team can say their velocity is 46 and the other can say 14, depending on what numbers they chose. Story points do not relate to hours.	May only be suitable for teams that are collocated, based on the challenges of the method. Also, it may not be suitable for effort calculation in hours as it will take additional calculations to convert story points to hours.

Table 21: Popular estimating techniques – Agile

Key findings of RQ$_3$

The majority of the academic work on RC is focused on devising solutions for the different areas of RCM. Based on the discussion that formulated the answer for RQ$_3$, the following are the key findings:

1) Change identification methods do not seem to have much consensus on how the identification should be done nor are many of the methods formal.
2) Most change identification methods found are based on two techniques: through taxonomies and through classifications.
3) The change taxonomies tend to be based on larger concepts such as use cases and UML models whilst change classifications use more simplified mechanisms such as change directions and parameters.
4) Change identification usually leads to understanding of the need for the change, which also relates to further analysis of the change.
5) Traceability techniques have been the more popular choice when analysing change as requirement traceability facilitates the identification of the impact of change more

efficiently. However, this seems to be a theoretical concept as requirement traceability has many limitations.

6) The main idea of change analysis is to identify how the requested change impacts the existing design or system. To this effect, methods of change impact analysis found in literature can be grouped based on objects that are impacted: starting impact set, estimated impact set and actual impact set.
7) In terms of the agile context, changes in requirements are expected and welcome aspects of development. As we discovered in the literature, change identification and analysis tend to happen at almost all parts of the iterative process in development.
8) Due to the change-susceptive nature of agile development, unlike traditional development, in most cases change identification and analysis does not require special processes but are embedded into the processes that are part of the development cycle.
9) Costing techniques dedicated for estimating the cost of RC seem to be rare. In most cases, existing costing techniques such as COCOMO, expert judgement, etc. are used for this purpose.
10) It is possible to divide existing costing techniques into two categories: algorithmic and non-algorithmic.
11) Depending on which point of the lifecycle the software project is and what artefacts are used for the cost estimation, each estimation can be judged for suitability to be used for cost estimation of RCs.
12) Some methods can be used but with many risks (i.e. expert judgement), some methods can be used for changes introduced in the latter phase of the project life cycle (i.e. bottom-up, COCOMO, etc.), some methods can be used for changes introduced in the early phase of the project life cycle (i.e. top-down) and some other methods are not suitable for change cost estimation (i.e. price to win, Price-S, etc.).
13) Unlike change identification and analysis, cost/effort estimation in agile development requires special attention. The nature of agile development tends to discover requirements through several iterations and therefore, any estimations at the beginning of a project change significantly along the development cycle. Given this criterion, special techniques are required for the estimation of cost and effort, which, we discovered in the literature, are mostly dependent on expert judgement and team collaboration.

7. Results for RQ_4: How do organizations make decisions regarding requirements changes?

An organization has a harmonious existence when coordination and integration between business objectives and IT services and infrastructure in realizing the common business goals are in alignment [175-177]. However, when managing RCs of system software or software projects, stakeholders may perceive different end goals at different levels of the organization [178]. In other words, change management and analysis plans and strategies vary with organisational level, where each strategy tends to have different goals and objectives. An organization can be categorized into two parts: business organization and IT organization and each of these two categories can be split into three levels, as illustrated in Figure 4. We used the search expressions A_3 OR A_6 OR A_8 (see Table 1) to extract the relevant literature.

(i) Executive level

Once the need for a change in a software process or requirement arises, the top level management (CEO, CIO, etc.), which is the executive level, formulates very broad strategies for managing the said change. The tendency to create broad plans is usually due to the responsibilities of the top level executives in terms of what the organization as a whole stands to gain by implementing these changes [178]. In some instances, business and IT tend to have a contradictory understanding of the need for change. Decisions by the IT side for obtaining new technology that is required for implementation of the change may not always be agreed upon by the business counterparts of an executive level [82, 178]. Research has demonstrated that when business and IT top management fail to understand the need for the change and the IT capabilities that are required for its realization, these software projects tend to have unsatisfactory outcomes in the form of cost overruns and failure [44, 176, 178, 179].

(ii) Tactical level

The tactical level in Figure 2 corresponds to the change management plans and strategies formulated by the middle management of an organization. These strategies can be referred to as functional strategies. The main concern at this level is to assess the change with respect to cost and benefits and find ways to introduce the change without adversely affecting the project

[3, 46, 82, 178]. The broad strategies at an executive level may not always match with the strategies formulated at a tactical level. For example, the end goal of a change at an executive level could be to improve quality while at a tactical level, the goal would be to complete the project successfully and therefore, may consider the change intrusive [82, 178]. It is also noteworthy that the notion of business vs. IT mindset exists at this level too. One of the key barriers in creating a cohesive change strategy between business and IT at this level is due to interpretation and communication barriers that stem from the lack of a common change specification technique [10-13].

(iii) Operational level

As the strategies flow down the organizational structure, they tend to become less complicated and less abstract. At this stage, it becomes a process of understanding the strategies laid down by the tactical level and formulate plans as to how to best implement them. The goals at this level are more short-term due to the fact that development teams are dealing with simpler strategies. Provided that business and IT change strategies at this level are aligned, the combination of such short term strategies could be linked back to the business objectives set at the executive level [180]. Moreover, it is essential at this level that development teams are able to cope with the changes in the business strategies originating at a higher level. Therefore, strategies formulated at an operational level should incorporate a mechanism to deal with such changes that will ensure the final product is what is expected by the executive level.

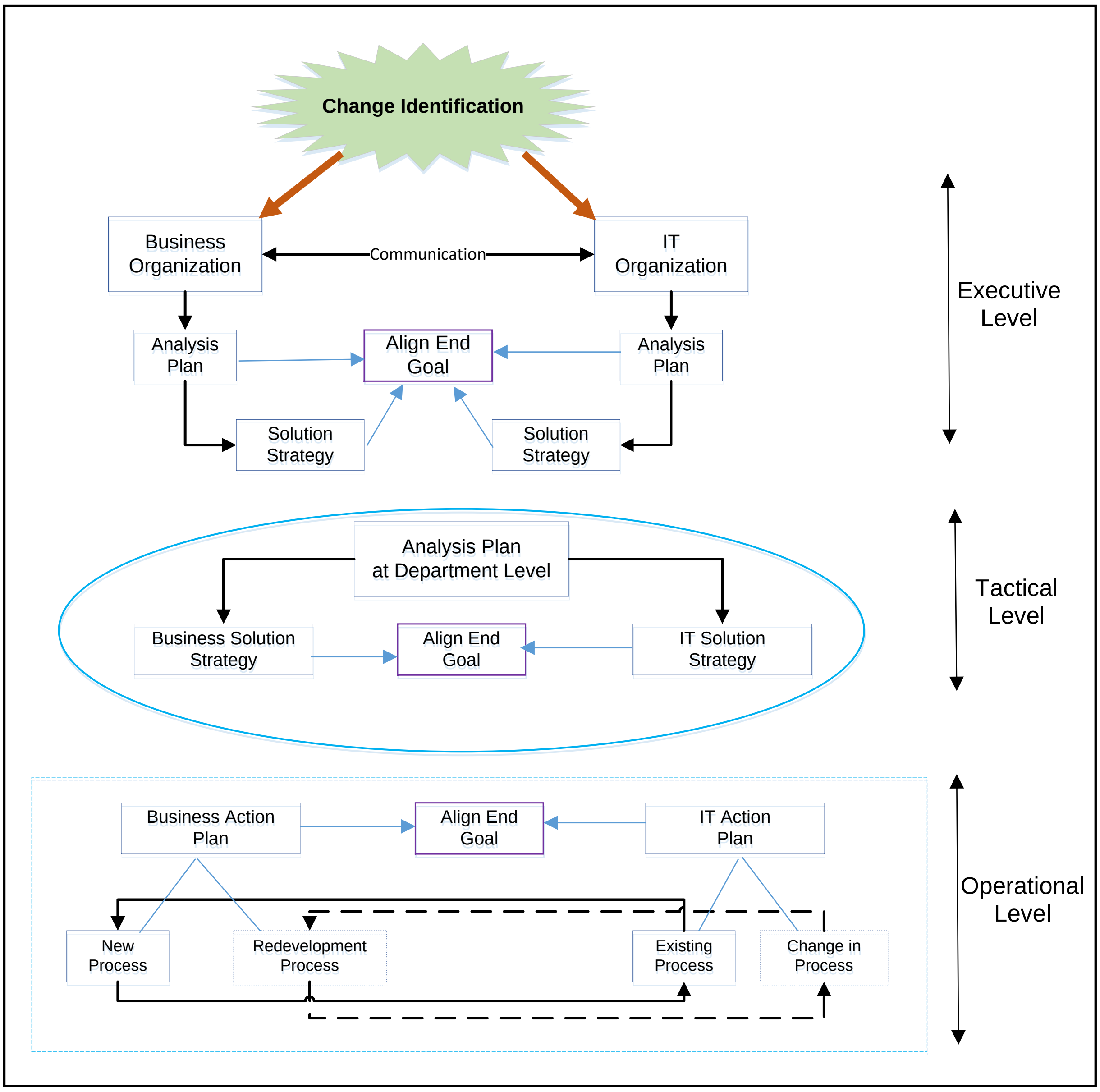


Figure 4: RCM with respect to organization level

(iv) Different viewpoints based on structure

Change analysis can be observed from two main viewpoints: one from a developer point of view at a code level and the second from a decision-maker's point of view at a higher abstraction level. The executive and the tactical levels can be considered as the decision-maker point of view while the operational level represents the developer point of view. There has been debate over which of these levels is more important in change management. Some of the literature emphasizes the importance of managing change at a program modification level where such analysis would be helpful to a programmer to effectively implement the change

[181-183]. In support of a higher level of decision making to effectively manage change, many studies argue that it is inaccurate to realize change at the code level, where in fact the source of the change is at a requirement level and therefore should be managed at a higher abstraction level [140, 144, 145].

(v)Decision making and organizational culture in agile development

The primary goal of all agile methods is to deliver software products quickly, and to adapt to changes in the process, product, environment, or other project contingencies [184]. While evidence suggests that agile methods have been adopted in a wide variety of organizational settings [185-187], such methods are assumed to be more suited to certain organizational environments than others. According to [185-188], agile development is more suited to smaller organizations as development is carried out in small teams. There are scalability issues when it comes to large organizations or large projects [186, 187]. In smaller organizations, there is a strong positive correlation in some aspects of organizational culture with that of agile development; the organization values feedback and learning; social interaction in the organization is trustful, collaborative, and competent; the project manager acts as a facilitator; the management style is that of leadership and collaboration; the organization values teamwork, is flexible and participative and encourages social interaction; the organization enables the empowerment of people; the organization is results-oriented; leadership in the organization is entrepreneurial, innovative, and risk taking; and the organization is based on loyalty and mutual trust and commitment [189].

There are certain characteristics of agile development, such as cross-functional teams and customer involvement that create harmonious interaction between various levels of the organization in decision making. Cross-functional teams include members from different functional groups who have similar goals [98, 190]. Such a practice combined with customer involvement helps reduce challenges such as over scoping of requirements and communication gaps, which are some of the key causes of requirement change. According to these studies, agile development has the ability to create harmony within the organizational culture and within the structure of the organization that will positively contribute to the reduction of the number of changes required and will be able to gain better clarity in decision making and the development of software projects.

Key findings of RQ$_4$

Not many studies in the literature used for this survey discuss how decision making at various levels of the organization may differ. We feel that this is an important concept to investigate as such differences in decisions can create difficulties in coming to a consensus on accepting the change and also moving forward by executing the change. Based on the discussion that formulated the answer for RQ$_4$, the key findings are as follows:

1) It is important to realize that based on the level of the organizational structure, decision-making concepts differ and this can be detrimental to the success of a project when dealing with RCs.
2) An organization can be divided into two parts i.e. the business organization and the IT organization.
3) Each of these two parts can then be divided into three levels of structure: Executive, Tactical and Operational. The differing levels of decision making between these structural levels have been identified to be a challenging factor in RCM.
4) Not only can decision making be contradictory at each level, it can also cause a contradictory understanding of the change between the business and IT counterparts.
5) There are also two viewpoints to consider: the developer and the decision maker. The literature seems to be divided on which viewpoint is more important, providing cause and effect for merit for both viewpoints.
6) Agile techniques tend to be a better way of development when it comes to creating better harmony within the organizational culture and decision making. However, this comes with the constraints of scalability and therefore is better recommended for development using smaller teams or for smaller organizations.

8. Comparison with related work

There is a plethora of work which has been evaluated in various areas of RCM, such as change impact analysis, change complexity analysis, change decision support, change identification, etc. A number of literature reviews related to change management have been conducted on research topics such as identifying change causes [61], change taxonomies [57] and requirement change process models [42]. These reviews deal with only one aspect of RCM, as detailed in Table 22.

Research work	Findings and contributions
Towards an understanding of the causes and effects of software requirements change: two case studies [57]	The study identifies various causes of requirement change and uses a simple taxonomy to group these causes for better understanding and future identification.
Causes of requirement change-a systematic literature review [61]	Similar to the previous study, identifies the causes of requirement change and groups these cause into two categories; essential and accidental. The main difference from [57] is that the study is done as a systematic review.
Requirement change management process models: Activities, artifacts and roles [42]	The study brings together various requirement management models, identifying their key features.

Table 22: Comparison with related work

In comparison, the work presented in our systematic review investigates the causes of requirement change and the processes/models used for RCM, it explores in-depth the techniques used in RCM and the decision making in managing change and provides a critical analysis of the methods extracted by identifying research gaps. The methods extracted comprise both traditional and agile techniques in RCM. In summary, this review provides information related to many aspects of RCM in more detail, giving a more holistic view for its readers.

9. Threats to validity

The findings presented in this review study have the following threats to validity.

(i) Construct validity: this is primarily related to obtaining the right information by defining the right scope. At this stage, the biggest challenge is to decide what should be included in the review. To address this issue, we considered all the studies which provided empirical, case study, experimental, industrial and survey-related information about RCM.

(ii) External validity: the findings of this review cannot be generalized because the results are based on a specific set of keywords and the research repositories that have been used for the data collection. Therefore, our results could be limited and cannot be applied to every organizational setup.

(iii) Results validity: the concept of RCM has a very long history dating back to the early 1980s. The area is still evolving and a large set of keywords are available which can be used to represent the concept of RCM. In this review, we considered 12 different keywords which are mostly used in the context of RCM in software development, and used six research repositories to conduct an initial search in the study selection process. Thus, our findings are only based on the selected set of keywords and from six research repositories.

(iv) Internal validity: this is mainly related to the capability of replicating similar findings. We addressed this aspect by defining and later following the systematic review procedure, described in section 3. Two researchers were involved in the review process, who, over a period of time, worked together to avoid duplications and achieved consensus in the acceptance of the identified studies. However, it could be possible that if this study is replicated by other researchers, minor variations in the identified studies will be observed due to differences in personal aptitude and thinking. Regardless of this fact, the findings presented in this review will enable readers to obtain a clear picture of RCM.

(v) Conclusion validity: The number of research articles presented in this study does not indicate the actual number of RCM practices being undertaken in reality. Thus, the number could only be used to make inferences as to how practical and applicable RCM methods are.

10. Conclusions and Future Work

It is evident that changes in requirements occur for many reasons and can be caused by multiple stakeholders. Regardless of who or what cause these changes, the need for appropriate management is great due to the undesirable consequences if left unattended. However, through this review, it was discovered that change management is an elusive target to achieve and that there are many ways to tackle it. The main objective of this review was to collate information and techniques related to RCM and critically analyse the functionality of such techniques in managing change. This also led to identifying strengths and limitations of these techniques, which signifies the need to enhance the existing change management approaches. This review is also a guide for future researchers on change management in terms of what major work has been undertaken thus far.

In the review, the section on factors that cause change in requirements provides an understanding on how vast and constant these changes can be. There is no one root cause for changes which makes change management a challenging task. Therefore, even with an abundance of research on change management, there is still room for improvement. Given the complexity of changes, it is important to identify the processes in place to manage them. It is clear from the available literature that there is no consensus on how to manage change. In some instances, it is based on the type of organization and the environment and in many cases, it is based on the type of changes. Through the available process steps, three common processes were identified; identification, analysis and cost estimation of change. Significant work has been done in each of these areas and several models that encompass these steps have been developed in an effort to provide a full-scale solution for change management. It is also important to understand that the approaches vary depending on the level of the organisation managing the change.

When identifying future work in RCM, we deemed it useful to focus on the three areas of RQ_3 where the majority of the techniques have been discussed. We do not directly suggest future work but identify the research gaps in the areas of change identification, analysis and cost estimation where the possibility for new research lies.

10.1 Research gaps in change identification

Accurate change identification not only leads to a better understanding of the required change but also the impact it can cause on the entire system and project. The techniques discussed in change identification can be divided into two categories: change taxonomies and change classification as discussed in the previous section. Given the existence of these methods, their still remains several major gaps that need to be addressed:

1) The parties involved in the elicitation and identification process of changes are from a variety of backgrounds and experience levels. Common knowledge for one group may be completely foreign for another. This is especially true in the case of communication between the analyst and the stakeholder(s).
2) The language and terminology used to communicate the changes to and from the stakeholder to the analyst and then to software practitioners (designers, developers, testers, etc.) may be either too formal or informal to meet the needs of each party involved.

3) There will be a large amount of information gathered that is part of one single change. Not having a common structure to categorize this information may lead to misinterpretation of the need for the change and the change itself.
4) Information gathered at one level of the organization could be biased based on the parties involved if one form of structure is not used to capture the changes at all levels.
5) The methods already in existence provide minimal guidance in terms of applying them to identify changes.

10.2 Research gaps in change analysis

As seen in the previous section on change analysis, it is clear that traceability is one of the most popular techniques to analyse the impact of changes on a system, either in existence or in the design phase. Several other non-conventional methods were also identified that contribute to change analysis. Through these methods and the existing knowledge on the volatility of requirements, several gaps in the research are identified:

1) Although traceability is a common method of identifying impact, it can be costly and time consuming, and in most cases, the benefits (of traceability) are realized immediately. This gives rise to a need for another method that addresses these limitations.
2) In most existing methods of change impact analysis, the priority of changes is not established. Understanding priority benefits the decision-making process by allowing software practitioners to establish which change to implement first and also how critical the change is to the existing system and hence, resources can be allocated accordingly.
3) The existing literature is unclear on ways to identify the difficulty of implementing a change in an early phase of the change request process. Understanding the difficulty associated with a change leads to better decision making in two ways: firstly, if the difficulty of implementing the change is too high and the delivery of the product is time sensitive, the change could be held back for a consecutive version; secondly, the difficulty can be used as a gauge of the effort required to implement the change.

10.3 Research gaps in change cost estimation

The cost estimation methods discussed in the previous section were not explicit for the estimation of implementing changes. In practice, these methods can still be applied for this purpose yet there is still much room for improvement. Based on the information discussed earlier and in the other related literature, several gaps in the research were identified:

1) No significant work in the existing literature caters explicitly for estimating the cost of implementing RCs. As demonstrated in the previous sections, changes occur for a plethora of reasons and can occur during any phase of the software development life cycle. Therefore, it would be beneficial if there was a dedicated method by which to estimate the cost of such changes as the implication of these changes based on the project's timeline results in different outcomes.
2) Estimation done at an early stage of the development process is usually based on expert judgement with less precise input and less detailed design specification. In some cases, this may result in effort estimation which is too low which leads to issues such as delayed delivery, budget overrun and poor quality while high estimates may lead to loss of business opportunities and the inefficient use of resources.
3) Estimating the cost in the early stages of development depends on expert judgment and historical data, which can be biased and inconsistent. There needs to be ways to eliminate these ambiguities in change cost estimation.

The research gaps identified indicate the importance of having a full- scale model that increases the efficiency of managing change with better accuracy. The review highlights that although the concept of change management has been in existence for many years, the applicability of the available methods has many limitations and has room for improvement. With challenges such as poor communication, impact identification issues and no dedicated method for change cost calculation, the avenues for future research is promising.

Chapter 3
Managing Software Requirements Changes through Change Specification and Classification

3.1 Preface

Based on the systematic review, we discovered that communication ambiguities during elicitation and negotiation periods of requirements changes lead to quite a number of negative outcomes in terms of managing and implementing these changes. As a result, some projects experience budget/time overruns and/or poor quality end products. We also discovered that change identification is a main activity in the process of change management. In order to clearly identify a change, it is imperative that there is clear communication between the business and IT staff involved in the change elicitation process. The main purpose of this chapter is to introduce a way to communicate and understand requirements changes without any ambiguities. To this end, we propose two methods as part of the change management process where the change specification method deals with the communication part and the change classification method deals with the identification of the change.

This chapter consists of a paper that investigates the research space on requirements change communication and taxonomies in order to produce the aforementioned methods. A preliminary version of this paper was presented at the 2013 Australian Software Engineering Conference. The findings of this chapter not only satisfy the requirements of a clear communication medium but this is also the first phase of the requirements change management process and these findings are used in the successive methods of this thesis.

3.2 Publications

S. Jayatilleke, R. Lai, and K. Reed, "Managing Software Requirements Changes through Change Specification and Classification," *To appear in Computer Science and Information Systems*, 2018. DOI: 10.2298/CSIS161130041J.

Manuscript title	**Publication status**	**Nature and extent of candidate's contribution**	**Nature and extent of co-authors' contribution**
S. Jayatilleke, R. Lai, and K. Reed "Managing Software Requirements Changes through Change Specification and Classification". Computer Science and Information Systems	Published online in accordance with Computer Science and Information Systems author guidelines.	**Eighty percent** contribution by candidate. This included gathering information, drafting and revisions of the manuscript.	**Twenty percent** contribution by co-authors. This included discussions of ideas expressed in the paper, critical review and submission to the journal.

A preliminary version of this paper was presented at the 2013 Australian Software Engineering Conference (S. Jayatilleke and R. Lai, "A method of specifying and classifying requirements change," in *Software Engineering Conference (ASWEC)*, 2013 22nd Australian, 2013, pp. 175-180: IEEE.)

Signed : [illegible] Date : 12/02/18

(S. Jayatilleke)

Signed : [illegible] Date : 12/02/18

(R. Lai) (on behalf of co-authors)

Managing Software Requirements Changes through Change Specification and Classification[1]

Abstract — Software requirements changes are often inevitable due to the changing nature of running a business and operating the Information Technology (IT) system which supports the business. As such, managing software requirements changes is an important part of software development. Past research has shown that failing to manage software requirements changes effectively is a main contributor to project failure. One of the difficulties in managing requirements changes is the lack of effective methods for communicating changes from the business to the IT professionals. In this paper, we present an approach to managing requirements change by improving the change communication and elicitation through a method of change specification and a method of classification. Change specification provides a way such that communication ambiguities can be avoided between business and IT staff. The change classification mechanism identifies the type of the changes to be made and preliminary identification of the actions to be taken. We illustrate the usefulness of the methods by applying them to a case study of course management system.

Keywords—Requirements change, change specification, change classification, ontology, terminology.

1. Introduction

The inevitable development of globalization, service-oriented environments and continuous technological advances compel organizations to change their strategies and business processes to meet customer demand. In addition, there is the impact of software evolution and maintenance. Although change is an evident factor in today's highly competitive business environment, many organizations find themselves at the losing end of this game. Volatile nature of business requirements usually increases the cost of development [1, 16, 32, 55, 191, 192] and also poses a threat to the project schedule [16]. Changing requirements are considered one of the main contributors to project failure [45, 46, 193]. The real problem is not the

[1] A preliminary version of this paper was presented at the 2013 Australian Software Engineering Conference.

changing nature of requirements, but the lack of understanding of this volatility. Change management, therefore, is a critical task for organizations.

Requirements engineering consists of a set of core activities that are in reality interleaved and iterative [38]. Requirements change is part of this requirements engineering process and it is not a standalone activity but consists of several core activities that can be described as a process. This process begins with communicating the requirements change (change request). Successfully completing this step will result in the elicitation of the correct goals in relation to the changes (change goals), which is the next step in the process. . Understanding the change goals leads to the proper execution of the third step, which is representing the change in the system design. The second and third steps effectively assist the analysis of the requirements change to assess its appropriateness and whether it should be accepted. The final step in the process is based on the results of the analysis. Depending on the outcome, a change can be accepted or rejected. Therefore, the final outcome of the change request depends heavily on the first step. This process is iterative, usually due to the inability of management to agree to the change request and due to insufficient information. It is further hindered due to poor change communication, misinterpretation of change goals, incorrect representation of changes in the system design, discrepancies in analysing the changes, and inaccurate decision making in relation to the requested changes.

One of the key reasons for difficulty in managing change occurs at its initiation. Effective interpretation and communication change, from the customer to the development level has proved to be a challenging task [10-13]. Some literature suggests that this is due to the lack of a formal process specifying change [10, 13]. The specification method used by change originators should be understood by both business and IT personnel since it is the bridge between the change originators (users, customers, etc.) and the change implementers (system analysts, designers, developers, etc.) [194-196]. Therefore, being able to specify and understand the requirements change should make in the process of incorporating the change into the existing design or system more seamless.

In this paper, we present an approach to managing requirements change by improving the change communication and elicitation through a method of change specification and a method of classification. Change specification provides a way such that communication ambiguities can be avoided between business and IT staff. This is the first step towards better and effective management of requirements change in rapidly changing business environments. The change specification process is incomplete without classifying the changes. The change classification mechanism identifies the type of the changes to be made and preliminary identification of the actions to be taken. To aim readers to have a better understanding of the change specification and classification methods, we use a simple mail order system as a running example. Finally, we illustrate their usefulness by applying them to a case study of course management system.

A preliminary version of this paper was presented at the 2013 Australian Software Engineering Conference [76]. The following items are contained in this paper but not in [76]:

(i) a discussion on the related work to give better understanding of our methods;
(ii) a description of the overview of the methods;
(iii) a justification of the use of Goal Question Metrics (GQM) and Resource Development Framework (RDF) approach; and
(iv) to illustrate the usefulness of our methods, the results of applying them to a running example and a case study.

2. Overview of the methods

In this section, we present an overview of our approach to managing requirements change through a method of change specification and a method of classification. Managing change begins with an understanding of what is involved in this phenomenon. But as previous studies have proven, there is no real consensus on the nature of change, rather there are disparate multifaceted views and approaches. We therefore see the need for a versatile, consolidated, solution that brings these together. Based on previous research work and also through industrial interviews described later, we were able to pinpoint the gap in change identification. There is an inadequacy in applying change identification in the practical context. Figure 1 using the IDEF0 notation shows the broad layout of the methods aiming to overcome this limitation. Once a change is requested, the layout follows two steps;

1) Change specification
2) Change classification

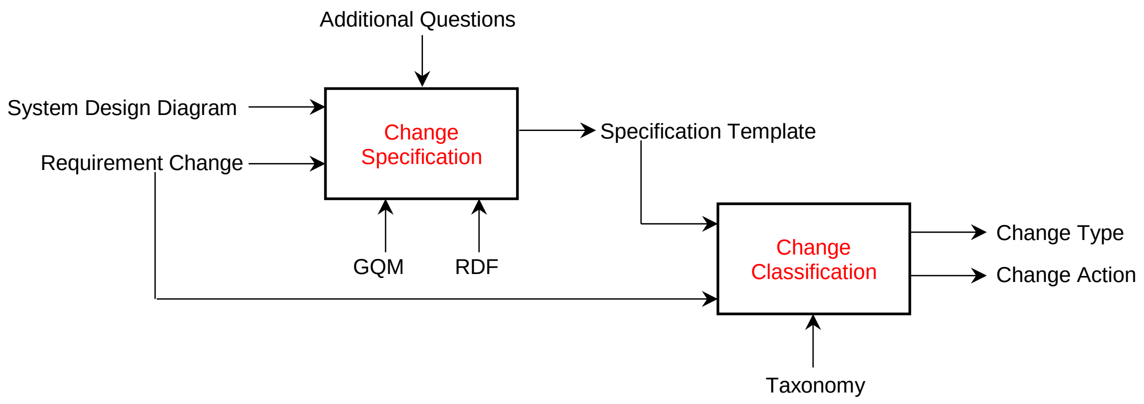


Figure 1: Layout of overview of the methods

Change specification denotes a way of specifying a change so that communication ambiguities can be avoided between business and IT staff. Once a requirement change has been initiated from the client side, this method will use the system design diagram as an input to map out the location of the change. In order to create the specification template we will use two established methods, i.e. Goal Question Metrics (GQM) [197] and Resource Description Framework (RDF) [198]. We will also use a set of additional questions to enable better identification when using the speciation template output. The purpose of using GQM and RDF is to establish terminology and ontology (respectively) concepts in the specification method. The use of terminology will enable the specification template to have standardized terms whilst ontology will ensure a logical connection between the terms used in the specification template. The purpose of using both terminology and ontology is further discussed in section 3.2.1. The outcome of the specification template will be the identification of the location, purpose and focus of the change.

GQM approach, which was developed by Basili and Weiss and expanded by Rombach [197], is the most widely known goal-focused approach for measurement in software. One of the reasons for its success is that it is adaptable to many different organizations (e.g. Philips, Siemens, NASA) [197]. Another reason for the success of GQM is that it aligns with organizational directions and goals. Rather than using a bottom-up method (generally problematic) [199], metrics are defined top-down. This way the measurements are linked to organizational goals [199-201]. This same concept can be applied in describing change. If the changes described are linked to goals, then understanding and application of such changes could be far more efficient [202].

Introduced by Tim Berners-Lee in 1998, RDF is an ontology language for making statements about resources [198]. It was designed for describing Web resources such as Web pages. However, RDF does not require that resources be retrievable on the Web. RDF resources may be physical objects, abstract concepts, in fact anything that has an identity. Thus, RDF defines a language for describing just about anything. Furthermore software modeling languages and methodologies can benefit from the integration with ontology languages such as RDF in various ways, e.g. by reducing language ambiguity, enabling validation and automated consistency checking [203]. Given the benefits of both GQM and RDF, it was deemed appropriate to use these methods for specifying requirements changes. With these being the general benefits of GQM and RDF, their specific purpose and use in the specification method are described in detail below.

The change classification method uses the outcome of the specification template to expand on the type of change along with preliminary guidance for action to be taken in managing the change. The classification itself is based on the concepts of change taxonomy that was found in existing change management literature and refined using unstructured interviews of practitioners in the field of change management. The outcome of the change classification will provide software developers with a better understanding of what the change is and the preliminary guidance on how to incorporate the change into the existing system. We believe the combination of change specification and classification leads to a better realisation of changes requested.

2.1 A running example

To aim readers to have a better understanding of the change specification and classification methods, we will use a simple mail order system for CDs and DVDs as a running example which is described below.

Diskwiz is a company which sells CDs and DVDs by mail order. Customer orders are received by the sales team, which checks that customer details are completed properly on the order form (for example, delivery address and method of payment). If they are not, a member of the sales team contacts the customer to get the correct details. Once the correct details are confirmed, the sales team passes a copy of the order through to the warehouse team to pick and pack, and a copy to the finance team to raise an invoice. Finance raises an invoice and sends it to the customer within 48 hours of the order being received. When a member of the warehouse team receives the order, they check the real-time inventory system to make sure the discs ordered are in stock. If they are, they are collected from the shelves, packed and sent to the customer within 48 hours of the order being received, so that the customer receives the goods at the same time as the invoice. If the goods are not in stock, the order is held in a pending file in the warehouse until the stock is replenished, whereupon the order is filled. This process can be illustrated by the following system design diagram.

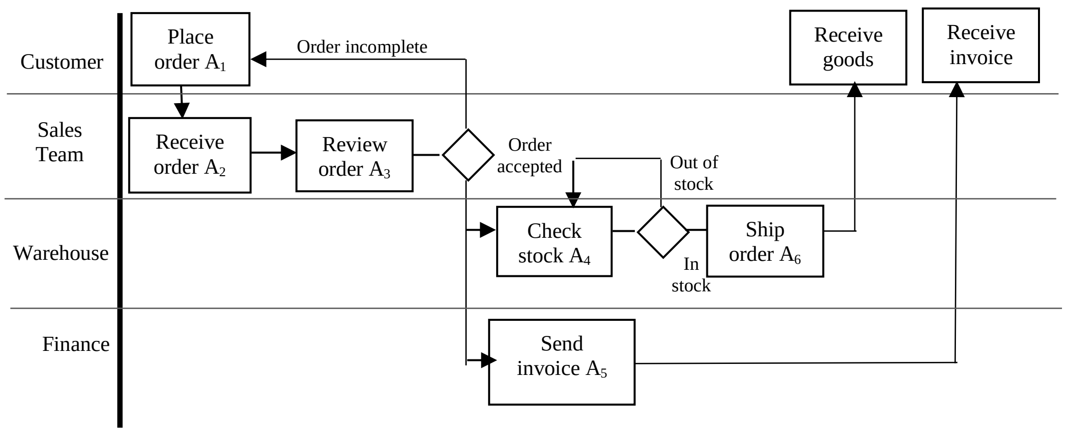


Figure 2: Diskwiz customer order fulfillment process diagram

The example consists of a scenario where the specification method is applied in specifying the change and the change classification method is used to identify change type and corresponding action. The scenario is as follows:

The management is not satisfied with some parts of the process and point out that the following issue should be rectified: “It is identified, due to a design error, there is no communication between finance and the warehouse to confirm discs are in stock so that the order can be shipped. Therefore finance could be raising invoices when the order has not been sent.”

3. The change specification method

Figure 3, represents collaboration of the different entities of the change specification method. The change specification consists of three key elements: a system design diagram, a specification template and additional questions. The foundation of specification component is made up of GQM and RDF. The GQM-RDF combination is a result of amalgamating ontology and terminology which in this paper, we refer to as onto-terminology. A detailed description of the onto-terminological concept and the interaction of the three elements in specifying changes are explained in the following sections. We point out that in fact, or method is "system description technique agnostic", and, could be used in any environment where a systematic system description methodology has been used, reducing the adoption casts.

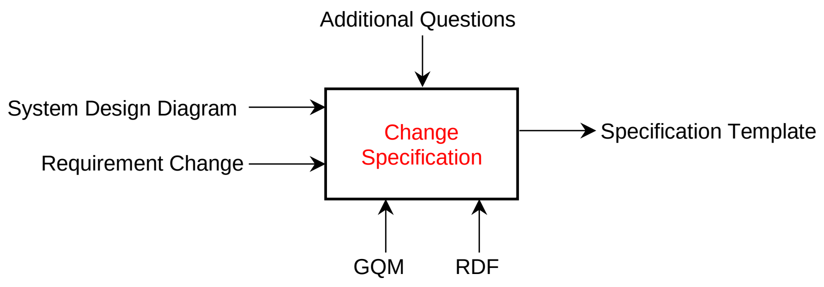


Figure 3: Layout of the change Specification

According to Figure 3, an important input is the use of system design diagrams. In this cases where the initiation of the change takes place on the business side. Therefore, the initial part of the change specification should be familiar to the business personnel involved. To achieve this, system design diagrams are used as part of the change specifying process where the notations and the language used are more business related. Any business analyst communicating a requirement change to the IT side should be capable of understanding and interpreting a system design diagram.

The successful application of the change specification calls for a few key assumptions. First, the specification of changes may take place at the operational level of the organization. We believe that as changes flow from an executive level (top) to the operational level (bottom), they become less abstract, making it easier to feed the change into the specification and classification methods. Second, in reality, for a system to be stable, the changes being made are proportionately small (5% – 10%) in comparison to the complete system [65]. On the other hand, if the change requires more than a 50% change to the system, it is usually implemented in a successive release of the current system. Finally, a design diagram (preferably the system design diagram) should be available for mapping the change to the system.

3.1. Specification prerequisites

Although there is a plethora of ways to describe change, most fall into ad-hoc methods of communication. In the authors' view, a void exists which could be filled by a more effective and efficient template and a set of guidelines that can be used to communicate requirements change. Given the current trend of business being more service-oriented, the change specification should be a bridge between customer requirements and the final product [204]. The new specification template introduced in this paper will reflect this. The following two key properties are essential for a specification method to be both functional and constructive [204].

A primary objective for the specification method is user friendliness to ensure ease of adoption. It is important to recognize that the process of specifying either requirements or changes to requirements is a human activity process [204-206]. Therefore, the method used for such specifications should be human friendly [204]. The initial response to a new method is generally resistance and an unwillingness to use it [204, 207]. This is usually because the difficulty level of the new method is unknown to the users. Also, both businesses and IT stakeholders involved in the change management process tend to trust tried and tested methods of specifying change simply because there are no "surprises" in store. For these reasons, rather than inventing an entirely new method, we have opted to use a combination of existing methods which we believe has the most desirable qualities of a specification method and with which the users are familiar. This, in our view, will minimize the short-term productivity losses associated with learning new process, and also reduce the likelihood of opposition.

The second property is the method style. Text-based specification methods are formed using either natural language or formal language [204]. Although easier to understand, the drawback in using natural language is that it may be interpreted in different ways, resulting in ambiguities. Whereas a mathematically influenced formal language may be ideal for a computer, it may not be human friendly. Therefore, it is important to find a balance in textual illustration. Also equally important is that both business and IT stakeholders involved in the process understand the specification method. To achieve this, we introduce a semi-formal method which is aided by system design diagrams.

3.2. Onto-terminology framework

3.2.1. The purpose of ontology and terminology

The specification method introduced in this work is a means of semi-formal communication of requirements change. And for this method to be both informative and useful, it needs to satisfy several conditions. A specification method should take into consideration: standardised terms, the usage of the terms, connotative information and linguistic relationships as well as a logical and philosophical point of view of the standardised terms [208]. We point out that these features stem from two different concepts i.e. terminology and ontology. The relationship between terminologies and ontologies has been the subject of analysis by others, as we see from the following discussions.

Terminology is a “set of designations belonging to one special language” [209]. The main purpose of using terminology in a specification method is to eliminate ambiguity and ensure the use of standard terms [208]. International standards state that the goal of terminology is to clarify and standardize concepts for communication between humans [209]. This is a crucial property of our proposals as this is a method of conveying changes in requirements from business personnel to IT personnel. However, terminology generally lacks computational representation as well as logic [210]. Of these, our concern with regard to change specification is logic. Logical accuracy will ensure that the action taken to implement the change is correct. Therefore terminology, on its own, cannot be considered for the semi-formal framework of the change specification method.

Ontologies are similar to terminologies in that both the communication of concepts. According to Gruber [211], ontology describes a concept and its relationships in a way that can be manipulated logically. The way ontology defines a concept depends entirely on the formal language used for the communication of the concept. Ontology is not a terminology [208]. In fact, ontology lacks the standardized terms and linguistic relationships of a concept which are key features in terminology [208]. These features are imperative to change specification as they build the actual form of communication terms to be used in the specification.

The conceptualization of the change specification method needs to be guided by both linguistic and logical principles. Given the strengths and weaknesses of terminology and ontology, the combination of these two concepts will provide a better framework for the specification. Onto-terminology, which results from this combination, formally defines the concept (ontology logic) as well as explains the term and its usage from a linguistic point of view (terminology).

3.2.2. Building the relationship between GQM and RDF

To ensure the correct combination of logic and terminology, we have selected two well-known methods where one represents terminology and the other represents ontology. A generalization of GQM is used as the linguistic function of the specification method representing terminology. It is important to note that the abstraction of GQM relates to the goal specification and not to the questions or the metrics. The purpose of using GQM is that it enables the extraction of specific terms that define the requirements change. Since these terms have been successfully utilized to extract business goals [199, 200], we found it's use satisfactory in change specification. The logical connections for the terms are sourced from RDF representing the ontology component specification. However, it can also be used to link information stored in any information source that can be ontologically defined [210].

Three terms are extracted from the goal specification of GQM that can best describe a requirement change; Object, Purpose and Focus (of change). The meanings of these three elements have been adjusted for the purpose of describing change. The Object needs to be changed due to the Purpose using the Focus. The terms extracted from RDF are Object, Attribute and Value, which is referred to as the RDF triplet [210]. The logical relationship of the RDF triplet can be stated as Object O has an Attribute A with a Value V (Professor; Reads; a Book). The rationale behind the correspondence between RDF triplet and to the GQM terms is due to the similarity and the meanings of the terms, which is described in Table 1.

RDF term	GQM term	Correspondence	Rationale
Object	*Object*	One-to-one	Same concept
Attribute	*Purpose*	One-to-one	Both terms are activities. *Purpose* is an activity that is generated due to various business requirements.
Value	*Focus*	One-to-one	*Value* of RDF creates the significance for *Attribute* (of RDF). *Focus* of GQM creates the significance for *Object* (of GQM) by activating the term *Purpose* of GQM.

Table 1: Rationale of RDF and GQM relationship

GQM terms alone could have been used if the three terms have a logical connection; and we have explained above as to why it is important to have this logical connection in a specification language. The main reason for using RDF is hence to create the logical relationship between GQM terms. Figure 4 represents the relationship mapping between RDF and GQM. As such, the logical relationships between GQM terms can be stated as Object O needs Purpose P by using Focus F. Given the logical connection established, any change specified (regardless of the application of the system) using the GQM terms will satisfy the requirements of a semi-formal method of communication as stipulated above (see 4.2.1). From now, we shall use these three terms in the specification method.

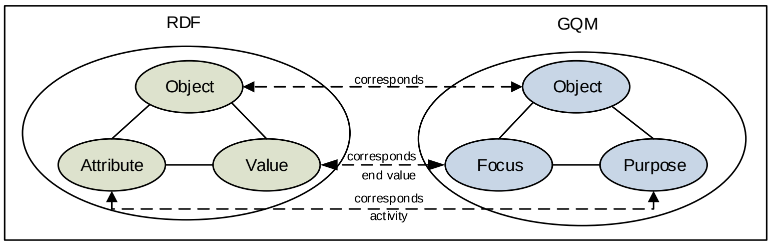


Figure 4: RDF-GQM Relationship

The framework presented in Figure 5 is based on the above relationship and is the foundation of the specification method. The three elements OBJECT, PURPOSE and FOCUS are used to capture the requirement change. The OBJECT of change is any activity in the system design which needs a PURPOSE to change. This purpose is created as a result of changing business goals, customer requirements, etc. The object is changed by the FOCUS of change, where any change type can denote the focus. Therefore, each activity in the system design is an object, each changing business goal and customer requirement is a purpose and each change type is a focus.

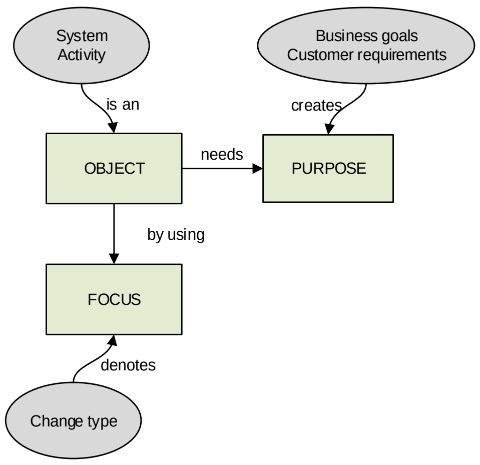


Figure 5: Onto-terminology Framework

3.3. Text-based specification tool

During the preliminary studies we examined several different types of change request forms from industry to understand what information is vital for understanding a requirement change and how it was presented. We discovered two common denominators that should be included in our specification tool. First, the type of change which assists the system designers to understand the action they need to take in order to accomplish the change. Second, the reason for change which gives a better insight as to why the change was requested.

The template designed for the change specification based on the framework in Figure 4 is given in Table 2. By selecting the object of change using the system design diagram, designers and decision makers can accurately locate the main target of change, resulting in a clarification of the location of change. Knowing the reason for the change through the purpose ensures that change implementers are able to clarify the need for the change. The focus of change acts as advice on the basic implementation needed to execute the change, resulting in the clarification of the action of change. It indicates to the designers what to do instead of how to do the change. We believe that clearly describing the location, need and action of a change request using this template will resolve much of the existing miscommunication issues.

	Description
OBJECT	The activity name according to the system design diagram
PURPOSE	The reason for the change (can be descriptive)
FOCUS	Select from Add, Delete, Modify or Activity Relocation (description given in table 6)

Table 2: Template for change specification

An additional question (see Table 3) is used along with the above template based on the focus of change that investigates additional inputs and/or outputs required for the change. Answer to this question will be used as input for the change classification method, which is discussed below.

Focus of change	**Additional question**
Add	Need addition Input/output?
Delete	Connected to neighbor activity with input/output?
Modify	Input/output modification?
	If Yes; Input modification? Output modification?
Activity Relocation	Relocation requires input /output?

Table 3: List of addition questions

3.4 Results of applying it to the running example

By applying the change specification method to the running example, we obtain the following results.

	Description
OBJECT	A_4 and A_5
PURPOSE	Resolution of design error
FOCUS	Add
Additional Question	Need addition Input/output? Y

Table 4: Application of the Change specification method

We have used the templates given in Tables 2 and 3 in order to populate the information in Table 4. It is mentioned in the change scenario that this change is required due to a design error. Therefore, the purpose of this change is listed as a resolution for a design error. The activities that are affected by the change are identified through the design diagram to be Check Stock (A_4) and Send Invoice (A_5). This is again based on the change scenario. The analyst then needs to decide with which focus this change will be executed. In this particular case, it is determined that a new activity needs to be added to handle the change. The next step is to identify if the addition of the new activity would cause new input/output between the existing activities (A_4 and A_5) and the new activity. As we are trying to bridge the communication between A_4 and A_5, based on Table 3 it is most likely that such input/output would be generated and therefore the answer to the additional question is ‘Yes’.

4. The change classification method

The main purpose of change classification method is to ensure that change implementers are able to identify and understand unambiguously the requirement change [10, 31]. Therefore it is essential that the classification itself is not complex. The change specification method is incomplete without having to classify the change as it provides a further understanding of the underlying causes of requirements change [31, 212]. This is the first step towards better and effective management of requirements change in this rapidly changing environment. Other studies [10, 213] also suggests that a classification of change is a scientific step to improve our ability in understanding requirements evolution.

4.1. Preliminary studies

To explore the scope and complexity of the existing change classifications and determine the criteria for our change classification, two key investigative methods were undertaken. Firstly, a literature review of existing research on change management with a focus on change classification was undertaken. Keyword searches included change management, change classification, change types, change taxonomy, and change specification. The total result of 43 included journal papers and text books. This was filtered using selection criteria which were limited to articles referring to classification, type and taxonomy which yielded in 12 academic works [10, 12, 16, 18, 23, 31, 32, 75, 114, 212-214]. These papers allowed us to extract the most common and regular change types used in the industry.

Secondly, unstructured interviews of 15 practitioners in the field of change management were conducted. Table 5 summarizes the important questions discussed and how they are related to this study. Respondents included project managers, business analysts, IT analysts, and software architects. Since these practitioners were from several software development organizations, the methods followed in change management was quite diverse. One of the key findings was the difficulty in relaying the business requirement change down the IT development line. A secondary related problem which arouse was the misinterpretation of the requirement change and business goal. There were many cases where parts of the final product did not meet the customer satisfaction as the changes requested had not been implemented appropriately. This justified our efforts in creating a change classification that facilitated better understanding of the requested change. We used these interviews to further confirm the change types identified through the literature survey and were able to gain better insight to improve the change classification.

Question	Purpose
How often are changes requested and where do they originate from?	To understand the frequency of change request and where they are usually generated from
What are the types of changes that are often requested?	To identify the different types of changes
Is there a process for requesting change? If so, what are the details?	To identify the steps involved in a change request and what vital information needs to be captured
What are the difficulties in communicating change?	To understand the existing problems in the industry and what is lacking in their process of change communication
Is unambiguous communication of change important? If so, why?	To identify if there is a need for a new method of specification and classification of change

Table 5: Key question of the interview

4.2. Taxonomy development

Our classification is based on previous work-see [16, 32, 75, 214]. Table 6, demonstrates how each previous work has influenced the creation of taxonomy. However, further adjustment was made to improve the classification as mentioned above. The focus of change represents the most common forms of changes found in requirement change requests. Table 7 lists the detailed description of these basic changes. Changes Add, Modify and Delete were identified initially as the classification as a result of both previous literature and practitioner interviews. Change, Activity Relocation was included as a result of information gathered through the interviews as we discovered, is a frequent form of change requested. In normal circumstances, combinations of these basic change types can be used to represent more complicated change scenarios. These same change focuses were used in the specification method in-order to create a clear connection between the two methods.

Previous work	**Concepts extracted**	**Application to the classification**
Nurmuliani, Zowghi & Williams [32]	Common types of changes used (add, delete, modify) and classification of changes	Helped in creation of the most common focus types
McGee & Greer [16]	Change causes and use of experts in defining a taxonomy	Leading to different change activities and the use of change practitioners
Nurmuliani, Zowghi & Williams [75]	Categories of change	Helped in creation of the most common focus types
Xiao, Quo & Zou [214]	Primitive changes in business functions	Further expression of change types

Table 6: Key literature used in creation of classification

Change focus	**Answer to Additional Question**	**Change type**	**Action**
Add	No	Matched links	Add new activity without changing the current activity or any connected links
	Yes	Mismatched links	Add new activity by changing the activity and/or connected links
Modification	No	Inner property modification	Modify the implementation of an activity without changing the connected links
	Yes	Input data modification	Modify the input link and internal properties of an activity
	Yes	Output data modification	Modify the output link and internal properties of an activity
Delete	No	Matched links	Delete activity without changing connected activities
	Yes	Mismatched links	Delete activity by changing connected activities and links
Activity Relocation	No	Relocation with matched links	Relocate existing activity without changing the activity or connected links
	Yes	Relocation with mismatched links	Relocate new activity by changing the activity and/or connected links

Table 7: Detailed change description

Application of Table 7 in the classification method can be described as follows. The change focus and the answer to the additional question of the specification method will be used in the classification method as follows. For example, if ‘Add’ was selected as the change focus and the answer was ‘Yes’ to the question ‘Need additional input and/or output?’, then according to Table 4 the linking interface(s) of the new activity and the neighboring activities will mismatch. Therefore the change will be categorised under ‘Add’ change focus with ‘Mismatched links’. The 4th column in Table 6 represents the necessary action to be executed for each change type.

‘Modification’ change focus is divided into three types of change. Inner property modification will deal with modifications done to the variables and operation of an activity that does not affect its external links (input/output) to neighboring activities. Input and output data modification will respectfully affect neighboring activities linked to the input/output of the target activity as well as the internal properties of the target activity depending on the input and/or output added to it.

In ‘Delete’ change focus with ‘Matched links’, no modification is needed once the target activity has been removed. The rationale behind this action is that the deleted activity does not provide any output or take in any input from its neighbors. In contrast, with ‘Mismatched links’, once the target activity is deleted, the neighboring activities have to be modified depending on the input/output connection(s) to the deleted activity.

Activity relocation will involve moving an activity from its current location and linking it into a new location in the system design. This can be achieved in two ways. One, the activity being relocated is not linked to its neighbors through input/output and able to relocate to the new position without any modifications to the neighboring activity. Two, the target activity in the current location and the new location are affected through input/output and needs to be modified.

At implementation time, the key elements of the two methods (specification and classification) are incorporated into a single table (see Table 8). In the table, change number refers to the number given to each change as they are requested. The object, purpose and focus in Table 8 correspond to the information given in Table 2 i.e. activity name according to the system design diagram (this is the activity affected by the change), reason for change and select from Add, Delete, Modify or Activity relocation respectively. The additional question selected from Table 3 will be based on what has been selected for the focus and the information provided through the content of Table 2. Change type and action can be sourced from Table 7 based on the information provided for object, focus and additional question respectively. The possibility columns represent how each change may be described using different focuses. This may not apply to all changes. The ability to create multiple possibilities which will be based on the experience of the analyst and complexity of the change. This feature was added to the implementation template to provide more diversity and flexibility of communicating a change. Having multiple possibilities also provides flexibility of how the change can be implemented.

	Change No.	Possibility 01	Possibility 02	Possibility n
Specification Method	**OBJECT**			
	PURPOSE			
	FOCUS			
	Additional Question			
	RESULT			
Classification Method	**CHANGE TYPE**			
	ACTION			

Table 8: Template for implementation

4.4 Results of applying it to the running example

By applying the template for implementation for the above scenario, we obtain the following result as given in Table 9:

Change 01	**Possibility 01**	**Possibility 02**
OBJECT	A_4 and A_5	A_4 and A_5
PURPOSE	Resolution of design error	Resolution of design error
FOCUS	Add	Modify
Additional Question	Need addition Input/output? Y	Input/output modification? Y
Result		
Change Type	Add new activity between A_4 and A_5 (Mismatched links)	Inner property modification and Output data modification A_4 and input data modification of A_5
Action	Add new activity by changing the activity and/or connected links of A_4 & A_5	Modify A_4 to send message to A_5

Table 9: Application of the implementation template

In Table 9, we describe the two possibilities for the scenario provided in the running example. For both possibilities, the object and the purpose remains the same and coincide with what has been discussed in Table 4. We are of the opinion that there are two ways this change can be described and the focus of each possibility demonstrates this fact. Possibility 1 was introduced in Table 4. The sections above the Results row of Table 9 is based on applying Tables 2 and 3 of change specification and were discussed in section 3.4. Based on the information provided for the Focus and Additional question, change type and action can be extracted from Table 7.

This extraction is shown in Table 9, for each possibility based on the different change Focus which has been identified. In the case of Possibility 1, the Focus identified is ‘Add’ and the Additional question has been given an answer ‘yes’. When this information is mapped to Table 7, it provides a Change type of ‘Mismatched links’, which requires a change Action of ‘Add new activity by changing the activity and/or connected links’. When adding the new activity between A_4 and A_5, connections need to be made with both activities. Therefore, both A4 and A_5 will be directly affected by this addition. The modification possibility of A_4 will directly affect A_5 as there will be link input from A_4 to A_5. In both possibilities, all activities that are connected to A_4 and A_5 will be indirectly affected by the alterations.

5. An application of the methods

Yin [215, 216] explained the usefulness of using case studies to explore the merits of an application of a research idea/ hypothesis. We therefore demonstrate the usefulness of the change specification and classification methods by applying them to a software project case study. We make two key assumptions with the case study that the project is in a state where the requirements elicitation has occurred and the process diagram has been established. We have already used a simple case study as a running example. The case study introduced in this section enable us to illustrate the versatility of the methods by way of using various change focus, various change types and how the outcome of the change classification differs with the need for input/output modifications.

5.1. The case Study

Figure 6 represents a partial system design diagram of a course management system adopted from [142]. The diagram illustrates the relationships and some dependencies the activities have with each other. The relationships denoted in the diagram can be defined as follows:

- Requires (Req): An activity A_1 requires an activity A_2 if A_1 is fulfilled only when A_2 is fulfilled. A_2 can be treated as a pre-condition for A_1 [142].
- Refines (Ref): An activity A_1 refines an activity A_2 if A_2 is derived from A_1 by adding more details to it [142].
- Contains (Con): An activity A_1 contains information from $A_2...A_n$ if A_1 is the conjunction of the contained information from $A_2...A_n$ [142].

The identification of these relationships is beneficial in determining the impact of change when applying our methods to the case study.

The detailed purpose of each activity is described as follows:

A1. The system allows end-users to provide profile and context information for registration.

A2. The system provides functionality to search for other people registered in the system.

A3. The system provides functionality to allow end-users to log into the system with their password.

A4. The system supports three types of end-users (administrator, lecturer and student).

A5. The system allows lecturers to set an alert on an event.

A6. The system maintains a list of events about which the students can be notified.

A7. The system notifies the students about the occurrence of an event as soon as the event occurs.

A8. The system actively monitors all events.

A9. The system notifies students about the events in the lectures in which they are enrolled.

A10. The system allows students to enroll in lecturers.

A11. The system allows lecturers to send e-mail to students enrolled in the lecture given by that lecturer.

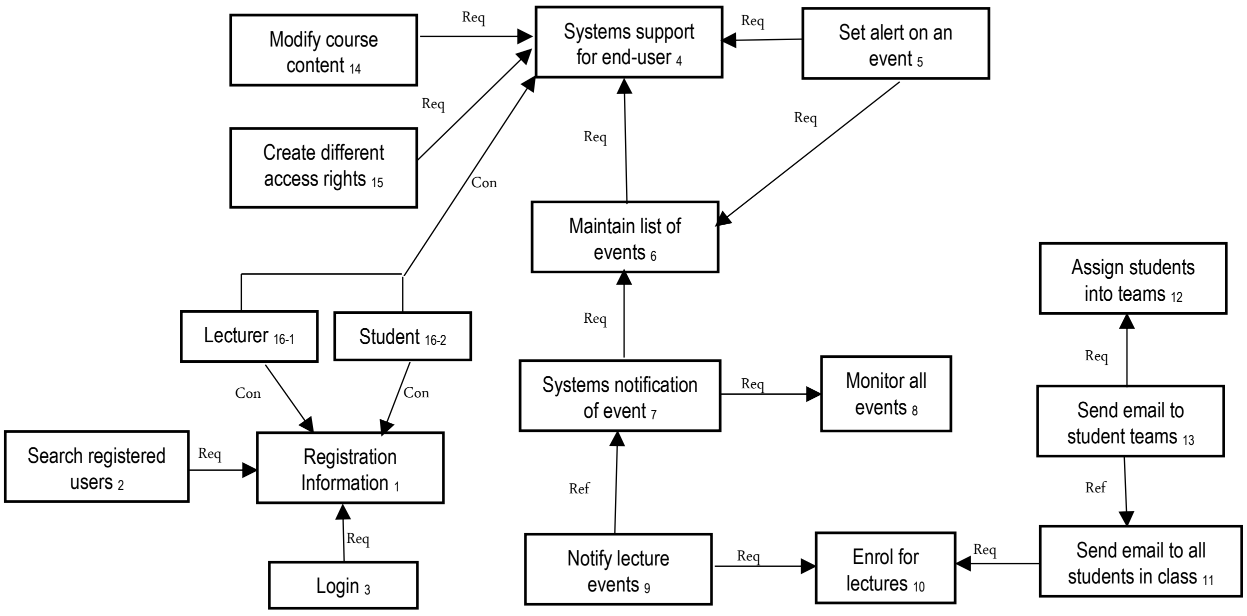
A12. The system allows students to be assigned to teams for each lecture.

A13. The system allows lecturers to send e-mail to students in the same group.

A14. The system allows lecturers to modify the content of the lectures.

A15. The system gives different access rights to different types of end-users.

A16. The system supports two types of end-users (lecturer and student) and it will provide functionality to allow end-users to log into the system with their password.



Modify course content 14
Req
Systems support for end-user 4
Req
Set alert on an event 5
Create different access rights 15
Req
Con
Req
Req
Maintain list of events 6
Assign students into teams 12
Lecturer 16-1
Student 16-2
Req
Req
Con
Con
Systems notification of event 7
Req
Monitor all events 8
Send email to student teams 13
Search registered users 2
Req
Registration Information 1
Ref
Ref
Req
Notify lecture events 9
Req
Enrol for lectures 10
Req
Send email to all students in class 11
Login 3

Figure 6: Partial system design diagram of a course management system

5.2. Applying them to the case study

The example consists of two scenarios, where we apply the specification and classification methods. These scenarios are based on our observations as university academics who use similar course management systems. The following hypothetical new requirements are identified:

1. In an emergency, it would be more effective to send an SMS notification to students as well as an email.
2. Marking attendance manually tends to be rather ineffective, especially when a census needs to be carried out. It would be better to mark attendance electronically.

The application of the implementation template yields the following results.

Change 01	**Possibility 01**	**Possibility 02**
Object	Enrol for lectures A_{10}	Send email to all students A_{11}
Purpose	Functionality enhancement	Functionality enhancement
Focus	Add	Modify
Additional Question	Need additional Input / Output? Y	Input/output modification? Y
Result		

Change Type	Add new activity	Inner property + Output interface modification
Action	Add new activity by using information from A_{10}	Modify A_{11} internally and the output interface

Table 10: Change 01 result

Change 02	**Possibility 01**
Object	Enrol for lectures A_{10}
Purpose	Identification of new requirement
Focus	Add
Additional Question	Need additional Input / Output? Y
Result	
Change Type	Add new activity
Action	Add new activity by using information from A_{10}

Table 11: Change 02 result

5.3. Discussion of the results

Tables 10 and 11 demonstrate how the specification and classification methods can be applied to this case study. The template given in Table 8 has been used for obtaining the result for each change.

Multiple possibilities can be created for each change event, depending on the event, availability of existing activities and various combinations that could be incorporated to realize the change. Such an instance has been provided for the 1st change event. In this change, the need to send SMS to students can be accomplished by either creating a new activity or modifying an existing activity (A_{11}). As such, when creating a new activity, it requires information from A_{10}. Therefore, the activity directly affected by the event is A_{10}. Rest of the table for the case study follows the process as explained through the simple stock control example.

In the second change event, we considered only one possibility. The requirement is to allow lecturers to mark attendance electronically. There doesn’t seem to be any existing activity that can be modified to serve this purpose, therefore the only option is to create a new activity. As such a new activity is created that requires student information, which is provided by A_{10}.

Therefore, the activity directly affected by the event is A_{10} and the rest of the table also follows the same principle as explained through the simple stock control example.

This example demonstrates how the specification and classification methods can be used to generate multiple possibilities for a single change. This outcome provides decision makers with the option of choosing the most appropriate way of implementing the change. The example above illustrates the way these methods can help both business and IT personnel involved, analyse business changes and thereby assist in the change management process. At the business level, the business analyst can use Tables 1 and 2 to define and describe the requirements change without any ambiguities. As a result of this IT personnel are able to not only understand the change but also understand the need for change and identify the location of change.

6. Comparison with related work

We shall describe what the literature has said about the related work and concepts like taxonomies and classification which are important concepts in studying change identification and classification.

6.1. Taxonomies

1) Research analysing change uses a plethora of techniques in order to build a taxonomy that can be used to identify changes as well as their impact. One such mechanism is the use of requirement engineering artifacts, such as use cases. The research done by Basirati et al. [14] establishes a taxonomy of common changes based on their observation of changing use cases that can then be used in other projects to predict and understand RCs. They also contribute to this research space by identifying which parts of use cases are prone to change as well as what changes would create difficulty in application, contributing also to the impact analysis of change.

2) The taxonomy developed by Buckley et al. [15] proposes a software change taxonomy based on characterizing the mechanisms of change and the factors that influence software change. This research emphasizes the underlying mechanism of change by

focusing on the technical aspects (i.e. how, when, what and where) rather than the purpose of change (i.e. the why) or the stakeholders of change (i.e. who) as other taxonomies have done. This taxonomy provides assistance in selecting tools for change management that assist in identifying the changes correctly.

3) McGee and Greer [16] developed a taxonomy based on the source of Requirements Change (RC) and their classification according to the change source domain. The taxonomy allows change practitioners to make distinctions between factors that contribute to requirements uncertainty, leading to the better visibility of change identification. This taxonomy also facilitates better recording of change data which can be used in future projects or the maintenance phase of the existing project to anticipate the future volatility of requirements.

4) Gosh et al. [17] emphasize the importance of having the ability to proactively identify potentially volatile requirements and being able to estimate their impact at an early stage is useful in minimizing the risks and cost overruns. To this effect, they developed a taxonomy that is based on four RC attributes i.e. phases (design, development and testing), actions (add, modify and delete), sources (emergent, consequential, adaptive and organizational) and categories of requirements (functional, non-functional, user interface and deliverable).

5) The taxonomy established by Briand et al. [18] is the initial step in a full-scale change management process of UML models. In their research, they establish that change identification is the first step in the better management of RCs. The classification of the change taxonomy is based on the types of changes that occur in UML models. They then use this taxonomy to identify changes between two different versions of UML models and finally to determine the impact of such changes.

6.2. Classifications

There are many benefits of using a classification, the main benefits being to manage change to enable change implementers to identify and understand the requirements of change without ambiguity [19, 111]. The classification of RC has been studied in various directions. Table 12 lists the different directions which have been the subjects of studies.

Direction	Parameters	Comment
Type [17, 19-23]	Add, Delete, Modify	The most common way of classifying change.
Origin [10, 17, 112]	Mutable, Emergent, Consequential, Adaptive, Migration	Derived from the places where the changes originated from.
Reason [12, 19, 20]	Defect fixing, Missing requirements, Functionality enhancement, Product strategy, Design improvement, Scope reduction, Redundant functionality, Obsolete functionality, Erroneous requirements, Resolving conflicts, Clarifying requirements, Improve, Maintain, Cease, Extend, Introduce	Helps determine the causes of change and understand change process and related activities.
Drivers [113]	Environmental change, RC, Viewpoint change, Design change	Helps change estimation and reuse of requirements.

Table 12: Direction is change classification

6.3. Other change identification methods

1) Kobayashi and Maekawa [1] proposed a model that defines the change requirements using the aspects where, who, why and what. This allows the system analyst to identify the change in more detail, resulting in better impact identification as well as risk and effort estimation. This method consists of verification and validation and can be used to observe the RCs throughout the whole lifecycle of the system.

2) The change identification method usually has a pre-established base upon which its semantics are built. Ecklund's [114] approach to change management is a good example of this. The approach utilizes use cases (change cases) to specify and predict future changes to a system. The methodology attempts to identify and incorporate the anticipated future changes into a system design in order to ensure the consistency of the design.

6.4. Identifying limitations and comparison

We use the work listed in Table 13 (discussed above) to describe the limitations of the existing work and compare our methods to define what has been achieved.

Technique	Limitations	What our methods can address
Basirati et al. [14] and Ecklund [114]	Can only be applied if use cases are available or used in the development process.	They are applied at a design phase, which enables the identification of changes at an early stage. Can be used as long as there is a form of design diagram of the system.
Buckley et al. [15]	It did not directly address issues arising from miscommunication of change.	They can be directly used for managing changes for the purpose of identifying changes.
McGee and Greer [16] and Ecklund [114]	They are limited to providing assistance in predicting change.	They provide a way of communicating change as well identifying them in an early stage as to where and how the change should be applied.
Gosh et al. [17]	Only used for identification of change.	Provide preliminary guidance on how to manage the changes.
Briand et al. [18]	Can be used only if UML models are available.	Can be used as long as there is a form of design diagram for the system.
Kobayashi and Maekawa [1]	This is a complex method for verifying changes.	They address change management issues arising from miscommunication.

Table 13: Comparison with the related work

An examination of the work reported above leads to the identification of four key limitations:

1. There is little agreement and commonality between the studies;
2. For the process of specification and classification of change to be used successfully, in our view it needs to be a part of the same process (change request); they complement each other by providing a better understanding of the requirements change;
3. There has been little emphasis on designing specification methods related to change; and
4. A common limitation of the above classifications is the lack of guidance in applying them to change management activities.

As a result, we believe that a void exists in the practical application of change specification and classification and our methods address this research gap.

7. Conclusions and future work

The purpose of the change specification and classification methods presented in this paper is to manage requirements change by improving change communication and elicitation. Under normal circumstances, business changes flow from the business side to the IT side. Therefore, the impact of this study belongs to both these categories i.e. business and IT. First, considering the business side, we ensure a requirements change has been clearly communicated to the IT side. As mentioned earlier, there is often difficulty in promoting effective dialogue about the nature of the change between these two parties. Therefore, a change specification method would be essential for business analysts in communicating change.

Second, on the IT side, it is critical that change enablers have a mutual understanding of not only the precise nature of the change but also the reason for its existence, i.e. its purpose. This insight translates into a better realization of the requirements change. Equally important is a quick response from IT in redesigning the system to suit the requirements change. The three main categories: object, purpose and focus of the change specification method enhance understanding while the classification of the change type and the resulting action assists system designers to incorporate the change into the system design much faster.

Given the above impact of our methods, we believe that there are substantial benefits of specification and classification methods that will lead to improvements in the change management process. In our view, the benefits of these methods are:

- Promotes a mutual understanding of requirements change between business and IT through the templates provided by Tables 2 and 3.
- Supports the decision-making process by helping to determine the need for the change.
- Assists in determining the best course of action in implementing the requirements change through Table 7.

In future work, we plan to use the multiple change identification possibilities to evaluate the best course of action to enable system designers to respond quickly to change requests. Furthermore, we suggest it will be useful in evaluating the interdependencies of these change requests as they relate to interdependencies of the system requirements and its implementation. Identification of interdependencies between changes can lead to identification of conflicts between requirement changes. Also, it would be valuable if it were possible identify the difficulty level and priority of the changes so that resources such as time and effort can be allocated more effectively. Identifying the difficult level of the change would further result in assisting the decision of the plausibility of implementing the change.

Chapter 4
A Method of Requirements Change Analysis

4.1 Preface

In order to assess whether a requirements change should be implemented, it is important to evaluate its impact on the existing system. This is due to the fact that requirements are not independent entities and can have very complex relationships. Therefore, a change made to one requirement can have a ripple effect on other requirements. Traceability is a popular technique used in many change analysis methods to trace the impact of a change through the existing software system. However, traceability techniques have a few drawbacks that can potentially outnumber their benefits. The main purpose of this chapter is to introduce a change analysis method that uses a different technique to identify the impact of change. We use the changes themselves to identify the interconnections they may create at implementation time. The idea is to be able to map the changes to the system activities, which eliminates the need for traceability.

This chapter consists of a paper that investigates the research space on requirements change analysis with an emphasis on impact analysis in order to produce the proposed method, which is a continuation of the specification and classification methods presented in chapter 3. This can be considered as the second phase of the requirements change management process introduced in this thesis. The outcome of this method is used for the final stage of the RCMP presented in chapter 5.

4.2 Publication

S. Jayatilleke, R. Lai, and K. Reed, "A method of requirements change analysis," *Requirements Engineering*, pp. 1-16, 2017, DOI: 10.1007/s00766-017-0277-7.

Manuscript title	**Publication status**	**Nature and extent of candidate's contribution**	**Nature and extent of co-authors' contribution**
S. Jayatilleke, R. Lai, K. Reed "A Method of Requirement Change Analysis". Requirements Engineering	Published online in accordance with Requirement Engineering and Springer guidelines.	**Eighty percent** contribution by the candidate. This included gathering information, drafting and revisions of the manuscript.	**Twenty percent** contribution by co-authors. This included discussions of ideas expressed in the paper, critical review and submission to the journal.

Signed : [illegible] Date : 12/02/18

(S. Jayatilleke)

Signed : [illegible] Date : 12/02/18

(R. Lai) (on behalf of co-authors)

A Method of Requirements Change Analysis

Abstract – Software requirements are often not set in concrete at the start of a software development project; and requirement changes become necessary and sometimes inevitable due to changes in customer requirements and changes in business rules and operating environment; hence, requirements development, which includes requirements changes, is a part of a software process. Previous research reports that correcting requirements errors late costs many times more than correcting them during the requirements development phase. There is, hence, a need to manage them well and to analyze them in order to identify the impacts, difficulties and potential conflicts with existing requirements. Most studies on requirements change analysis are done at the source code level while paying less attention to the initiation of changes at a higher level. In this paper, we present a method of requirements change analysis based on the changes themselves which are initiated at higher levels. This method consists of three steps: namely, (1) analyzing the change using functions, (2) identifying the change difficulty; and (3) identifying the dependencies using a matrix. We illustrate the usefulness of our method by applying it to a course management system of a university.

Keywords*—Requirements change, dependency matrix, change interdependencies, change prioritization, impact analysis*

1. Introduction

Currently, software systems are becoming increasingly complex with system requirements being inherently changeable during all stages of the development life cycle. According to Bohem [217], “correcting requirements errors late can cost up to 200 times as much as correcting the errors during the requirements phase”. The size and complexity of software systems make change management costly and time consuming. The application of change management at the earliest possible point in the software development cycle has the potential to improve cost control. The complexity of the system further hinders the process of identifying the impact of changes on the existing system [131].

When addressing a change, most requirements cannot be treated as independent as very complex relationships can exist between them. As a result, an action performed on one requirement may have unexpected impacts on another [24-28]. Therefore, there is a need to identify requirement interdependencies. One of the most popular mechanisms for dealing with this is requirements traceability. Traceability is the ability to describe and follow the life of software artefacts [218] and is largely achieved by manually documenting different aspects of transformations of software development artefacts. As in any manual process, it is difficult, expensive and error prone. There are tools and approaches that automate change impact analysis, such as IBM Rational RequisitePro and DOORS and change impact analysis is implicit in model-driven development. In most of these, traces produced by these tools are only simple relations and their semantics is not considered. As a result, all requirements and architectural elements directly traced from the changed requirement are considered to be impacted. The requirements engineer then has to inspect all these candidate impacted requirements and architectural elements to identify changes, if there are any.

Although traceability is one of the best ways to identify the impact of change on the system, the greatest challenge of maintaining traceability is that the artifacts under consideration continue to change as the system evolves [35, 131, 219, 220]. A study conducted by Gotel and Finkelstein [131, 221] further elaborates on the difficulties of maintaining a traceability scheme. Among these difficulties (see[221]) are informal development methods, insufficient resources, time and cost for traceability, lack of coordination between people responsible for different traceable artifacts, lack of training in requirement traceability practices, imbalance between benefits obtained and effort spent implementing traceability practices, and failure to follow standards. Further, studies have also confirmed that the construction and maintenance of a traceability scheme proves to be costly for various reasons and commonly considered non-feasible from a financial point of view [124, 133, 222].

Based on above findings, we are motivated to propose a more efficient way to analyse the impact of requirement changes at an initial phase of the development process. One aspect of analysing changes would be to understand the dependencies between system activities and changes. Given the drawbacks of using requirement traceability (mentioned above), we are of the opinion that requirements changes themselves could be used to form part of the solution. The requirements changes will be mapped to the multiple activities of a system and as a result, dependencies and/or conflicts between these changes can be obtained. This analysis enables system developers to better manage requirements changes.

In this paper, we present a method of requirements change analysis based on the changes themselves which are initiated at higher levels, consisting of three steps: namely, (1) analyzing the change using functions, (2) identifying the change difficulty; and (3) identifying the dependencies using a matrix. These three steps give system analysts a better insight into which part(s) of the existing design, that will be affected. In step 1, the change analysis functions will expand the requested changes into more detailed steps enabling better insight to which part(s) of the existing design will be affected. The result(s) of step 1 will be used to identify the difficulty of implementation of the change(s), which is step 2. Finally, in step 3, the result(s) from steps 1 and 2 will be mapped to a matrix, which enables practitioners to identify the dependencies and/or the conflicts between the changes. When making a decision on the changes in terms of approval, setting priorities and understanding the implications, the results of all three steps will be taken into consideration. We demonstrate the usefulness of our method by applying it to a course management system of a university.

2. Rationale of the research approach

In order to establish a baseline for the current work, it is important that this is effectively an extension of the work done in [76]. It is important to correctly identify the required changes before they can be analysed. This identification requires the change to be communicated clearly and be identified with respect to what type it is. This is the reason why this piece of work is based on [76], which describes a technique to specify and classify requirement changes. The outcome of this specification and classification technique results in a clearer communication of changes between business and IT professionals and their identification of the changes. The specification and classification method [76] is only the initial phase of change requirement analysis. The natural flow is to extend this method to analyze further the impact of changes. Therefore, the identification of changes as proposed in [76] has been deemed necessary as the preliminary step for the current method.

We designed our work based on previous work done in the same area [18, 76, 140, 144, 145, 223]. Our method takes a decision-maker point of view in analysing changes, and is based on past research conducted using the same concept where the importance of this point of view has been established [76, 140, 145]. Our method establishes both direct and indirect impact analysis which is where the concept originated [18, 144]. As described in the following section, the proposed method employs a number steps to analyse impact due to the ease of both applying the method and understanding the outcome, which is a similar concept to that used in [144]. The method itself takes into consideration several different techniques that stem from other research. Work done in [140, 223-228] inspired us to use a matrix to represent the change conflicts in a more visual capacity while further analysis was carried out using change analysis functions, which is similar to parts of the work done in [144]. The initiation of further analysis is based on identifying the changes correctly using a change taxonomy that is adopted from [18, 76]. The method also attempts to prioritize changes based on a identification of the impact caused by the change[2]. The basis for this identification and prioritization was formed using [144] and [18].

Previous work	**Concepts extracted**	**Application to the analysis method**
Jayatilleke & Lai [76]	Change specification and classification method, decision makers point of view	This is the initial phase of the analysis method and is used as the baseline of the current method.
Li et al. [140]	Impact analysis algorithm, Interdependency graph and traceability matrix, decision makers point of view	The algorithm provided the idea for the implementation of functions in step 1 and the graph and matrix influenced the matrix in step 3.
Hassine et al. [145]	Use case map (UCM) slicing algorithm for dependency analysis, UCM diagrams, decision makers point of view	The slicing algorithm influenced the functions in step 1 and the UCM diagrams provided the idea of having change diagrams for aiding description also in step 1.
Briand et al. [18]	Use of change taxonomy, Change impact analysis rule algorithm	Justification of using the specification and classification method of [76] was based on this change taxonomy concept and some parts of the algorithm formed parts of the functions in step 1.
Brynjolfsson et al. [223]	Matrix of change	The conceptualization of including a matrix in step 3 was based on this work as well as ways and means of identifying the interactions between activities of the system design diagram. The design of the matrix was also influenced by this.
Ali & Lai [144]	Use of several steps to identify the impact, change analysis algorithms	The use of several steps in the analysis method was influenced by this research as

[2] We point out that requirements changes may also be prioritised by client need, however, for the purpose of this work, we assume we are dealing with changes of nominally equal to client priority. In practice, the results of the analysis of the type described here may be used to influence client priorities.

		well as the functions in step 1 was largely influenced by this algorithm.
Wang & Capretz [224]	Service dependency matrix	Used to establish and justify the concept of the matrix in step 3.
Zhang et al. [225]	Dependency matrix	Used to establish and justify the concept of the matrix in step 3.
Omer & Schill [226]	Web services dependency matrix	Used to establish and justify the concept of the matrix in step 3.
van den Berg [227]	Dependency matrix and crosscutting matrix	Provided the main design concept for the matrix in step 3.
Li [228]	Component dependency matrix	Used to establish and justify the concept of the matrix in step 3.

Table 1: Use of literature in creating analysis method

In conclusion, the preliminary concepts for the requirements analysis method are based on our previous work [76], traceability techniques, dependency / change functions and the dependency matrix. How these elements correspond with each other and provide an analysis of the changes is discussed in the following sections.

3. The method

3.1 An overview

Change impact analysis techniques can be divided into two categories: those based on traceability analysis and those based on dependency analysis [144]. In most of these methods, we observe that conflicts and dependencies between the changes themselves have not been identified. Furthermore, the prioritization of these changes is either not undertaken or occurs at a separate level. To overcome these limitations, we propose the following:

1) A way of identifying dependencies between changes
2) A way of assigning priority through difficulty identification

An overview of the method is given in Figure 1. Using this method, change practitioners will be able to achieve the following:

1. Identify conflicts and/or dependencies between multiple changes.
2. Identify which system activities (herein after referred to as activities) are most affected by the changes and thereby determine the suitability of the changes.
3. Calculate the difficulty level of each change.
4. Depending on the difficulty level, assign an implementation priority to each of the changes.

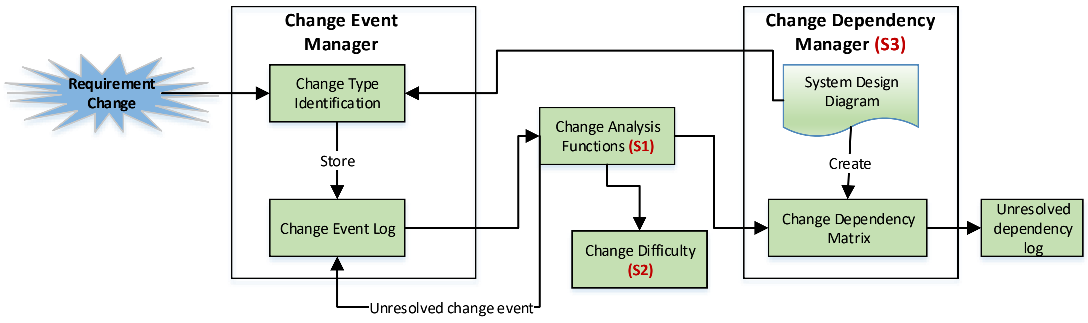
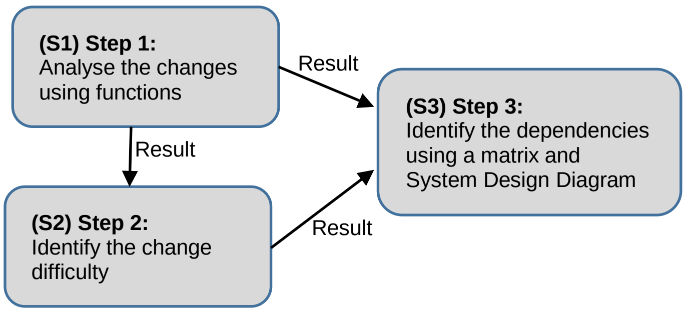


Figure 1: Change analysis method

Once a change has been identified through the Change Event Manager (CEM), the method follows three steps:



(S1) Step 1:
Analyse the changes using functions
Result
(S3) Step 3:
Identify the dependencies using a matrix and System Design Diagram
Result
(S2) Step 2:
Identify the change difficulty
Result

Figure 2: Three step analysis process

According to Figure 1, the CEM is the main system needed for the initiation of step 1 of the change analysis process. The CEM is responsible for the identification of the nature of the requested change which will be accomplished through the specification and classification method [76]. As explained in section 2 (above), further analysis of the change is difficult without this identification. The change analysis process is implemented using three steps for better clarity and ease of use. For each change identified by the CEM;

- Step 1 (S1) is for expanding the identified changes and for discovering the more detailed information for the implementation as a result of the changes. As shown in Figure 2, the two categories of change analysis functions (herein after referred to as functions) described in section 3.2 are employed for carrying out this step.
- Step 2 (S2) identifies the difficulty of implementing the change. The result of this will be used later for assigning a priority to each of the requested changes.

- Step 3 (S3) identifies the conflicts and/or dependencies between the required changes. As shown in in Figure 2, the key elements involved are the Change Dependency Matrix (CDM) and the System Design Diagram (SDD). The conflicts and/or dependencies between the changes are identified once the changes have been mapped to the matrix. The dependencies, which have been identified, can be between the changes themselves and/or between the changes and the activities of the system. The matrix will also be used for identifying the activities (of SDD) which have been most affected by the changes.

Detailed explanation of the three steps are given in the following sections.

3.2 The steps

Step 1 (S1): Analysing the changes

As shown in Figure 3, change initiated through a change request form is subject to the change specification and classification process [76], which completes the change type identification[3]. All change events that are identified are stored in a change event log. The change event log will have the dual role of being a repository for identified changes and a storage facility for unresolved changes. In this step, the changes that are stored in the change event log will be expanded using the functions. Once expanded, each change will have detailed information as to how the change is to be implemented. This will provide an idea for change practitioners to partially understand what activities of the existing system is going to be affected. Any change that cannot be evaluated using the functions is stored in the change event log for later resolution.

In [76], change focuses - add, delete, modify and relocation – are reported. Our functions are based on these change focuses, due to the fact that, each change identified using the CEM will be described using one of these change focuses. There are two categories of functions, namely primary and secondary;

[3] Please refer to [76] S. Jayatilleke and R. Lai, "A method of specifying and classifying requirements change," in *Software Engineering Conference (ASWEC), 2013 22nd Australian*, 2013, pp. 175-180: IEEE. for full details of the specification and classification method.

- The category of primary functions can be used for building a block of more complex functions. The need to do this is due to the facts that it is hard to project every possible way of implementing the changes and that practitioners can use this type of block to help them facilitate the changes.
- The category of secondary functions consists of more complex functions built by using a block of primary functions. These functions represent the change focuses and types described in [76].

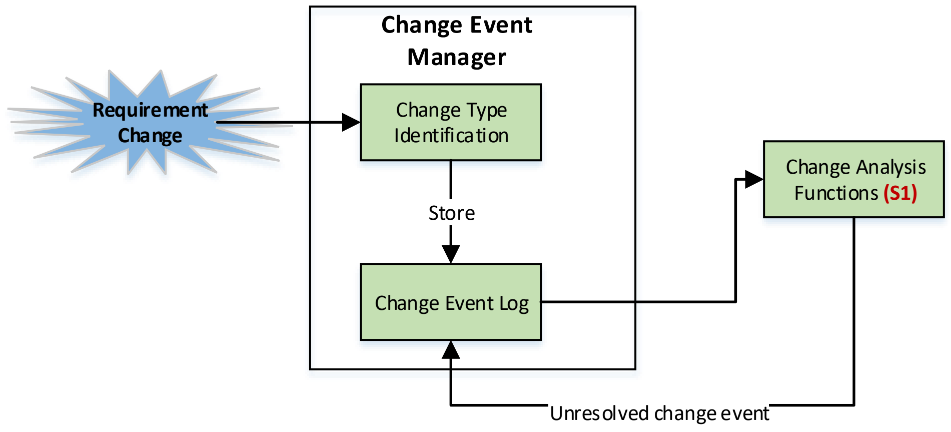


Figure 3: Step 1

The following terminologies are used for the functions:

A_N – New activity, A_O – Old activity, V – Value, A_T - Target activity, A_S - Input Sender, L – Link, Pt – Pointer, A_R – Relocating activity, A_C – Connected activity

The primary category consist of the following set of functions:

1. Function to create a new activity

   CreateFunc(String, V) $\rightarrow A_N$
2. Function to link a new activity with existing activities

   CreateLink(A_N, A_O, V)
3. Function to link existing activities

   CreateLink($A_{X\text{-}O}$, $A_{Y\text{-}O}$, V)
4. Function to delete an activity

   DeleteFunc(A_O)
5. Function to delete links between activities

   DeleteLink($A_{X\text{-}O}$, $A_{Y\text{-}O}$)

6. Function to modify inner property of an activity
   ModifyInner(A_T,V)
7. Function to modify input data of an activity
   ModifyIn(A_S, A_T, V)
8. Function to modify output data of an activity
   ModifyOut(A_S, A_T, V)
9. Function to create a pointer to an existing activity
   CreatePointer(Pt, A_T)
10. Function to delete a pointer
    DeletePointer(Pt)

The secondary category consist of the following set of functions:

The secondary functions are explained in Table 2. A function diagram is used for aiding the explanation of a secondary function. In each diagram, the roman numbers refer to the step numbers of the function. Each diagram is placed next to its corresponding function. In most diagrams, the function before the implementation of the change (left of the equal sign) and the function after the implementation of the change (right of the equal sign) are illustrated.

Function	Description
Add new activity	This function will be used to add a new activity to the system. This can be either with matched or mismatched interfaces.
Delete an activity	This function will be used to delete an existing activity of the system. Deleting an activity may have matched or mismatched interfaces.
Activity relocation	This function will be used to relocate an existing activity of the system from its current location to a new location. This may have matched or mismatched interfaces.
Merge activities	This function will be used to merge existing activities of the system. It could be any number of functions of the system.
Replace activities	This function will be used to replace an existing activity of the system. The replacement can be done by adding a new activity as well as using an existing activity of the system.

Table 2: Description of secondary functions

Note: Matched interfaced means that whatever the changes being made, the connected function interfaces do not have to be modified. With mismatched interfaces, the connected function interfaces need to be modified to implement the change. Explained in detail in [76].

1. Add new activity (A_N) Function (with matched interfaces)
   i.CreateFunc(String, V) →A_N
   ii.CreateLink(A_N, A_O, V)

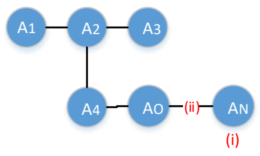
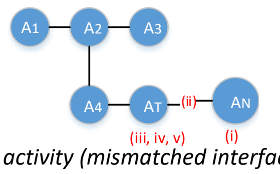


Figure 3.1: Add new activity (matched interfaces)



2. Add new activity (A_N) Function (with mismatched interfaces)
   i.CreateFunc(String, V) →AN
   ii.CreateLink(AN, AT, V)

Figure 3.2: Add new activity (mismatched interfaces)

{
i. ModifyInner(A_T,V) *If inner property modification is needed in target activity*
ii. ModifyIn(A_N, A_T, V) *If input data modification is needed in target activity*
iii. ModifyOut(A_N, A_T, V) *If output data modification is needed in target activity*
}

3. Delete an activity (A_O) Function (with matched interfaces)
   i. CreatePointer(Pt, A_O)
   ii. DeleteLink($A_{2\text{-}O}$, $A_{O\text{-}O}$)
   iii. DeleteFunc(A_O)
   iv. DeletePointer(Pt)

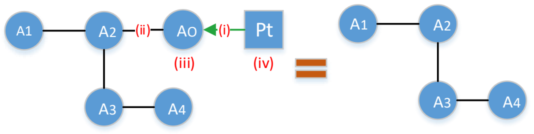


Figure 3.3: Delete activity (matched interfaces)

4. Delete an activity (A_O) Function (with mismatched interfaces)
   i. CreatePointer(Pt, A_O)
   ii. ModifyInner(A_C,V) *If inner property modification is needed in connected activity*
   iii. ModifyIn(A_O, A_C, V) *If input data modification is needed in connected activity*
   iv. ModifyOut(A_O, A_C, V)*If output data modification is needed in connected activity*
   v. DeleteLink($A_{X\text{-}O}$, $A_{Y\text{-}O}$)
   vi. DeleteFunc(A_O)
   vii. DeletePointer(Pt)

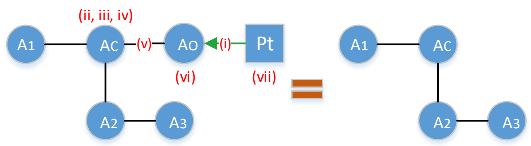


Figure 3.4: Delete activity (mismatched interfaces)

5. Activity relocation (A_R) Function (with matched interfaces)
   i. CreateLink($A_{R\text{-}O}$, $A_{T\text{-}O}$, V)
   ii. DeleteLink($A_{R\text{-}O}$, $A_{C\text{-}O}$)

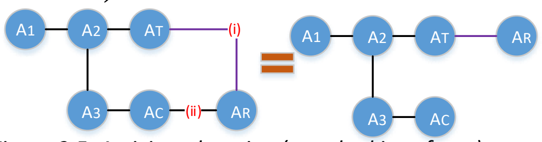


Figure 3.5: Activity relocation (matched interfaces)

6. Activity relocation (A_R) Function (with mismatched interfaces)
   i. CreateLink($A_{R\text{-}O}$, $A_{T\text{-}O}$, V)
   { *If modification is needed at target activity*
   ii. ModifyInner(A_T,V)
   iii. ModifyIn(A_S, A_T, V)
   iv. ModifyOut(A_S,A_T, V)
   }
   { * If modification is needed at current connected activity*

```
     v.  ModifyInner($A_C$,V)
    vi.  ModifyIn($A_O$, $A_C$, V)
   vii.  ModifyOut($A_O$, $A_C$, V)
       }
  viii.  DeleteLink($A_{R\text{-}O}$, $A_{C\text{-}O}$)
```

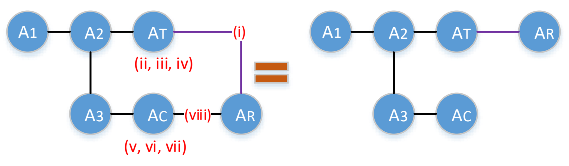


Figure 3.6: Activity relocation (mismatched interfaces)

```
7.  Merge activities Function (Merge $A_5$ and $A_6$)
      i.  CreateFunc(String, V) $\rightarrow A_N$
        for X=1 to no. of functions to merge
         ii.  CreateLink($A_N$,$A_{X\text{-}O}$ ,V)       v.
        iii.  CreateLink($A_N$, $A_C$,V)       vi.
         iv.  MergeFunc($A_N$, $A_{X\text{-}O}$, L)     vii.
       viii.  DeleteLink($A_{X\text{-}O}$, $A_{C\text{-}O}$)       x.
         ix.  DeleteFunc($A_{X\text{-}O}$)             xi.
        end for
```

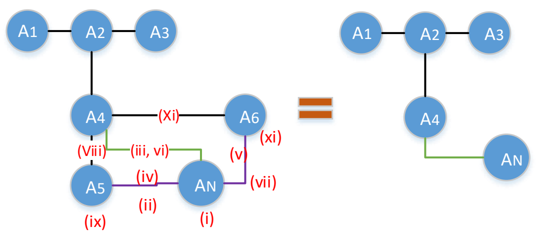


Figure 3.7: Merge activities

For merging two activities (A_5 and A_6) as shown in diagram, the loop will run twice and will produce the resulting diagram

8. Replace activities Function (Replace A_O with New activity A_N)

a) Replacing with new activity:

```
     i.  CreateFunc(String, V) $\rightarrow A_N$
    ii.  CreateLink($A_N$, $A_O$, V)
   iii.  CreateLink($A_N$, $A_C$, V)
    iv.  CreatePointer(Pt, $A_O$)
     v.  DeleteLink($A_{O\text{-}O}$, $A_{C\text{-}O}$)
    vi.  DeleteLink($A_{N\text{-}O}$, $A_{O\text{-}O}$)
   vii.  DeleteFunc($A_O$)
  viii.  DeletePointer(Pt)
```

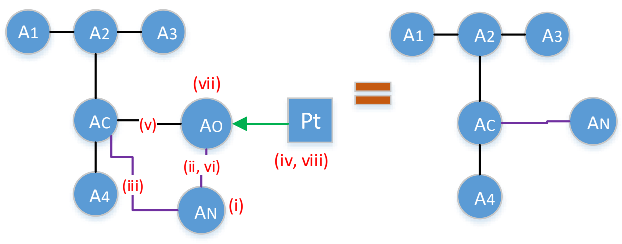


Figure 3.8: Replace activity with new activity

b) Replacing with existing activity (Replace A_R with existing activity A_Y)

c)

```
   i.CreatePointer(Pt, $A_R$)
     For X = 1 to no. of links in the replaced activity
  ii.CreateLink($A_{X\text{-}O}$, $A_{Y\text{-}O}$, V)
     end for
     For X = 1 to no. of links in the replaced activity
 iii.DeleteLink($A_{R\text{-}O}$, $A_{T\text{-}O}$)
     end for
  iv.DeleteFunc($A_R$)
   v.DeletePointer(Pt)
```

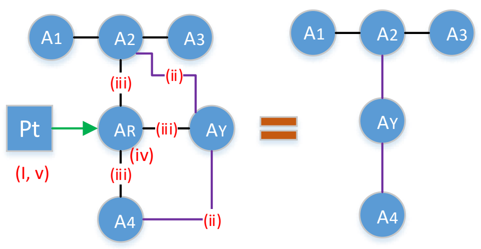


Figure 3.9: Replace activity with existing activity

Step 2: Identifying the change difficulty

The functions has a secondary purpose of assisting with the identification of the difficulty of the change. The difficulty here refers to an identification of how complex the implementation of the change will be. The expanded steps of each change are assigned a weight according to the rules given below. The total of these weights together with the number of activities affected by the change (also obtained from the functions) are used to determine the difficulty of the change. The activities affected directly are identified by expanding the changes through the functions. Indirectly affected activities are those connected to the directly affected activities. This can be identified through the SDD. It is also important to consider other artifacts such as databases which are affected by the administration of changes [229, 230]. The databases can be identified in the SDD. The identification rule would be if an activity is identified to be affected either directly or indirectly by a change, then check if the activity is associated with a database in terms of populating, updating and/or receiving information. If this condition is true, then the associated database is also deemed affected. The weights for the change categories are assigned based on [140] and [231]. In both these academic works, assigning change weight is based on their experience and in both studies the change weights are incorporated into calculations that calculate change complexity.

The rules of allocating weights for the steps in the function are as follows:

- All create functions will have the Add weight of 3
- All delete functions will have the Delete weight of 2
- All modify functions will have the Modify weight of 1
- All other functions are the combination of the main three functions i.e. create, modify and delete.

Table 3 will be populated with the above information in order to identify the change difficulty. The population of Table 3 is carried out as follows:

1. Identify the change focus for each step of the function of the change action.
2. Assign weight according to above rule for each identified change focus. And total these weights each change action.
3. Identify the activities affected (both directly and indirectly) by each change action based on the function steps using the SDD.

4. Identify the connected databases for each activity identified in the above step using the SDD.

Change action & Possibility	Function steps	Change focus	Change weight	Total	Activities affected (directly)	Activities affected (indirectly)	Affected databases

Table 3: Change difficulty identification

When identifying the difficulty of the calculation the following needs to be considered;

1. For each change action and its possibility, the change weight total, number of activities affected and the number of databases affected need to be considered.
2. When considering the activities, the number of directly affected activities take precedence over the indirectly affected activities.
3. Considering the databases, from experience we know that there are two main interactions between activities and databases; population of database and retrieval of information from a database.
4. When an activity is connected to the database in terms of population, the implications are higher as the activity can alter the data in the database. With retrieval alone, the consequences on the database is not as high due to the fact that in most cases the data manipulation occurs within the activity and does not update the database.
5. The difficulty of implementation of the change is a combination of the above four elements.

It is noteworthy that estimation of time to implement the change will also a play an important role on the decision of identifying change implementation difficulty. However, this estimation is outside the scope of this paper.

Step 3: Identifying the dependencies using a matrix

Dependency matrices have been used in several research work [224-228, 232] to identify conflicts and overlaps between requirements [233]. According to [233], this dependency identification technique is especially effective when there is a relatively small number of requirements. When this is not the case, the technique can still be applied by grouping requirements into smaller categories. Although this technique may be relatively simple, it is still quite versatile. The versatility of this concept allows the matrix method to be applied to many areas of analysis. In [224], the dependency matrix is applied to understand the evolution of web services. In [225], it is used to detect quality of service violations and to identify the cause of failures at the process level in service-oriented architecture. In [232], the technique is used to identify the interdependencies between projects, so that an organization can select the optimal projects to upgrade their technology. These various uses of dependency matrices prove that they are not only able to be used in many different areas but also for varying purposes.

Dependency can be represented as a graph or matrix-based model [226]. In our approach, we use the latter. The dependencies considered are between the changes and the activities. We established above that dependency matrices can be utilized in many ways. Therefore, in our method, the matrix is used to understand the dependencies and conflicts between changes. In addition, the matrix is used to visualize the impact of changes on activities.

According to Figure 3, the main element used to create the CDM is the SDD. The requirements for the SDD should be a design diagram, typically a UML [234] diagram which shows the relationships between different objects and activities. These relationships will assist in identifying the activities which are impacted by the requested changes. A dependency matrix (source × target) represents the dependency relation between source elements and target elements (inter-level relationship) [227]. Adopting the same concept, source elements (rows) are made up of the change focus [76] and the target elements (columns) are made up of all the activities affected directly and indirectly as identified in Table 2. In this matrix, a cell with a value denotes that the source element is mapped to the target element. Reciprocally, this means that the target element is impacted by the source element. Therefore as mentioned earlier, the dependencies identified can be between the activities due to changes and/or between changes themselves.

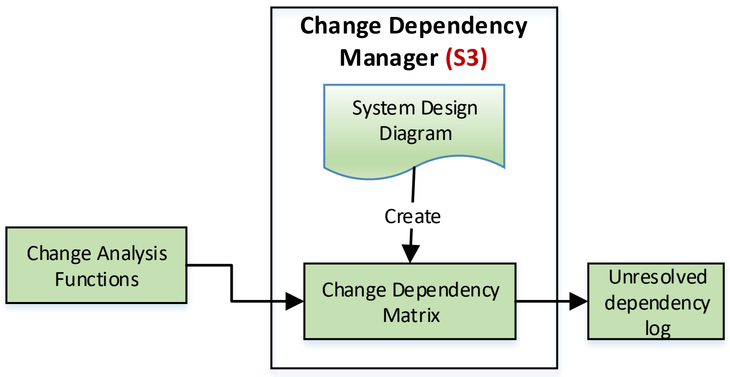

Change Dependency Manager (S3)
System Design Diagram
Create
Change Analysis Functions
Change Dependency Matrix
Unresolved dependency log

Figure 4: Step 3

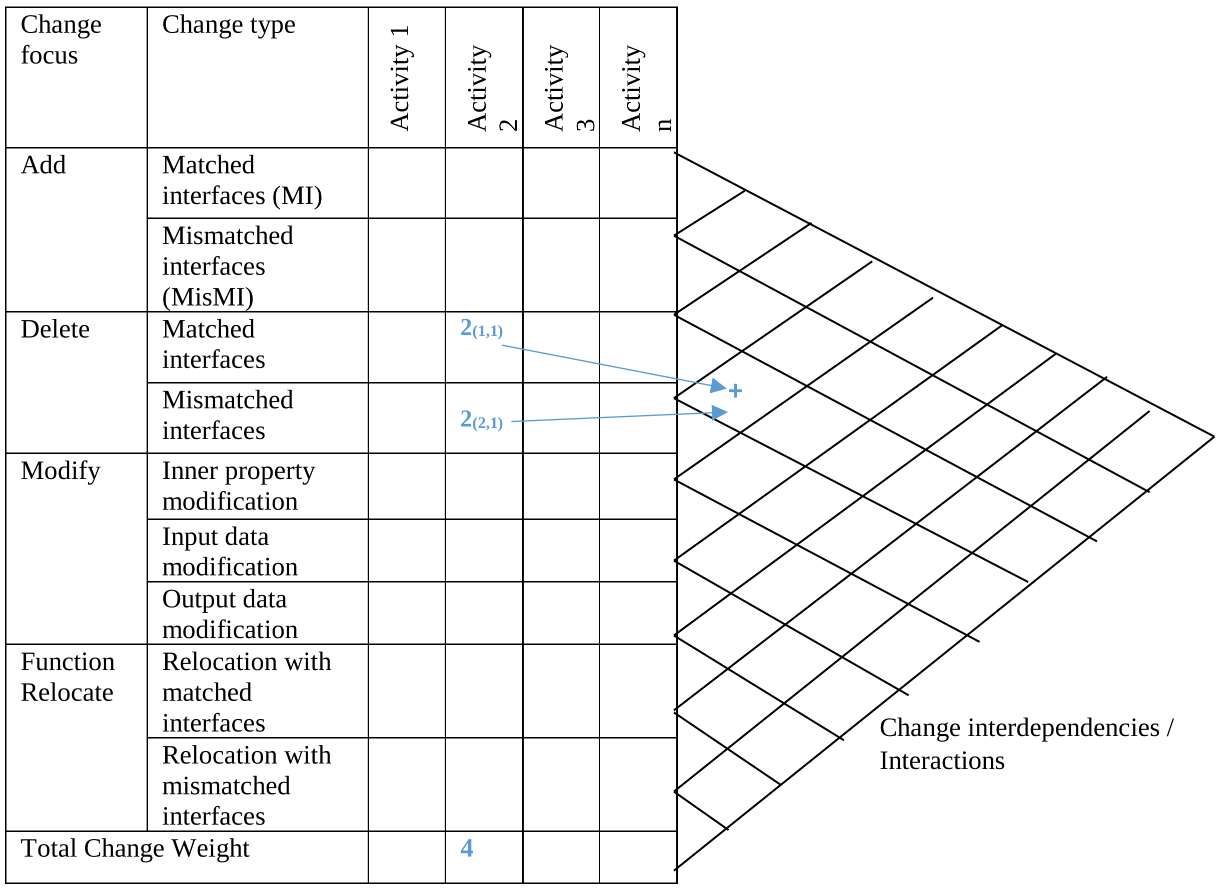
The result of application of functions will be applied to the CDM. As shown in Figure 4, any unresolved dependency events will be stored in a dependency log for later human-supported resolution. A change step identified though the function will be displayed in the matrix using the following rule:

- Change Focus Weight$_{\text{(Change No., Possibility No.)}}$
    - Change Focus Weight – numerical value assigned to each change focus as described in step 2
    - Change No. – A number given to each change that has been classified at the very beginning of the process
    - Possibility No. – If there is more than one possibility for the change to be implemented
- e.g.: Assuming this is the first change identified by the CEM with only one possibility and the change step considered is Add, then the change focus weight is 3. The change step therefore is represented in the matrix as $3_{(1,1)}$

The representation of the dependency matrix used for this step is given in Figure 5. The conventional appearance of the matrix has been slightly modified to suit the needs of our method. The triangular section in Figure 5 is used to visualize the conflicts and/or dependencies between changes. The dependencies are identified through observing activities that are affected by multiple changes to the system. If an activity is affected by more than one change, then the corresponding triangle is marked by a "+" symbol (see example in blue in Figure 5). The process of identifying the dependency is further explained in an application of the method to a case study. The total change weight produces a number that identifies (numerically) how each activity is impacted by the changes (see example in blue in Figure 5).

The purpose of calculating this value is to give change practitioners an idea of which activities in the system are most impacted by the changes and as a result, the need to give prioritized attention to such activities. The following rules are applied when calculating the total change weight:

a) If there is only one possibility of change, add all.
b) If there are changes with multiple possibilities, add each change with the same possibility number individually and then pick the possibility with the highest value.
c) If there are different changes, add all.
d) If there is a combination of the above two, first apply (a) followed by (b) and then (c).



Change focus	Change type	Activity 1	Activity 2	Activity 3	Activity n
Add	Matched interfaces (MI)				
	Mismatched interfaces (MisMI)				
Delete	Matched interfaces		$2_{(1,1)}$		
	Mismatched interfaces		$2_{(2,1)}$		
Modify	Inner property modification				
	Input data modification				
	Output data modification				
Function Relocate	Relocation with matched interfaces				
	Relocation with mismatched interfaces				
Total Change Weight			4		

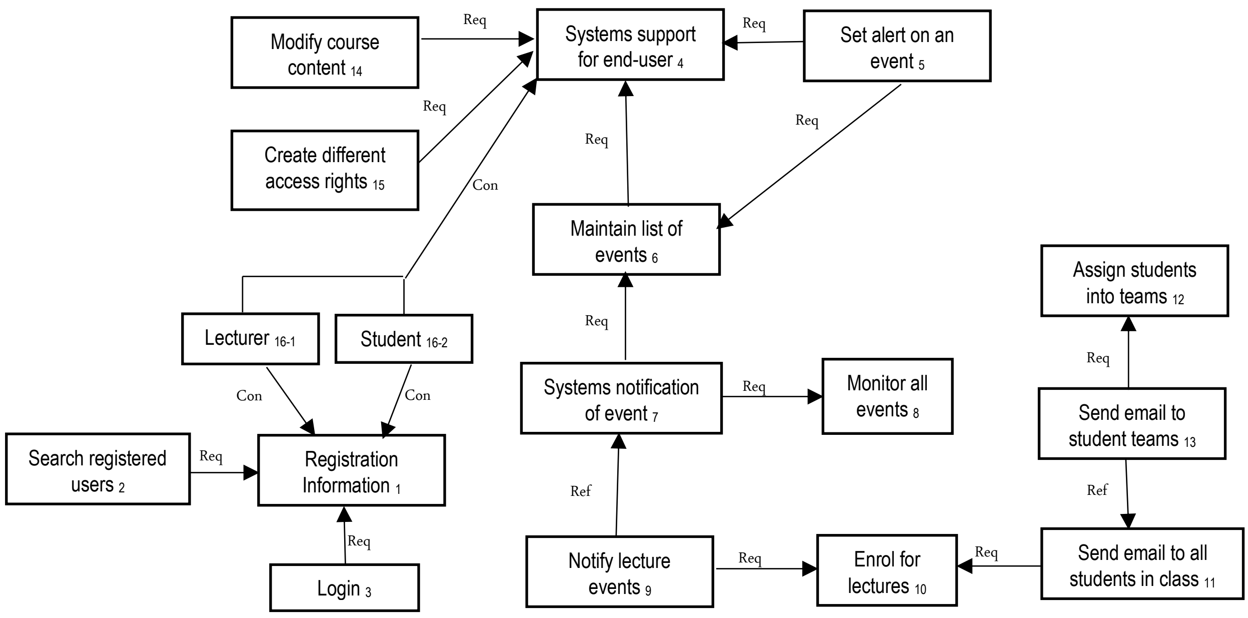
Figure 5: Change dependency matrix

4. A case study

We demonstrate the usefulness of our method in the following case study. Figure 6 represents a partial system design diagram of a course management system adopted from [142]. The diagram illustrates the relationships and some dependencies the activities have with each other. The relationships denoted in the diagram can be defined as follows:

- Requires (Req): An activity A_1 requires an activity A_2 if A_1 is fulfilled only when A_2 is fulfilled. A_2 can be treated as a pre-condition for A_1 [142].
- Refines (Ref): An activity A_1 refines an activity A_2 if A_2 is derived from A_1 by adding more details to it [142].
- Contains (Con): An activity A_1 contains information from $A_2 ... A_n$ if A_1 is the conjunction of the contained information from $A_2 ... A_n$ [142].

One of the main reasons for using this case study is the identification of the relationships. This identification is beneficial in determining the impact of change when applying our method to the case study. We have also included a table (Table 4) to describe the association of databases of this system to the activities given in the diagram.


Modify course content 14
Req
Systems support for end-user 4
Req
Set alert on an event 5
Req
Create different access rights 15
Con
Req
Req
Maintain list of events 6
Assign students into teams 12
Req
Lecturer 16-1
Student 16-2
Req
Con
Con
Systems notification of event 7
Req
Monitor all events 8
Send email to student teams 13
Search registered users 2
Req
Registration Information 1
Ref
Ref
Req
Login 3
Notify lecture events 9
Req
Enrol for lectures 10
Req
Send email to all students in class 11
Figure 6: Partial system design diagram of a course management system

The detailed purpose of each activity is described as follows:

1. The system allows end-users to provide profile and context information for registration.
2. The system provides functionality to search for other people registered in the system.
3. The system provides functionality to allow end-users to log into the system with their password.

4. The system supports three types of end-users (administrator, lecturer and student).
5. The system allows lecturers to set an alert on an event.
6. The system maintains a list of events about which the students can be notified.
7. The system notifies the students about the occurrence of an event as soon as the event occurs.
8. The system actively monitors all events.
9. The system notifies students about the events in the lectures in which they are enrolled.
10. The system allows students to enroll in lecturers.
11. The system allows lecturers to send e-mail to students enrolled in the lecture given by that lecturer.
12. The system allows students to be assigned to teams for each lecture.
13. The system allows lecturers to send e-mail to students in the same group.
14. The system allows lecturers to modify the content of the lectures.
15. The system gives different access rights to different types of end-users.
16. The system supports two types of end-users (lecturer and student) and it will provide functionality to allow end-users to log into the system with their password.

Database	Associated activities	Purpose of association
User Registration (staff and student) (D_1)	A_1	Populate and update database files
	A_2	Retrieve information
	A_3	Retrieve information of login details
	A_7	Retrieve information of student contact details
	A_{10}	Retrieve information for authentication
	A_{11}	Retrieve information of student emails
	A_{12}	Retrieve information of student contact details
	A_{15}	Retrieve information of user details
	A_{16}	Retrieve information of user details
Events (D_2)	A_5	Retrieve information
	A_6	Populate and update database files
	A_7	Retrieve information of event
	A_8	Retrieve and update of information
	A_9	Retrieve information of event
Student Enrolment (D_3)	A_9	Retrieve information of enrolled students
	A_{10}	Populate and update database files
	A_{11}	Retrieve information of enrolled students
	A_{12}	Retrieve and update information of enrolled students
	A_{13}	Retrieve information of enrolled students
Student Allocation (D_4)	A_{12}	Populate and update database files
	A_{13}	Retrieve information of allocated students
Course (D_5)	A_{14}	Populate and update database files

Table 4: Databases associated with the activities

5. Application of the method to the case study

The example consists of two scenarios, where we use the result from the specification and classification method [76] to obtain the output required for implementation of functions and the change dependency matrix. These scenarios are based on our observations as university academics who use similar course management systems. The following hypothetical new requirements are identified:

1. In an emergency, it would be more effective to send an SMS notification to students as well as an email.
2. Marking attendance manually tends to be rather ineffective, especially when a census needs to be carried out. It would be better to mark attendance electronically.

The change identification of the change analysis method yields the following specification and classification for the changes mentioned above.

Change 01	**Possibility 01**	**Possibility 02**
Object	Enrol for lectures A_{10}	Send email to all students A_{11}
Purpose	Functionality enhancement	Functionality enhancement
Focus	Add	Modify
Additional Question	Need additional Input / Output? Y	Input/output modification? Y
Result		
Change Type	Add new function	Inner property + Output interface modification
Action	Add new function by using information from A_{10}	Modify A_{11} internally and the output interface

Figure 7: Specification & classification of change 01

Change 02	**Possibility 01**
Object	Enrol for lectures A_{10}
Purpose	Identification of new requirement
Focus	Add
Additional Question	Need additional Input / Output? Y
Result	
Change Type	Add new function
Action	Add new function by using information from A_{10}

Figure 8: Specification & classification of change 02

The results given in Figures 7 and 8 are stored in the change event log and are then subjected to the three steps of the change analysis method.

Step 1:

The specified changes are expanded using the functions as follows:

Change action 1 – Possibility 1:

Add new activity (SendSMS) Function (with mismatched interfaces)

```
1. CreateFunc(String, V) →SendSMS
2. CreateLink(SendSMS, $A_{10}$, V)
   {
       3. ModifyInner(SendSMS,V)
       4. ModifyIn(SendSMS, $A_{10}$, V)
       5. ModifyOut($A_{10}$,Null,V)
   }
```

Change action 1 – Possibility 2

```
1. ModifyInner($A_{11}$,V)
2. ModifyOut($A_{11}$,Null,V)
```

Change action 2 – Possibility 1

Add new activity (eAttend) Function (with mismatched interfaces)

```
1. CreateFunc(String, V) → eAttend
2. CreateLink(eAttend, $A_{10}$, V)
   {
       3. ModifyInner(eAttend,V)
       4. ModifyIn(eAttend, $A_{10}$, V)
       5. ModifyOut($A_{10}$,Null,V)
   }
```

Step 2:

The results of step 1 can now be used to identify the change difficulty using the rules mentioned in the description of step 2. Table 5 shows the total change weight for each change and the number of activities affected (directly and indirectly) by each change.

Change action & Possibility	Function steps	Change focus	Change weight	Total	No of activities affected (directly)	No of activities affected (indirectly)	Affected databases
Change action 1 – Pos 1	1	Add	3	9	1 (A_{10})	2 (A_9 & A_{11})	D_1, D_2, D_3
	2	Add	3				
	3	Modify	1				
	4	Modify	1				
	5	Modify	1				
Change action 1 – Pos 2	1	Modify	1	2	1 (A_{11})	2 (A_{10} & A_{13})	D_1, D_3, D_4
	1	Modify	1				
Change action 2	1	Add	3	9	1 (A_{10})	2 (A_9 & A_{11})	D_1, D_2, D_3
	2	Add	3				
	3	Modify	1				
	4	Modify	1				
	5	Modify	1				

Table 5: Change difficulty identification-populated

It would also be beneficial to identify how each database is affected. We use Figure 9 illustrates the connectivity and relationship between the identified activities and the databases in Table 4. Based on the rules introduced in step 2 on determining the difficulty, the connection between A_{10} and D_3 has a higher implication as oppose to all the other connections, because A_{10} is connected to D_3 in terms of both population and retrieval. These implications will be discussed in section 6.

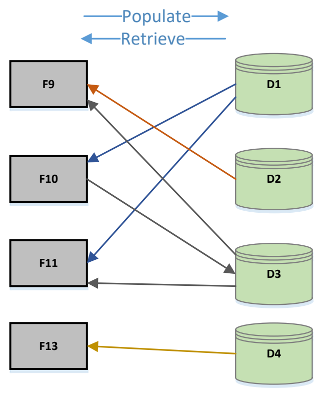


Figure 9: Activity-Database connectivity

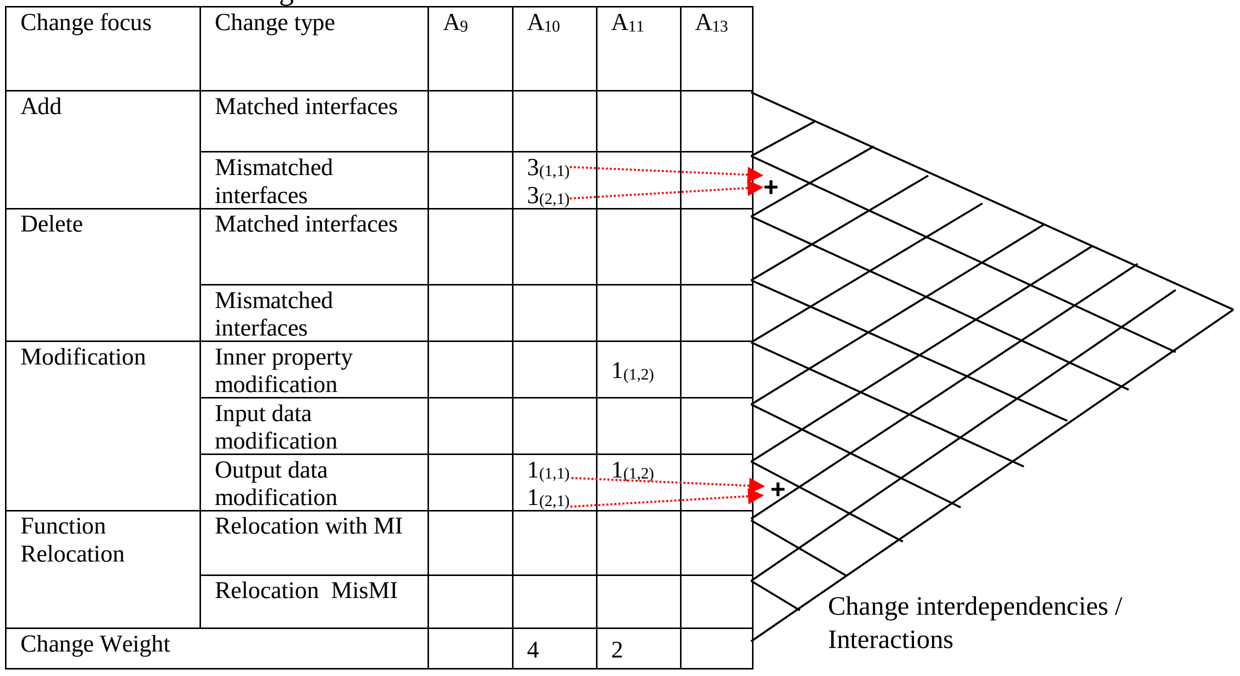
Step 3:

The final step of the method is to apply the results of the functions for each change into the CDM. In this step, the change focus mentioned in Table 5 for each change (and possibility) can be mapped into the dependency matrix easily. The rules mentioned in the description of step 3 in section 4 are incorporated in the change focus and the function steps when completing the matrix. Only the activities identified in the functions are mapped to the matrix. Table 6 shows how the matrix representations are obtained. The steps marked N/A correspond to the fact that it has no affiliation to the existing activities in the system and are therefore not represented in the matrix.

Change	Possibility	Function steps	Matrix representation
1	1	1. CreateFunc(String, V) →SendSMS	N/A
		2. CreateLink(SendSMS, A_{10}, V)	$3_{(1,1)}$
		3. ModifyInner(SendSMS,V)	N/A
		4. ModifyIn(SendSMS, A10, V)	N/A
		5. ModifyOut(A_{10},Null,V)	$1_{(1,1)}$
	2	1. ModifyInner(A_{11},V)	$1_{(1,2)}$
		2. ModifyOut(A_{11},Null,V)	$1_{(1,2)}$
2	1	1. CreateFunc(String, V) →eAttend	N/A
		2. CreateLink(eAttend, A_{10}, V)	$3_{(2,1)}$
		3. ModifyInner(eAttend,V)	N/A
		4. ModifyIn(eAttend, A_{10}, V)	N/A
		5. ModifyOut(A_{10},Null,V)	$1_{(2,1)}$

Table 6: Matrix representation

The finalized matrix is given as follows:



Change focus	Change type	A_9	A_{10}	A_{11}	A_{13}
Add	Matched interfaces				
	Mismatched interfaces		$3_{(1,1)}$ $3_{(2,1)}$		
Delete	Matched interfaces				
	Mismatched interfaces				
Modification	Inner property modification			$1_{(1,2)}$	
	Input data modification				
	Output data modification		$1_{(1,1)}$ $1_{(2,1)}$	$1_{(1,2)}$	
Function Relocation	Relocation with MI				
	Relocation MisMI				
Change Weight			4	2	

Figure 10: CDM

6. Discussion of the results

The following section provides an explanation of the use of the results of the CDM (Figure 10) and the change difficulty identification (Table 5 and Figure 9) obtained above to further analyze the requirements change.

According to Figure 10;

1. For A_{10} change No. 2 has a possible conflict or dependency with the first possibility of Change No. 1, therefore the **+** mark on the dependency section for both ‘Add’ and ‘Modification’ functions. The realization of this conflict is such that when applying these two changes, the change implementers need to be mindful of the possible ripple effect it might have on the activity and connecting activities.
2. A_{11} is not identified as impacted (with a + sign) as the steps are for the same change. Therefore, there are no dependency conflicts in applying this change.
3. Change weights in the dependency matrix are calculated (using rules of step 3) as follows :

- A_{10}: Follows rule (b)
- A_{11}: Follows rule (a)
- A_{10} has the highest change weight due to the changes and requires special consideration at the implementation level. This further clarifies the impact identified in the first finding.

According to Table 5 and Figure 9;

4. To identify the difficulty of change, the total change weight, the number of affected activities and databases need to be considered in unison. Therefore:
    a. Change 1 – Pos 1 and Change 2 have similar change weights and affect the same number of activities and databases both directly and indirectly.
    b. Referring to Figure 9, A_{10} is connected to D_3 in terms of population. Therefore the implication is higher. This condition is same in both Change 1 – Pos 1 and Change 2 and therefore will have similar implementation difficulty levels.
    c. A secondary observation is that out of the databases affected, D_3 has the highest connectivity level as well as the highest implication and therefore will need prioritized attention.
    d. Given that change 1 – Pos 1 and Change 2 have similar high difficulty levels, the same priority can be assigned. In addition, it should be kept in mind that these two changes have a conflict as demonstrated through the matrix. The second priority can be assigned to change 1 – pos 2.
    e. Change 1 has two possibilities of which possibility 1 has a higher difficulty level than possibility 2 and also possibility 1 has a conflict with Change 2. Based on these conditions, if possibility 2 of Change 1 is selected for

implementation along with Change 2, the conflict can be avoided and the implementation becomes less difficult.

In the overview of the research (section 3), we introduced four benefits that can be achieved by change practitioners using this method. As such, through our case study application outcome, we can demonstrate that we have achieved these benefits as follows:

1. Visual representation of the conflict that can occur between changes through the conflict inflicted on A_{10} activity by two different changes.
2. The most affected activity due to these changes (A_{10}) is identified through the change weight calculation in the matrix.
3. The difficulty level of each change is realized through the difficulty identification table, which used the total change weights, number of activities and databases affected by the change. We are able to determine the difficulty level of both changes, including the different possibilities.
4. The difficulty level is then used to compare the changes to determine the priority level of implementation as well as recommendations on choosing different possibilities of changes.

7. Comparison with the related work

Most research in requirement change management focuses on full-scale solutions that encompass requirements identification, impact analysis, change prioritisation and change measurement. According to Kilpinen [235], there are three main groups of impact analysis relative to the technique used i.e. traceability impact analysis, dependency impact analysis and experimental impact analysis. Following is a brief overview of the related work and a comparison to our work.

Li [140] elaborates the importance of understanding the impact of change from a decision-maker's point of view. The traceability techniques used in [140] involve an interdependency graph and traceability matrix. According to Goknil [142], the lack of semantics in trace links causes imprecise results in change impact analysis and further derails the impact problem. As a solution, the method proposed in [142] deals with a requirements metamodel with well-defined types of requirements relations. These relations are formalized and are then used to define change impact rules for requirements. In both of these studies, change prioritization and an understanding of the difficulty of applying the change is missing.

Ali and Lai [144] propose a method for impact analysis for a global software development (GSD) environment. The method consists of three stages, starting with understanding change, analyzing these changes against different GSD sites and finally making decisions regarding the change based on the analysis. Understanding the change is carried out with respect to the requirements. The impact is calculated as an estimate of the extent of changes that either could directly or indirectly affect development work at different GSD sites. The method however lacks a mechanism to prioritize changes.

The method in [145] uses slicing and dependency analysis at the use case map specification level to identify the potential impact of requirement changes on the overall system. The approach establishes the importance of understanding the impact of change at a higher level of abstraction given the disadvantages of code level analysis. Similar to [140] and [142], the method lacks prioritization and has difficulty measuring the changes. Similar to [145], the work done by Briand et al. [18] uses a UML model-based approach where the UML diagrams are first checked for consistency. This check ensures further analysis based on the diagrams is fault proof. The impact analysis is carried out using a change taxonomy and model elements that are directly or indirectly impacted by the changes. Though this method provides a prioritization technique, an understanding of the difficulty of applying the change is missing.

Analysing the above methods of analyzing requirements change, the following conclusions can be made:

- Most, if not all, methods mentioned with the exception of [18] focus predominantly on understanding which requirements are impacted by a change.
- Mechanisms for prioritizing the requested changes depending on their impact in order to facilitate better decision making are not presented.
- The level of difficulty in applying the change has not been discussed in any of these methods, which can be a major deciding factor in choosing to accept or reject the change and also possibly leading to cost and effort calculation.

8. Conclusions and Future Work

In this paper, we have presented a method of requirements change analysis based on changes initiated at higher levels. It consists of three steps: namely, (1) analyzing the change using functions, (2) identifying the change difficulty; and (3) identifying the dependencies using a matrix; and we illustrate the usefulness of our method by applying it to a university course management system. In order to explore the requirement change in greater depth, the method introduces two set of functions. These functions can be used to further expand changes identified through the method developed in [76]. The expanded changes provide an initial means of identifying which activities of the existing system may be affected by the change. In terms of understanding the impact on the system due to these changes, the method introduces two steps; namely, the identification of the difficulty level of the change and the CDM. The difficulty level of the change corresponds to the complexities involved in applying the change while the CDM provides a visual representation showing how each change affects the existing activities.

Through the application of our method to a case study as well as comparing our work with others, it can be concluded that the merits of our method include: (i) identification of conflicts and dependencies between requirement changes; (ii) allocation of priority to changes so that change practitioners are able to make informed decisions; (iii) understanding the difficulty of change so that early decisions can be made on the suitability of carrying out the change.

Though our method has only been applied to a University system, we believe that it still can be applied to a more complex system. Given that most real-world systems are complex, one possible way of applying our method and still achieving satisfactory results would be to categorize a complex system into smaller functional areas. Categorization of complex systems into smaller functional areas when analysing dependencies is supported by [233].

In future work, we plan to extend this approach in order to identify the effort that is needed to implement a requirement change, and to apply it to a more complex case study. The current work can be extended to look in-depth at the databases to identify exactly which database objects are impacted by the change. It would also be beneficial for the decision making process to estimate the time required to implement the change.

Chapter 5
A Method of Assessing Rework for Implementing Software Requirements Changes

5.1 Preface

Literature suggest the volatile nature of requirements to be an important cost driver. Such volatility can have a major impact on development efforts and project duration. In order to implement a requirements change, the existing system needs to be reworked. Prior to implementation of a change, change effort estimation should be carried out and in most situations, there are usually more than one-way to implement a change. In such situation estimations have to be carried out for all possible implementation options and this can be both tedious and time consuming. Therefore, prior to estimation, it would be beneficial to understand to which extent the system would be reworked for each implementation option so that the option with lesser rework can be used for estimation. It became evident through a review of related work that the term “Rework” has not been uniformly and clearly defined. The main purpose of this chapter is to define rework in the context of requirements change management and present a method of assessing rework for implementing software requirements changes. Based on the assessment we are able to identify the change implementation option with lesser rework.

The chapter includes a paper that investigates the research space on rework and change cost/effort estimation in order to produce the above method. The method is a continuation of the methods introduced in chapter 3 and 4. This is the third and last phase of the RCMP introduced in this thesis.

5.2 Publication

S. Jayatilleke and R. Lai (2018) "A method of assessing rework for implementing software requirements changes", *Requirements Engineering*.

Manuscript title	**Publication status**	**Nature and extent of candidate's contribution**	**Nature and extent of co-author's contribution**
S. Jayatilleke and R. Lai, "A Method of Assessing Rework for Implementing Software Requirements Changes". Requirements Engineering	Submitted manuscript prepared in accordance with Requirements Engineering author guidelines.	**Eighty percent** contribution by candidate. This included gathering information, drafting and revisions of the manuscript.	**Twenty percent** contribution by co-author. This included discussions of ideas expressed in the paper, critical review and submission to the journal.

Signed : [illegible] Date : 12/02/18

(S. Jayatilleke)

Signed : [illegible] Date : 12/02/18

(R. Lai)

A Method of Assessing Rework for Implementing Software Requirements Changes

Abstract

Software development is often affected by user/system requirements changes. To implement requirements changes, a system which is being developed needs to be reworked. However the term “Rework” has not been clearly defined in the literature. Depending on the complexity of the changes, the amount of rework required varies from some software module modifications to a non-trivial alteration to the software design of a system. The effort associated with such a rework obviously will vary too. To date, there has been scant research on rework assessment, and the relationship between it and change effort estimation is hardly understood. In this paper, we present a definition for rework, and describe a method of assessing rework for implementing software requirements changes. Our method consists of three steps: namely (i) change identification; (ii) change analysis; and (iii) rework assessment. To demonstrate the practicality that it enables developers to compare the rework between the different options available for implementing a requirements change and to identify the one which is less invasive and requires lesser amount of modifications to the software system design, we apply it to a course management system, where multiple options of implementation exist for one requirements change.

Keywords - Rework, Rework assessment, Requirements changes, Requirements change management, Software System Design Document

1. Introduction

Software development is often affected by changes in user/system requirements. Rapid changes in requirements are found to be one of the main cost drivers [5]; and they have a significant impact on development efforts and project duration [29, 30]. To implement requirements changes, a system in design phase or later (but not yet deployed) needs to be reworked on. However in the literature, the term “*Rework*” has not been uniformly and clearly defined as past practitioners and researchers considered terms like “reconsideration”, “re-instantiation”, “redoing” and “revision” as synonymous with rework, while the Oxford

Dictionary defines "*Rework*" as "making changes to the original version of something". Depending on the complexity of the changes, the amount of rework required varies from some software module modifications to a non-trivial alteration to software design of a system. The cost associated with such a rework obviously will vary too.

According to our systematic review on Requirements Change Management (RCM) [236], there are three main components in managing requirements changes: change identification, change impact analysis and change cost/effort estimation. Effort estimation is about calculating and predicting the effort of a set of activities before they are actually performed [157, 237]; and effort is a value usually expressed in terms of time and/or dollars. Subsequently, change effort estimations are to predict the cost and time required for implementing a change. Such estimations are important as underestimation can result in budget overrun, poor quality and delay in project completion; whereas overestimation may result in the allocation of too many resources which will cause inefficiency [157]. Accurate estimation can also help assess the feasibility of implementing a change, prioritize the implementation of the requested changes and determine the cost of the implementation of a change.

Prior to conducting a change effort estimation, we need to have a better understanding of the extent to which and how a system would be reworked as it is possible to have more than one option for implementing a change and different options require different amounts of rework to be made to a system. In such a situation, estimation might need to be done for each option in order to determine its suitability. It should be noted that with the complexity of the changes requested and the number of implementation options available, change effort estimation can be a tedious and time consuming task. It would therefore be beneficial to have a method which can identify the implementation option which involves the lesser amount of rework, before any estimation is carried out; and a lot of time will be saved by not having to conduct the unnecessary estimations. Given the importance of rework for estimation, the relationship between them is hardly understood.

In our systematic review [236], we explained how existing estimation methods and models can be applied to effort estimation related to implementing requirements changes and pointed out the fact that general effort estimation models may not be suitable for estimating the effort of implementing a requirements change. There are a few models that deal specifically with requirements change effort/cost estimation as discussed in the related work section of our systematic review paper [236]. Most existing methods use expert judgment which is based on the experience of the estimator which is not a consistent component and expert judgement also relies on past project data which may not be applicable to a particular case where there is no historical data. It is therefore important that the interrelations and dependencies between systems functions are identified for estimating the effort/cost of changes as the dependencies will have an impact on an implementation. An inherent drawback in most existing estimation methods is that these dependencies are not well understood [157].

To date, there is limited research on the concept of rework for software development and assessing rework for implementing requirements changes. In this paper, we first present our definition of rework and then describe a method of assessing rework for implementing software requirements changes in the context of its definition. Our method consists of three steps: (i) identification of the change; (ii) identification of the activities within the software system design which are affected by the change; and (iii) assessing the rework required. Steps 1 and 2 are based on the concepts and ideas described in our two previously published papers and the results of applying Steps 1 and 2 enable Step 3 to be carried out. Step 3 involves the computations of: (i) Interaction Comparison (IC); (ii) Interaction Weight (IW); and (iii) Rework which is based on IC and IW. To demonstrate the practicality that it enables developers to compare the rework between the different options available for implementing a requirements change and to identify the one which is less invasive and requires lesser amount of modifications to the software system design, we apply the method to a course management system, where multiple options of implementation exist for one requirements change.

2. The concept of rework and our proposed definition

The concept of rework exists in fields outside software development. In the field of medicine, doctors may need to rework treatment plans for patients who have developed unexpected reactions; in the building industry, civil engineers may need to rework plans for the load bearing of a bridge depending on future traffic conditions; academics will need to rework course and/or subject material depending on assessment outcomes or feedback by students. Several studies in civil engineering defined rework as "the unnecessary effort of redoing a process or activity that was incorrectly implemented the first time" [238, 239].

Rework is common in software development due to changes emanating from clients, development environment, and laws of the government and society. We discussed the causes of these changes extensively in our systematic review [236]. A key activity in RCM is to identify the amount of rework required for the proposed changes, as this will have a significant impact on the time and cost of a project. Studies show that normally rework leads to additional effort and cost [240-245] of a project. However, a clear relationship between rework and effort estimation has not been understood/established. Some studies proposed methods for reducing the amount of rework [241, 246], yet the fact remains that there will still be a considerable amount of rework to deal with. In the agile software development environment, it encourages rework instead of attempting to eliminate it [236, 247]. Rework is often unavoidable as the understanding of a problem and its possible solutions evolve over time.

Rework is a central activity in the development of software. The cost of rework is said to reach or even exceed 50% of the total project cost [240-242, 248]. These costs are one of the main concerns in software development since it is an important parameter defining the success of software projects [243, 244]. According to Charrette [245], software developers spend 40-50% of their time on rework activities. Based on the above facts, rework is generally considered as an important software development activity. In software development, Zhao and Osterweil [246] define rework as "the re-instantiation of tasks previously carried out in earlier development phases in a richer context that is provided by the activities and artifacts that had been performed and created during subsequent phases". In a simpler manner, Ghezzi et al. [248] suggest that rework consists of "going back to a previous phase" of software development to redo decisions made or work carried out in that previous phase". It is clear that the concept of rework has been subject to different interpretations. In short, rework has not been uniformly and well defined [241, 246, 249].

Based on the discussion above and our systematic review findings, we are of the opinion that the concept of rework needs to be more narrowly focussed on the following items:

- Requirements changes are the reasons for doing it;
- Instead of being broadly considered as a software development activity, it is one which falls in the area of RCM; and
- It is closely related to change cost/effort estimation, which is also a RCM activity.

Our proposed definition of *rework* is therefore as follows:

“Rework in the field of software engineering is an activity within the area of Requirements Change Management (RCM), which makes modifications/alternations to a system which has a software design document and is being developed (pre-delivery) for implementing certain requirements changes, with the alternations/modifications normally introducing extra work and increasing the total amount of cost/effort for completing the software project; and assessing rework, a preliminary step to change cost/effort estimation which is another RCM activity, is about studying how a system needs to be modified/altered for implementing the changes.”

According to this definition, we establish that rework is an activity conducted prior to the delivery of a system. Given that a software design document is necessary, rework assessment can be applied to any stage of software development as long as a software system design document is available; and it is independent of the type of software development methodology (be it waterfall or agile). With Agile Software Development, a design document becomes available as the development progresses and therefore rework assessment becomes plausible.

Another noteworthy point is that there is a key difference between our definition of rework and maintenance. According to IEEE standard 1219, software maintenance is defined as “the process of modifying a software system or component after delivery to correct faults, improve performances or other attributes, or adapt to a changed environment”[250, 251]. The post-delivery nature of maintenance is also emphasised similarly in the ISO/IEC [ISO95] definition [251, 252]. The modifications to a system during the maintenance phase will always preserve the integrity of the software product [251, 253]. If the software design of a system needs to be altered substantially, the alternation will not be done as a piece of maintenance work but rework which will lead to new version of the software product. An example is that Microsoft usually release a newer version of its Windows operating system every period of say 3-4 years, or sometimes shorter.

3. Overview of the method of assessing rework

We anticipate that our method of assessing rework enable us to understand to what extent a system needs to be altered for implementing the required changes. Based on our previously developed methods of specification and classification [76, 254], we have identified that some requirements changes can be implemented in more than one way, which we refer to as change implementation possibilities/options. We aim to realize the following:

1. A numerical representation of the assessment of rework required to implement a requirements change for all possible implementation options.
2. Selection of the option which requires a lesser invasive to the software design of the system, ergo is of lesser rework.
3. Comparison of the assessment of rework between multiple requirements changes.

This method is a continuation of the findings of the specification and classifications methods [76, 254] and change analysis methods [255] previously established by the authors. The use of these methods in the rework method is detailed in Figure 1.

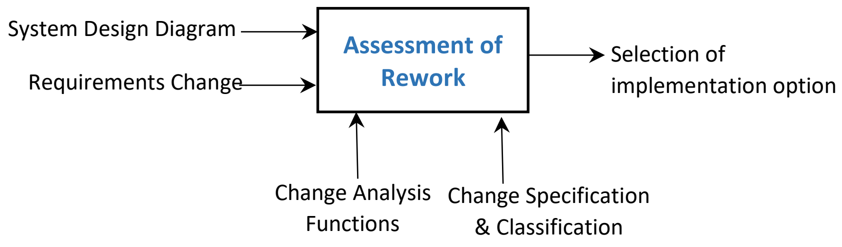

System Design Diagram
Requirements Change
Assessment of Rework
Selection of implementation option
Change Analysis Functions
Change Specification & Classification

Figure 1: Overview of the method

According to Figure 1, the method will use as input the requirements changes and the system design diagram (SDD). The output of the method is executed in three steps:

Step 1: Identification of the change

The change is identified and categorised using the methods which we have developed and reported in [76, 254].

Step 2: Identification of the system activities affected due to the change

Once the change is identified, we apply the change analysis functions of the change analysis method which we have developed and reported in [255]. As a result, the change is further exposed, enabling us to identify the activities that are directly affected by the change. Using the SDD, we then map the directly affected activities (DAA) to identify the indirectly affected activities (IdAA). The IdAA are the activities that are connected to DAA through input and/or output. IdAAs are considered in this assessment as modifications to a DAA which may have a direct impact on the activities associated with the DAA via the input-output links. The SDD is a design diagram. This can be any design diagram that shows the relationships between different objects and activities. Typically various forms of UML [234] diagrams such as activity diagrams, class diagrams, etc. can be used for this purpose.

Step 3: Assessing the rework required

Once all the activities related to the change are identified, we can assess the rework. In order to do this, we adopt the methods that are introduced in [256-258]. From [256] and [257], the concept we adopt is referred to as interaction frequency. This frequency refers to the ratio of the number of interactions (input-output) performed by the affected operations (of a change) and the number of interactions performed by all operations of the interface. A similar concept is used in [258] where instead, the number of interfaces are used. Given that the interactions between the activities are identified and indicated in the system diagram, we can use this concept to assess the rework and thereafter make a selection of the implementation option with a lesser rework.

4. The details of the Method

In this section, we describe the details of the method which consists of three steps.

4.1 Identification of the changes – Step 1

To identify a requested requirements change, we use the change specification and classification methods which we have developed and reported in [76, 254]; and a summary of them can be found in Appendix 1. Change specification denotes a way of specifying a change so that communication ambiguities between business and IT staff can be avoided. Once a requirements change has been initiated from the client side, this method will use the system design diagram as input to map the location of the change. In order to create the specification template, we use two established methods, i.e. Goal Question Metrics (GQM) [197] and the Resource Description Framework (RDF) [198]. We also use a set of additional questions to enable better identification when using the specification template output.

The change classification method uses the outcome of the specification template to expand on the type of change along with preliminary guidance on the action to be taken in managing the change. The classification itself is based on the concepts of the change taxonomy found in the existing change management literature [16, 32, 75, 214] and is refined using the unstructured interviews of practitioners in the field of change management. The outcome of the change classification will provide software developers with a better understanding of what the change is and offers preliminary guidance on how the change implementation can be carried out. The detailed change classification is shown in Table 1. The term link mentioned in Table 1 refers to the input-output connection between the activities. The term activity refers to the process activities in a design diagram.

Change focus	**Answer to Additional Question**	**Change type**	**Action**
Add	No	Matched links	Add new activity without changing the current activity or any connected links
	Yes	Mismatched links	Add new activity by changing the activity and/or connected links
Modification	No	Inner property modification	Modify the implementation of an activity without changing the connected links
	Yes	Input data modification	Modify the input link and internal properties of an activity
	Yes	Output data modification	Modify the output link and internal properties of an activity
Delete	No	Matched links	Delete activity without changing connected activities
	Yes	Mismatched links	Delete activity by changing connected activities and links
Activity Relocation	No	Relocation with matched links	Relocate existing activity without changing the activity or connected links
	Yes	Relocation with mismatched links	Relocate new activity by changing the activity and/or connected links

Table 1: Detailed change description

At implementation time, the key elements of the two methods (specification and classification) are incorporated into a single table (see Table 2). In the table, change number refers to the number given to each change as they are requested. The object, purpose and focus in Table 2 correspond to the specification method i.e. activity name according to the system design diagram (this is the activity affected by the change), the reason for the change and select from the Add, Delete, Modify or Activity relocation, respectively. Change type and action can be sourced from Table 1 based on the information provided for the object, focus and additional question, respectively. The option columns represent how each change may be described using different foci. This may not apply to all changes. This feature was added to the implementation template to provide more diversity and flexibility for communicating a change. Having multiple options also provides flexibility as to how the change can be implemented.

	Change No.	Option 01	Option 02	Option n
Specification Method	**OBJECT**			
	PURPOSE			
	FOCUS			
	Additional Question			
	RESULT			
Classification Method	**CHANGE TYPE**			
	ACTION			

Table 2: Template for implementation

4.2 Identification of the system activities affected by the change(s) – Step 2

We use a part of the change analysis method which we have developed and reported in [255] for expanding further the change identified; and a summary of this analysis method can be found in Appendix 2. Using this expansion, both DAAs and IdAAs are identified using the system design diagram. The change analysis functions are based on the change foci identified in [76, 254]: add, delete, modify and relocation. We use the category of primary change analysis functions to expand the changes. The category of primary functions can be used for building a block of more complex functions. The need to do this is due to the fact that it is hard to project every possible way of implementing the changes so practitioners can use this type of block to help them facilitate the changes.

The following terminologies are used for the functions:
The term activity in this method is used to represent process activities in the design diagram.
A_N – New activity, A_O – Old activity, A_T - Target activity, Pt – Pointer, A_R – Relocating activity, A_C – Connected activity
V – Value: the value passed onto the function for data manipulation
L – Link: the connection between two activities

The primary category consists of the following set of functions:

1) Function to create a new activity

CreateFunc(String, V) $\rightarrow A_N$

2) Function to link a new activity with existing activities

   CreateLink(A_N, A_O, V)

3) Function to link existing activities

   CreateLink(A_{X-O}, A_{Y-O}, V)

4) Function to delete an activity

   DeleteFunc(A_O)

5) Function to delete links between activities

   DeleteLink(A_{X-O}, A_{Y-O})

6) Function to modify inner property of an activity

   ModifyInner(A_T,V)

7) Function to modify input data of an activity

   ModifyIn(A_S, A_T, V)

8) Function to modify output data of an activity

   ModifyOut(A_S, A_T, V)

9) Function to create a pointer to an existing activity

   CreatePointer(Pt, A_T)

10) Function to delete a pointer

   DeletePointer(Pt)

Once the change has been expanded, the activities identified in the functions are mapped to the SDD. These are the DAAs. In the SDD, any activity connected as the input and/or output of a DAA is considered an IdAA.

In order to explain these steps, we consider the following running example.

Diskwiz is a company which sells CDs and DVDs by mail order. Customer orders are received by the sales team, which checks that the customer details have been completed properly on the order form (for example, delivery address and method of payment). If they are not, a member of the sales team contacts the customer to obtain the correct details. Once the correct details are confirmed, the sales team passes a copy of the order to the warehouse team to pick and pack, and a copy to the Finance team to raise an invoice. Finance raises an invoice and sends it to the customer within 48 hours of the order being received. When a member of the warehouse team receives the order, they check the real-time inventory system to make sure the discs ordered are in stock. If they are, they are collected from the shelves, packed and sent to the customer within 48 hours of the order being received, so that the customer receives the goods at the same time as the invoice. If the goods are not in stock, the order is held in a pending file in the warehouse until the stock is replenished, whereupon the order is filled. This process is illustrated in the following system design diagram.

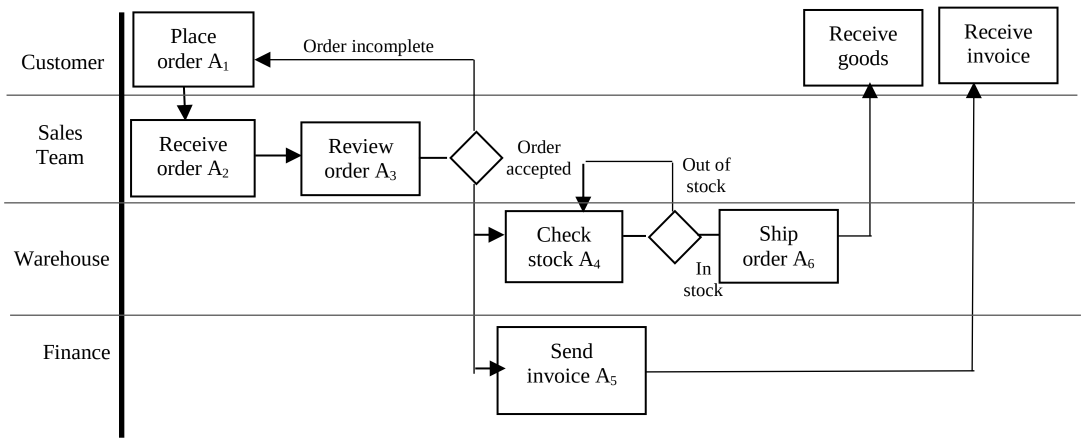


Figure 2: Diskwiz customer order fulfilment process diagram

The example consists of a scenario where the specification method is applied to specify the change and the change classification method is used to identify the change type and corresponding action. The scenario is as follows:

The management is not satisfied with some parts of the process and points out that the following issue should be rectified: "It is identified, due to a design error, there is no communication between Finance and the Warehouse to confirm discs are in stock so that the order can be shipped. Therefore Finance could be raising invoices when the order has not been sent."

One of the reasons for having no communication between Finance and Warehouse is because there is no communication between A_4 and A_5, where A_4 represent one activity of the Warehouse and A_5 represents Finance. Another way to view this would be that there is no communication between A_5 and A_6, where A_6 is another activity of the Warehouse. Based on these and the template of the specification and classification methods [76, 254], we obtain the following results for the identification of the change by applying step 1.

Table 3: Change classification outcome

Change 01	**Option 01**	**Option 02**	**Option 03**
OBJECT	A_4 and A_5	A_4 and A_5	A_5 and A_6
PURPOSE	Resolution of design error	Resolution of design error	Resolution of design error
FOCUS	Add	Modify	Modify
Additional Question	Need addition input/output? Y	Input/output modification? Y	Input/output modification? Y
Result			
Change Type	Add new function between A_4 and A_5 (Mismatched links)	Inner property modification and output data modification A_4 and input data modification of A_5	Inner property modification and output data modification A_6 and input data modification of A_5
Action	Add new function by changing the function and/or connected links of A_4 & A_5	Modify A_4 to send message to A_5	Modify A_6 to send message to A_5

In accordance to this example and Table 3, the change can be implemented using one of the three options. In step 2, we apply the preliminary functions from the change analysis method for the 3 options and we generate the following expansions of the change:

Option 1	Option 2	Option 3

CreateFunc(String, V) $\rightarrow A_N$
CreateLink(A_N, A_4, V)
{
ModifyInner(A_4,V)
ModifyIn(A_N, A_4, V)
ModifyOut(A_N, A_4, V)
}
CreateLink(A_N, A_5, V)
{
ModifyInner(A_5,V)
ModifyIn(A_N, A_5, V)
ModifyOut(A_N, A_5, V)
}
	ModifyInner(A_4,V)
	CreateLink(A_4, A_5, V)
	ModifyOut(A_4, A_5, V)
	ModifyIn(A_4, A_5, V)
		ModifyInner(A_6,V)
		CreateLink(A_5, A_6, V)
		ModifyOut(A_6, A_5, V)
		ModifyIn(A_6, A_5, V)

Table 4: Expansion of change options

Based on Table 4, we are able to identify the DAAs for each option. Then by mapping the DAAs to the SDD, we are able to identify the IdAAs for each DAA. In this paper when selecting the IdAAs, we consider only the first impact level. Investigation of further levels can be considered as a future enhancement, which is outside the scope of this paper.

Options	DAAs	IdAAs
1	A_4	A_3, A_6

	A_5	A_3
2	A_4	A_3, A_6
	A_5	A_3
3	A_5	A_3
	A_6	A_4

Table 5: Identification of DAAs and IdAAs

4.3 Assessing the rework required – Step 3

Through the numerical values generated, we are able to assess the rework to be carried out as a result of the change. In order to ensure the assessment of the rework is based on both the total interactions of the activities to be reworked as well as the difficulty level of implementing the change action, we use the number of affected interactions as well as the change weights introduced in the change analysis method [255]. The values for the weights are adopted from [140]. It has been established that in the change analysis method, each change action / type has a different difficulty level. Therefore, this difficulty level needs to be represented in the rework.

The assessment of the work required to implement a change involves the following calculations:

1. The interaction comparison (IC) of the affected activities (direct and indirect)
2. The interaction weight (IW) using the change weights of the affected activities (direct)
3. The rework based on IC and IW

As a result of the values generated from IC and IW, developers will have a numerical view of the assessment of the rework for implementing a change. If there are more than one option of implementation, then based on the combination of IC and IW, the developer can choose the lesser invasive option, which would result in the option with lesser rework.

When choosing the lesser invasive option, first preference is given to the lesser value of IC as this denotes lesser number of connections in the software design of the system will need to be altered. In the event that the IC value is the same for two or more options, IW will be considered. Use of IW is explained in the following sections.

4.3.1 Interaction comparison (IC) Calculation

Interaction comparison is the identification of the percentage of interactions that need to be altered in order to accomplish the required change. An interaction is a connection between two or more process activities (input-output links) in a SDD. This is in comparison to the total number of interactions identified in the SDD. Using the SDD, the following steps are used to calculate IC:

- For each activity (DAAs and IDAAs) involved in the change, identify the number of interactions. These interactions will be the number of connections each activity has with the other activities of the system.
- Identify the total number of interactions in the entire system.
- Calculate IC.

Using the above example, we show how the value of IC is calculated for all the options.

IC calculation for option 1:

The number of interactions for each identified activity based on Table 5 is as follows:

A_4 – has 2 interactions (Connected to A_3 and A_6)

A_5 – has 1 interaction (Connected to A_3)

A_3 – has 4 interactions (Connected to A_1, A_2, A_4 and A_5)

A_6 – has 1 interaction (Connected to A_4)

Considering all the interactions, the system design contains six activities. The interaction count for each activity is as follows:

A_1 – has 2 interactions (Connected to A_2 and A_3)

A_2 – has 2 interactions (Connected to A_1 and A_3)

A_3 – has 4 interactions (Connected to A_1, A_2, A_4 and A_5)

A_4 – has 2 interactions (Connected to A_3 and A_6)

A_5 – has 1 interaction (Connected to A_3)

A_6 – has 1 interaction (Connected to A_4)

The way of calculating the value of IC is adopted from [256].

$$IC_{CO} = \frac{N_I}{N_{TI}}$$

where CO is the Change Option number, N_I is the number of interactions per change action and N_{TI} is the total number of interactions for the system according to the SDD.

$$N_I = \sum_{x=1}^{n} N_{Ix}$$

where x is the number of activities affected by the change action and N_{Ix} is the interactions for each affected activity.

$$N_{TI} = \sum_{x=1}^{n} N_{TIx}$$

where x is the total number of activities of the system and N_{TIx} is the interactions for each activity.

Applying to the example option 1:

When calculating N_I we consider the interaction of all the activities (DAAs and IdAAs) of option 1 which include: A_4, A_5, A_3 and A_6 (extracted from Table5). Based on the interactions identified for these activities, N_I is;

$N_I = 2 + 1 + 4 + 1 = 8$

When calculating N_{TI} interactions of all the activities are considered. Based on the interactions identified for all activities, N_{TI} is;

$N_{TI} = 2+2+4+2+1+1 = 12$

$$IC_1 = \frac{8}{12} = 67\%$$

According to this value, when considering option 1 for change implementation, 67% of all the interactions have to be altered in order to implement the required change.

IC calculation for option 2:

The number of interactions for each identified activity based on Table 5 is as follows:

A_4 – has 2 interactions (Connected to A_3 and A_6)

A_5 – has 1 interaction (Connected to A_3)

A_3 – has 4 interactions (Connected to A_1, A_2, A_4 and A_5)

A_6 – has 1 interaction (Connected to A_4)

The total number of interactions is the same as that of option 1

Therefore;

$$IC_2 = \frac{N_I}{N_{TI}}$$

Applying the same principles as option 1;

$N_I = 2 + 1 + 4 + 1 = 8$

$N_{TI} = 2+2+4+2+1+1 = 12$

$$IC_2 = \frac{8}{12} = 67\%$$

According to this value, when considering option 2 for change implementation, 67% of all the interactions have to be altered for implementing the required change.

IC calculation for option 3:

The number of interactions for each identified activity based on Table 5 is as follows:

A_5 – has 1 interaction (Connected to A_3)

A_6 – has 1 interaction (Connected to A_4)

A_4 – has 2 interactions (Connected to A_3 and A_6)

The total number of interactions is the same as that of option 1

Therefore;

$$IC_3 = \frac{N_I}{N_{TI}}$$

Applying the same principles as option 1;

$N_I = 1 + 1 + 2 = 4$

$N_{TI} = 2+2+4+2+1+1 = 12$

$$IC_3 = \frac{4}{12} = 33\%$$

According to this value, when considering option 3 for change implementation, 33% of all the interactions have to be altered for implementing the required change.

4.3.2 Interaction weight (IW) Calculation

The interaction weight is the change weight corresponding to the directly affected interactions due to the requirements change. The change weight concept was established in the change analysis method [255]. The weights for the change categories are assigned, using the principles described in [140] and [231] and based on the knowledge they have gained in working in the industry as well as extensive research on requirements change management. In both studies the change weights are incorporated in mathematical formulas which compute a change complexity. IW adds depth to the IC value by providing a numerical representation of the difficulty level of implementing the change and how this relates to the interactions. The value of IW becomes further important in assessment and selection, when the value for IC can be the same for different options of a given change, as we demonstrated in the running example. We establish that the lower the IW, the less difficult it would be to implement a change. In order to calculate IW, the following steps are used:

- Identify the change types using the expanded change action steps (Step 2).
- Calculate the total change weight based on the change analysis method.
- Use the interactions and the total change weight to calculate IW.

In order to calculate IW, we consider only the activities directly affected by the change. This is because the identification of change types are acquired from step 2 where it only contains DAAs.

From the change expansion in step 2, we consider the change functions *Create, Modify* and *Delete* when calculating IW.

Using the same running example, we use the outcome of Table 4 to identify the change types as follows:

Option 1	Option 2	Option 3

CreateFunc(String, V) →A_N
CreateLink(A_N, A_4, V)
{
    ModifyInner(A_4,V)
    ModifyIn(A_N, A_4, V)
    ModifyOut(A_N, A_4, V)
}
CreateLink(A_N, A_5, V)
{
    ModifyInner(A_5,V)
    ModifyIn(A_N, A_5, V)
    ModifyOut(A_N, A_5, V)
}
	ModifyInner(A_4,V)
	CreateLink(A_4, A_5, V)
	ModifyOut(A_4, A_5, V)
	ModifyIn(A_4, A_5, V)
		ModifyInner(A_6,V)
		CreateLink(A_5, A_6, V)
		ModifyOut(A_6, A_5, V)
		ModifyIn(A_6, A_5, V)

Create Functions – 3 Modify Functions – 6 Delete Functions – 0	Create Functions – 1 Modify Functions – 3 Delete Functions – 0	Create Functions – 1 Modify Functions – 3 Delete Functions – 0

Table 6: Change weight identification

Using the weighting system introduced in the change analysis method, we develop Table 5 to calculate the change weight (CW):

- All create functions will have the Add weight of 3
- All modify functions will have the Modify weight of 2
- All delete functions will have the Delete weight of 1
- All other functions are a combination of the main three functions i.e. create, modify and delete

Change Type	Option 1	Option 2	Option n
Add	No. of functions × CW Add	No. of functions × CW Add	…. × ….
Modify	No. of functions × CW Mod	No. of functions × CW Mod	…. × ….
Delete	No. of functions × CW Del	No. of functions × CW Del	…. × ….
Total CW			

Table 7: Change weight calculation

Applying the findings of the running example of Table 6:

Change Type	Option 1	Option 2	Option 3
Add	3 × 3 = 9	1 × 3 = 3	1 × 3 = 3
Modify	6 × 2 = 12	3 × 2 = 6	3 × 2 = 6
Delete	N/A	N/A	N/A
Total CW	21	9	9

Table 8: Calculated change weights

$$IW_{CO} = \left(\sum_{X=1}^{n} N_{CO}\right) \times \sum CW_{CO}$$

where CO is the Change Option number and N_{CO} is the number of interactions per change action where only interactions of the DAAs are considered. We reiterate the reason for only considering DAAs is they are directly attached to the change actions (as seen in Table 4) and IdAAs are not. The number of interactions for the DAAs was identified when calculating the IC value. CW_{CO} is the total change weight for that option as shown in Table 8.

Applying the equation to the running example:

For option 1:

The directly affected activities are A_4 and A_5. Therefore,

$N_1 = 2+1$

$CW_1 = 21$

$IW_1 = (2 + 1) \times 21 = 63$

For option 2:

The directly affected activities are A_4 and A_5. Therefore,

$N_2 = 2+1$

$CW_2 = 9$

$IW_2 = (2 + 1) \times 9 = 27$

For option 3:

The directly affected activities are A_5 and A_6. Therefore,

$N_3 = 1+1$

$CW_3 = 9$

$IW_3 = (1 + 1) \times 9 = 18$

4.3.3 Rework calculation based on IC and IW

In section 4.3.1, IC was established to be the percentage of interactions that need to be altered in order to facilitate the required change and in section 4.3.2, IW was established to be the change weight corresponding to the directly affected interactions due to the requirements change. Based on these two values, the assessment of rework is a combined look at both the interactions that need to be altered in comparison to the full system depicted in the SDD and the difficulty of implementing the change action on those interactions. In order to display the comparison between the rework required for the changes requested and their multiple options, we use Table 9 as a template.

	Change 1			Change 2	Change n
	Opt 1	Opt 2	Opt n		
IC					
IW					

Table 9: Template of comparison between rework

To better understand this template, we populate it with the outcome of the running example:

	Change 1		
	Opt 1	Opt 2	Opt 3
IC	67%	67%	33%
IW	63	27	18

Table 10: Outcome of comparison

According to this example, one change was requested with three possible actions that can be taken to implement it. According to the above table, the value of IC is the same for options 1 and 2. Option 3 has a lower IC value than that of options 1 and 2. This is a good indication that option 3 is the lesser invasive option for implementing the change as a lesser number of interactions has to be altered. This fact is further validated by the IW value where option 3 has the lowest IW value corresponding to a lower difficulty level of implementing the change.

Based on the above results, it can be said that:

- option 1 and 2 require 67% of the interactions to be altered while option 3 requires only 33% alterations;
- based on IW, option 3 has a lesser difficulty level of implementation as compared to the other options; and
- therefore, the lesser invasive change implementation is option 3, based on both the IC and IW values.

5. A case study

The usefulness of our method can be illustrated by applying it to a software project case study. Figure 3 represents a partial system design diagram of a course management system adopted from [142]. The same case study has been used in the specification and classification methods as well as the change analysis method [76, 254, 255]. The use of the same case study provides a holistic view as to how all these methods can be used to manage requirements changes. This is a typical real-life system which we work on as academics at a University. The diagram illustrates the relationships and some dependencies the activities have with each other. The relationships denoted in the diagram can be defined as follows:

- Requires (Req): An activity A_1 requires an activity A_2 if A_1 is fulfilled only when A_2 is fulfilled. A_2 can be treated as a pre-condition for A_1 [142].
- Refines (Ref): An activity A_1 refines an activity A_2 if A_2 is derived from A_1 by adding more details to it [142].
- Contains (Con): An activity A_1 contains information from A_2...A_n if A_1 is the conjunction of the contained information from A_2...A_n [142].

The detailed purpose of each activity in the diagram is described as follows:

1) The system allows end-users to provide profile and context information for registration.
2) The system provides functionality to search for other people registered in the system.
3) The system provides functionality to allow end-users to log into the system with their password.
4) The system supports three types of end-users (administrator, lecturer and student).
5) The system allows lecturers to set an alert on an event.
6) The system maintains a list of events about which the students can be notified.
7) The system notifies the students about the occurrence of an event as soon as the event occurs.
8) The system actively monitors all events.
9) The system notifies students about the events in the lectures in which they are enrolled.
10) The system allows students to enroll in lectures.
11) The system allows lecturers to send e-mail to students enrolled in the lecture given by that lecturer.
12) The system allows students to be assigned to teams for each lecture.
13) The system allows lecturers to send e-mail to students in the same group.
14) The system allows lecturers to modify the content of the lectures.
15) The system gives different access rights to different types of end-users.
16) The system supports two types of end-users (lecturer and student) and it provides functionality to allow end-users to log into the system with their password.

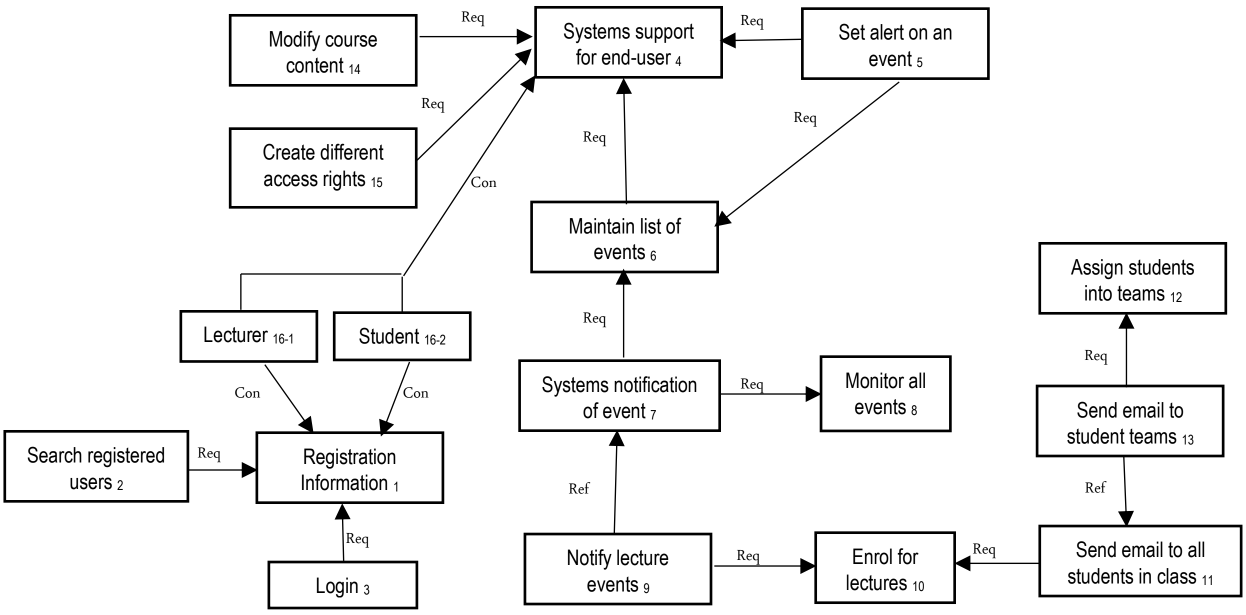

Modify course content 14
Create different access rights 15
Systems support for end-user 4
Set alert on an event 5
Maintain list of events 6
Assign students into teams 12
Lecturer 16-1
Student 16-2
Systems notification of event 7
Monitor all events 8
Send email to student teams 13
Search registered users 2
Registration Information 1
Login 3
Notify lecture events 9
Enrol for lectures 10
Send email to all students in class 11
Req
Con
Ref

Figure 3: Partial system design diagram of a course management system

The example consists of two scenarios. These scenarios are based on our observations as university academics who use similar course management systems. The following hypothetical new requirements are identified:

1) In an emergency, it would be more effective to send an SMS notification to students as well as an email.
2) An academic who is part of the project suggests that it would be beneficial to send an alert of the event to the students in the class before the event occurs.

5.1 Identification of the changes – Step 1

Applying the change identification of the rework method yields the following specifications and classification for the aforementioned changes.

Change 01	**Option 01**	**Option 02**
Object	A_{10}	A_{11}
Purpose	Functionality enhancement	Functionality enhancement
Focus	Add	Modify
Additional Question	Need additional Input / Output? Y	Input/output modification? Y
Result		

Change Type	Add new function	Inner property + Output interface modification
Action	Add new function by using information from A_{10}	Modify A_{11} internally and the output interface

Table 11: Identification of Change 01

Change 02	**Option 01**	**Option 02**
Object	A_5 & A_7	A_7
Purpose	New requirement	New requirement
Focus	Add & Modify	Modify
Additional Question	Need additional Input / Output? Y Input/output modification? Y	Input/output modification? Y
Result		
Change Type	Add new link Inner property + Output interface modification	Inner property + Output interface modification
Action	Link A_5 with A_7 by modifying A_5 internally and externally. Then modify A_7 internally and externally to send alert	Modify A_7 internally and externally to send alert before event

Table 12: Identification of Change 02

5.2 Identification of the system activities affected by the change – Step 2

The outcome of Tables 11 and 12 is then expanded using the change analysis functions (Step 2) as follows:

Change action 1 – Option 1:

Add new activity (SendSMS) Function (with mismatched interfaces)

```
1. CreateFunc(String, V) →SendSMS
2. CreateLink(SendSMS, A10, V)
   {
       3. ModifyInner(SendSMS,V)
       4. ModifyIn(SendSMS, A10, V)
       5. ModifyOut(A10,Null,V)
   }
```

Change action 1 – Option 2:

1. ModifyInner(A_{11},V)
2. ModifyOut(A_{11},Null,V)

Change action 2 – Option 1:

Modify A_5 and link with A_7

1. ModifyInner(A_5, V)
2. CreateLink(A_5, A_7, V)
3. ModifyInner(A_7, V)
4. ModifyOut(A_7,Null,V)

Change action 2 – Option 2:

Modify A_5 and link with A_{11}

1. ModifyInner(A_7, Null)
2. ModifyOut(A_7, Null, V)

In order to assess the rework for the above changes, the activities affected by the change need to be identified. Based on the above functions, the affected activities for each change and option are as follows:

Change 1:

Option 1: A_{10} (direct); A_9 and A_{11} (indirect)

Option 2: A_{11} (direct); A_{10} and A_{13} (indirect)

Change 2:

Option 1: A_5 and A_7 (direct); A_4, A_6, A_8 and A_9 (indirect)

Option 2: A_7 (direct); A_6 and A_9 (indirect)

5.3 Assessing the rework required – Step 3

Based on the outcome of step 2, IC and IW needs to be calculated before the rework can be assessed. In order to calculate IC, the number of interactions of both directly and indirectly affected activities needs to be identified.

5.3.1 IC calculation

Using the outcome of step 2, the interactions and the calculation of IC for both changes are as follows:

Change 1:

Following are the number of interactions for each activity based on the system diagram:

A_9 – has 2 connections

A_{10} – has 2 connections

A_{11} – has 2 connections

A_{13} – has 2 connections

Considering all the interactions, the entire system design contains 16 activities. The interaction count for each activity is as follows:

A_1 – 4, A_2 – 1, A_3 – 1, A_4 – 5, A_5 – 2, A_6 – 3, A_7 – 3, A_8 – 1, A_9 – 2, A_{10} – 2, A_{11} – 2, A_{12} – 1, A_{13} – 2, A_{14} – 1, A_{15} – 1, and A_{16} – 2

Calculating IC for ***option 1*** of **change 1**

$$IC_{1\ of\ 1} = \frac{N_I}{N_{TI}}$$

$$N_I = \sum_{x=1}^{n} N_{Ix} = Connections\ (\text{A10} + \text{A9} + \text{A11}) = 2+2+2 = 6$$

$$N_{TI} = \sum_{x=1}^{n} N_{TIx} = All\ Connections$$
$$= 4+1+1+5+2+3+3+1+2+2+2+1+2+1+1+2 = 33$$

$$\boldsymbol{IC_{1\ of\ 1}} = \frac{\mathbf{6}}{\mathbf{33}} = \mathbf{18}\%$$

Calculating IC for ***option 2*** of **change 1**

$$IC_{2\ of\ 1} = \frac{N_I}{N_{TI}}$$

$$N_I = \sum_{x=1}^{n} N_{Ix} = Connections\ (\text{A11} + \text{A10} + \text{A13}) = 2+2+2 = 6$$

$$\boldsymbol{IC_{2\ of\ 1}} = \frac{\mathbf{6}}{\mathbf{33}} = \mathbf{18}\%$$

Change 2:

Following are the number of interactions for each of the above activities based on the system diagram:

A_4 – has 5 connections

A_5 – has 2 connections

A_6 – has 3 connections

A_7 – has 3 connections

A_8 – has 1 connection

A_9 – has 2 connections

Calculating IC for ***option 1*** of **change 2**

$$IC_{1\,of\,2} = \frac{N_I}{N_{TI}}$$

$$N_I = \sum_{x=1}^{n} N_{Ix} = Connections\ (\text{A5} + \text{A7} + \text{A4} + \text{A6} + \text{A8} + \text{A9}) = 2+3+5+3+1+2 = 16$$

$$\boldsymbol{IC_{I\,of\,2}} = \frac{\mathbf{16}}{\mathbf{33}} = \mathbf{48.5}\%$$

Calculating IC for ***option 2*** of **change 1**

$$IC_{2\,of\,1} = \frac{N_I}{N_{TI}}$$

$$N_I = \sum_{x=1}^{n} N_{Ix} = Connections\ (\text{A7} + \text{A6} + \text{A9}) = 3+3+2 = 8$$

$$\boldsymbol{IC_{2\,of\,2}} = \frac{\mathbf{8}}{\mathbf{33}} = \mathbf{24.2}\%$$

5.3.2 IW calculation

Using the outcome of step 2, we first identify the change types for each change. The results are as follows:

Change 1		**Change 2**	
Opt 1	**Opt 2**	**Opt 1**	**Opt 2**
Create functions – 2 Modify functions – 3 Delete functions – 0	Create functions – 0 Modify functions – 2 Delete functions – 0	Create functions – 1 Modify functions – 3 Delete functions – 0	Create functions – 0 Modify functions – 2 Delete functions – 0

Using the weight system of the change analysis method given in section 5.2, we obtain the following table:

	Change 1		**Change 2**	
Change type	**Opt 1**	**Opt 2**	**Opt 1**	**Opt 2**
Add	2 × 3 = 6	N/A	1 × 3 = 3	N/A

Modify	3 × 2 = 6	2 × 2 = 4	3 × 2 = 6	2 × 2 = 4
Delete	N/A	N/A	N/A	N/A
Total CW	12	4	9	4

Table 13: Change Weight calculation for Change 1 and 2

Using the outcome of Table 13, we calculate the IW for Change 1 and 2 as follows:

Considering only the DAAs: A_{10} = 2 interactions

$$\boldsymbol{IW_{I\ of\ 1} = (2) \times 12 = 24}$$

Option 2** of **Change 1:

$$IW_{2\ of\ 1} = \left(\sum_{X=1}^{n} N_{2\ of\ 1}\right) \times \sum CW_{2\ of\ 1}$$

Considering only the DAAs: A_{11} = 2 interactions

$$\boldsymbol{IW_{2\ of\ 1} = (2) \times 4 = 8}$$

Option 1** of **Change 2:

$$IW_{1\ of\ 2} = \left(\sum_{X=1}^{n} N_{1\ of\ 2}\right) \times \sum CW_{1\ of\ 2}$$

Considering only the DAAs: A_5 and A_7 = 2 and 3 interactions respectively

$$\boldsymbol{IW_{1\ of\ 2} = (2 + 3) \times 9 = 45}$$

Option 2** of **Change 2:

$$IW_{2\ of\ 2} = \left(\sum_{X=1}^{n} N_{2\ of\ 2}\right) \times \sum CW_{2\ of\ 2}$$

Considering only the DAAs: A_7 = 3 interactions

$$\boldsymbol{IW_{2\ of\ 2} = (3) \times 4 = 12}$$

5.4 Rework calculation

The rework calculation results for the changes are as follows:

	Change 1		**Change 2**	
	Opt 1	**Opt 2**	**Opt 1**	**Opt 2**
IC	18%	18%	48.5%	24.2%
IW	24	8	45	12

Table 14: Rework assessment for changes 1 and 2

Based on the outcomes, the following section provides an explanation of the use of the results of the method in assessing the rework and identifying the implementation option for rework minimization. The above values provide a numeric view for developers for selecting the implementation option which is lesser invasive to the software design of the system. These values do not represent cost and effort estimates. A method for estimating them would be another piece of research work within the area of RCM and is beyond the scope of this paper.

Interpretation of Table 14:

- Change 01 can be implemented using two options, where both options have the same IC of 18%. This indicates that in order to implement change 01 using either one of the options, 18% of the interactions of the software design of the system need to be altered.
- Given that both options of change 01 have the same IC percentage, the value of IW needs to be considered in establishing which option requires a lesser amount of rework.
- Based on the IW calculation for the two options, option 2 has a lower value, which translates into lower difficulty of implementation and therefore, lesser rework in comparison to option 1.
- Based on the IC, of the two options of Change 02, it is an indication that option 2 will alter fewer interconnections compared to option 1.
- This is further clarified by the fact that IW is less for option 2, which denotes that it has a lesser difficulty level of implementation and would require lesser rework.

Based on the above results, the following observations can be made:

- Between the two options for the implementation of change 01, option 2 should be chosen as it is lesser invasive to the software design of the system and requiring lesser rework, with 18% of the interactions to be altered.
- Between the two options for the implementation of change 02, option 2 should be chosen as it is lesser invasive to the software design of the system and requiring lesser rework, with 24% of the interactions to be altered.
- When comparing the two chosen change implementation options, change 02 will require more rework as its IC shows a higher number of interactions to be altered than change 01 as well as its higher IW denotes a higher difficulty level of implementation.

6. Related work

To the best of our knowledge, in the literature there has been no paper published on assessment of rework in the area of RCM. However, we are able to find two papers in the literature which focus on effort estimation related to implementation of requirements changes. Although these methods do not assess rework, they use requirements changes and their impact in the calculation process in a similar manner to our method. We shall discuss below a comparison with these two pieces of work.

Requirements changes can occur at any phase of the development process and even after deployment. There are few estimation methods dedicated to change effort/cost estimation and the importance of such methods were established in the introduction. The following discussion elaborates on two methods that deal specifically with change effort/cost estimation that use a similar rationale to the method introduced in this paper.

The estimation method introduced by Jeziorek [258] attempts to estimate the cost of the impact of a design change to development. The author emphasises the importance of identifying the functional requirements and design parameters that are impacted by the change, before attempting to estimate the cost of change. He uses this identification in the form of a matrix to detect the physical interactions between components. These physical interactions are used to determine how the change propagates through the system. The model developed in [258] outputs the affected components, how they are affected and what the cost of impact will be. In this particular method, the use of interactions between components and the mapping of the propagation of the change through the system are similar activities as used in our method.

In the method established by Lavazza and Valetto [259], several different artifacts are used to calculate the change costs. The key feature of this method is the use of requirements instead of lines of code to calculate the cost. Therefore, the method utilizes the design document and traceability techniques for estimation. The estimation is carried out in two steps: 1) characteristics such as the size and the complexity of the code are estimated on the basis of the size of the complexity of the requirements and the skill and experience of the implementation team; 2) effort is estimated based on the knowledge of the relations that link the inputs, outputs and the resources required. Most parts of the estimation are based on historic data. The use of requirements to establish the complexity and the linking of inputs and outputs resonate with the rework method introduced in this paper.

We use the aforementioned work to describe the limitations of the existing work and compare our methods to define what has been achieved. The limitations focus only on the techniques comparable with our method.

Technique	**Limitations**	**What our method addresses**
Jeziorek [258]	Initially, a lot of time needs to be spent in developing the matrices needed to identify the impact. These matrices are non-transferable and therefore for every project, new matrices need to be established.	New diagrams are not needed. The method uses the system diagram which a software project would usually have.
Lavazza and Valetto [259]	The use of historical data which may not be available for some projects and is therefore limited to systems development that has such data. The use of traceability methods that have inherent limitations such as informal development methods, insufficient resources, time and cost for traceability, lack of coordination between people responsible for different traceable artifacts, imbalance between benefits obtained and effort spent implementing	The method uses data only from the current project. The change identification and analysis techniques used in this method do not use traceability techniques and therefore do not have the drawbacks associated with traceability techniques.

	traceability practices, and construction and maintenance of a traceability scheme proves to be costly [35, 124, 131, 133, 219-222]	

Table 15: Comparison with related work

7. Conclusions and future work

In this paper, we have presented a definition of rework – "*Rework in the field of software engineering is an activity within the area of Requirements Change Management (RCM), which makes modifications/alternations to a system which has a software design document and is being developed (pre-delivery) for implementing certain requirements changes, with the alternations/modifications normally introducing extra work and increasing the total amount of cost/effort for completing the software project; and assessing rework, a preliminary step to change cost/effort estimation which is another RCM activity, is about studying how a system needs to be modified/altered for implementing the changes.*" We have also described a method of assessing rework for implementing software requirements changes. Once a change has been proposed, our method identifies the paths of implementation, which lead to the identification of the impacted activities of the system through the SDD. Using these activities, two values (IC and IW) are computed to help assess the rework required for all the possible options. Based on the IC and IW values, a developer can choose the lesser invasive option which requires lesser rework.

To demonstrate the practicality of our method, we have applied it to a course management system. For the two requested requirements changes, we generated multiple implementation options and for each option, IC and IW were calculated. Based on the course management case study and the running example, we demonstrated that when multiple options of implementation exist for one change, IC alone is not sufficient to make an assessment and selection. In both applications, a change resulted in IC to produce the same value for possible implementation. In such instances, IW plays an important role in the assessment process. Based on the values of IC and IW, rework was assessed and comparisons were then made between the implementation options of a change and we were able to identify which implementation option was the less invasive option which requires a lesser amount of rework.

The assessment of rework was also used for comparing the different changes in order to determine which change would require more rework.

The results of applying our method to this case study show that it is useful in the area of RCM as it enables developers to have a better understanding of the rework required and to be able to compare the rework between the different options available for implementing a change and to identify the one which is less invasive to the software system design. Given that the implementation path is extracted from the SDD, our method can be applied during any phase of the software development provided that its design document is available.

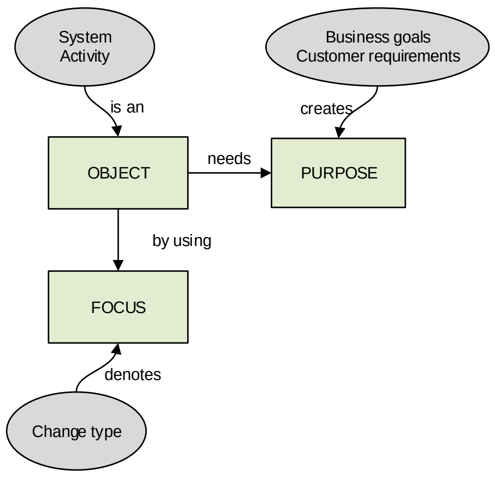
As our method is able to provide a better understanding of the additional work required and to identify the implementation with a lesser amount of rework, we can thus conclude that it can serve as a precursor to change effort estimation, whereby it is not necessary to carry out estimation for all the possible implementation options but the one which has been assessed to involve lesser rework. With the results derived from using our method, a directly related future work would be to develop a change effort estimation method for estimating the time and cost expected for implementing a change for the selected implementation option.

Appendix 1

The change specification method:

The specification method is made up of GQM and RDF. The GQM-RDF combination is a result of amalgamating ontology and terminology which in this paper, we refer to as onto-terminology. The method has both linguistic and logical principles. To ensure the correct combination of logic and terminology, we have selected two well-known methods where GQM represents terminology and the other RDF ontology. Three terms are extracted from GQM that can best describe a requirement change; Object, Purpose and Focus (of change). The terms extracted from RDF are Object, Attribute and Value, which is referred to as the RDF triplet. The logical relationship of the RDF triplet can be stated as Object O has an Attribute A with a Value V (Professor; Reads; a Book). The rationale behind the correspondence between RDF triplet and to the GQM terms is due to the similarity and the meanings of the terms, which is described in table below.

RDF term	GQM term	Correspondence	Rationale
Object	*Object*	One-to-one	Same concept
Attribute	*Purpose*	One-to-one	Both terms are activities. *Purpose* is an activity that is generated due to various business requirements.
Value	*Focus*	One-to-one	*Value* of RDF creates the significance for *Attribute* (of RDF). *Focus* of GQM creates the significance for *Object* (of GQM) by activating the term *Purpose* of GQM.



Onto-terminology Framework

The template designed for the change specification based on the framework above is given in the table below. By selecting the object of change using the system design diagram, designers and decision makers can accurately locate the main target of change, resulting in a clarification of the location of change. Knowing the reason for the change through the purpose ensures that change implementers are able to clarify the need for the change. The focus of change acts as advice on the basic implementation needed to execute the change, resulting in the clarification of the action of change. It indicates to the designers what to do instead of how to do the change. We believe that clearly describing the location, need and action of a change request using this template will resolve much of the existing miscommunication issues.

	Description
OBJECT	The activity name according to the system design diagram
PURPOSE	The reason for the change (can be descriptive)
FOCUS	Select from Add, Delete, Modify or Activity Relocation

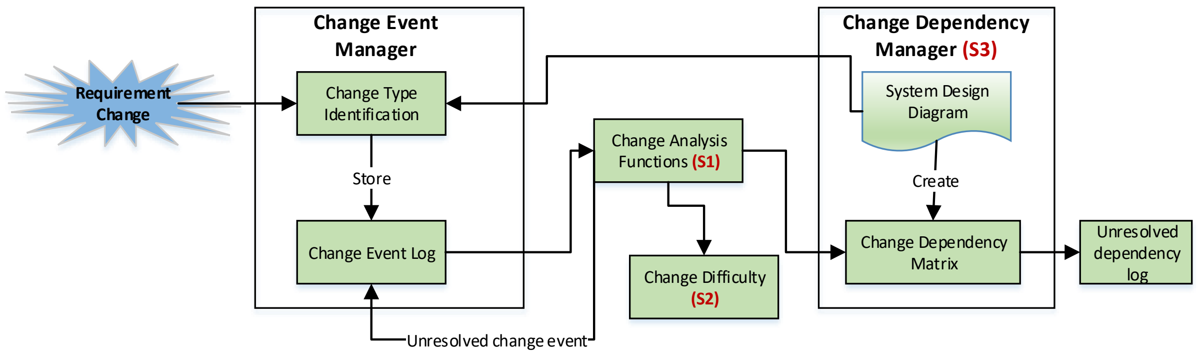
The change classification method:

The main purpose of change classification method is to ensure that change implementers are able to identify and understand unambiguously the requirement change. The classification is based on previous literature on the same and unstructured interviews of 15 practitioners in the field of change management. The result of this investigation is given in section 4.1 Table 1.

Appendix 2

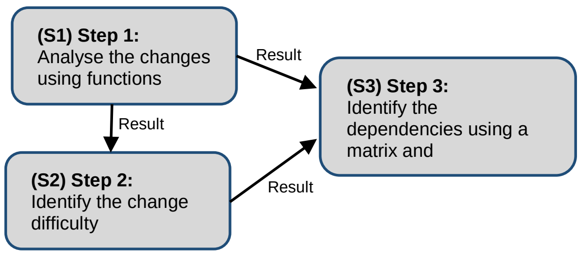
The Method of Requirements Change Analysis

The method consists of three steps: namely, (1) analyzing the change using functions, (2) identifying the change difficulty; and (3) identifying the dependencies using a matrix. We have used step 1 in the rework method introduced in this paper.



Change analysis method

Once a change has been identified through the Change Event Manager (CEM), the method follows three steps:


(S1) Step 1: Analyse the changes using functions
Result
(S3) Step 3: Identify the dependencies using a matrix and
Result
(S2) Step 2: Identify the change difficulty
Result

Three step analysis process

- Step 1 (S1) is for expanding the identified changes and for discovering the more detailed information for the implementation as a result of the changes. As shown in Figure 2, the two categories of change analysis functions (herein after referred to as functions) described in section 3.2 are employed for carrying out this step.
- Step 2 (S2) identifies the difficulty of implementing the change. The result of this will be used later for assigning a priority to each of the requested changes.
- Step 3 (S3) identifies the conflicts and/or dependencies between the required changes. As shown in in Figure 2, the key elements involved are the Change Dependency Matrix (CDM) and the System Design Diagram (SDD). The conflicts and/or dependencies between the changes are identified once the changes have been mapped to the matrix.

Chapter 6

Conclusions and Future work

This chapter concludes the work presented in this thesis. It discusses the research work conducted to achieve the aims and objectives of the thesis and provides a critical appraisal of the work done. In addition, this chapter also outlines the remaining challenges in the discipline and the potential areas of our work which have opened for future research.

6.1 Research work conducted

This research has achieved its goals with respect to the aims and objectives set out in chapter one. The following subsections summarise the work that was carried out to achieve these aims and objectives.

6.1.1 A systematic review of requirements change management

An extensive survey of the literature was conducted in this research area, and the findings created a clear path for the rest of the research work carried out in this thesis. The amount of research carried out in the area was quite extensive and was in a variety of directions. These directions were categorised using four research questions: RQ1: causes of requirements change; RQ2: processes required to manage requirements changes; RQ3: techniques used in RCM; and RQ4: decision making when dealing with RCs. All four research questions were investigated from the perspective of both traditional and agile development.

Based on the findings of RQ1, it was clear that requirements changes originate from several different sources, which include the customer organization, the development organization and the external environment. These causes of change were categorised into five areas based on their origin: external market, customer organization, project vision, requirements specification and solution. Based on the findings of RQ2, there are two main categories suggested in the literature for managing change: semi-formal processes and formal processes. There are three key areas that are common to both types of processes: change identification, change analysis and change cost/effort estimation. The identification of these three areas formed the basis of RQ3.

In RQ3, the techniques of RCM were discussed in detail within the categories mentioned above. According to the literature, change identification methods did not have much agreement on how to identify requirements changes. In most of the above studies, two techniques were prominent: taxonomies and classification. One of the main activities of change analysis is to analyse the impact of change on the existing system or design. According to the literature, traceability techniques are the most common way to identify the impact of change. There were a limited number of methods dedicated for change cost/effort estimation. In the methods that deal with RCM, change cost estimation was executed using existing costing techniques such as COCOMO, expert judgement, etc.

The literature relevant to RQ4 makes it clear that an organization can be divided into two parts: the business organization and the IT organization and that furthermore, there are differences in the decisions made concerning RCM in the two different parts of an organisation and there are also two differing viewpoints, these being of the developer and the manager.

6.1.2 Managing requirements changes through change specification and classification

As a result of the inadequacy of current knowledge in relation to in change identification, as revealed by our systematic review, we presented the methods of change specification and classification in chapter 3. Using these methods, business and IT staff are able to communicate and identify requirements changes using a set of common terminology.

The change specification method provides a way to state the change so that the business part of the organisation and the IT part of the organisation can avoid communication ambiguities. The specification method is able to identify the key activities of the system design where the change needs to be implemented, the reason for the change request and the basic focus of the change. The outcome of the classification method is to elaborate further on the change focus and indicate in more detail what type of change is required as well as provide initial guidelines on how to implement the change. To assist in this process, we were able to identify four main types of changes; add, delete, modify and relocate.

6.1.3 A method of requirements change analysis

Within the area of change analysis, change impact analysis is a popular topic due to its level of importance. In order to analyse this impact, it is essential that the propagation of a change can be traced within the existing system. This allows the development team to understand how the change "travels" within the system and to identify the activities are affected as a result. Based on the available methods, the most common technique used to identify the change propagation path is traceability methods. However, we have uncovered several drawbacks with traceability based approaches.

Based on these findings, we introduced the method of requirements change analysis. This method provides a way to identify the propagation of a requirements change through alternate techniques that do not use traceability. Based on this analysis, we are able to identify the system activities that are affected both directly and indirectly by the change. We use the change analysis functions introduced in this method to further expand the changes, which provides the development team with more detailed implementation directions. We are also able to comment on how difficult the implementation of a change is, identify conflicts and/or dependencies between changes and based on the difficulty and dependency level, assign a propriety to the changes.

6.1.4 A method of assessing rework for implementing software requirements changes

It is possible that changes can be implemented in more than one way. As a result the rework required for each of these options will vary. In such scenarios, it would be beneficial to assess the rework for each implementation option and identify which option requires lesser rework prior to change effort estimation.

The research work presented in chapter 5 contains two key objectives: to define rework in the context of RCM and to present a method of assessing rework for all possible implementation options of a RC. The main outcome of the method is to compare the rework between the multiple options of implementing one change and between different changes. As a result, developers are able to choose the implementation option with lesser rework. Because one option is selected, change effort estimation needs only to be calculated for this option and therefore, the rework method acts as a precursor to change effort estimation.

6.2 Evaluation of the research work conducted

In this section, a critical appraisal of the research work conducted is presented based on the experience gained and results obtained during the research.

6.2.1 A systematic review of requirements change management

Strengths

Research on RCM has a very rich history with a plethora of work carried out in various directions. One of the key strengths of this review is that it is the first of its kind. We have not been able to find any other systematic review that brings together the vast amount of knowledge on RCM into one location. This is further enhanced by the fact that we identify how RCM varies between traditional and agile development. Another benefit of the review is that it forms a guide for all parties interested in dealing with RCM to be able to clearly understand the research space of RCM and to choose the state-of-the-art techniques to manage change based on their needs. This is further strengthened by the critical analysis carried out on various techniques used in RCM, describing their strengths, limitations and their potential for improvement. Another merit of this work is that it provides directions for future work in the form of research gaps, which is very beneficial for those interested in improving the techniques in RCM.

Limitations

The main limitation is due to the set of specific keywords used for data collection. As a result, the findings of this review could not be generalized. The findings of this review are limited to the key words and the six research repositories used. Although we took several precautions, there could be minor variations to the findings if carried out by another researcher based on their personal apptitude and thinking.

6.2.2 Managing requirements changes through change specification and classification

Strengths

One of the key contributors to the difficulty of managing RCs is the ambiguity in the communication between business and IT staff. To address this issue, this method provides a semi-formal medium to promote the mutual understanding of RCs between stakeholders, which is its main strength. The methods introduced promote the mutual understanding of RCs between business and IT staff. Another merit is the initial identification of what activities might be impacted by the change and through the multiple possibilities generated, developers will have a general sense of which direction to proceed in, in terms of executing the change.

Limitations

The methods introduced require human intervention, hence they draw on the experience and the expertise of the individuals involved in the process. Therefore, the possibility of inconsistencies based on experience is to be expected. Because this method was evaluated using a case study, the findings reflect a typical situation and cannot be generalised to every possible scenario or type of development.

6.2.3 A method of requirements change analysis

Strengths

Analysing a change and understanding its impact on the existing system is an imperative activity in RCM. One of the challenges presented by the existing work is the need to use traceability techniques to map the impact of change on the system design and these techniques came with a few inherent drawbacks. The core strength of this method is that traceability techniques are replaced by the changes themselves. Instead of using requirements to trace the impact of change, the RCs are used to work out the connections and conflicts that may arise in the system if the changes are implemented. This means that by using the method, we were able to address the key drawbacks of traceability techniques. The ability to allocate priority to changes and to assess the difficulty of changes were secondary strengths that assist in making decisions relating to the suitability of carrying out a change.

Limitations

We have postulated that the change analysis functions are based on the most commonly used change types and we have anticipated, to the best of our knowledge, the possible variations that may arise from these common types. However, the list is not exhaustive and therefore there may be certain scenarios of changes that might be outside the range of the functions described in this thesis. The application of the method is evaluated through a case study that limits its applicability in all possible development techniques, in which case, we recognize the importance of conducting experiments in an industrial environment.

6.2.4 A method of assessing rework for implementing software requirements changes

Strengths

One of the main strengths of the work presented in chapter 5 is to create clarity regarding the term *rework* by defining it in the context of RCM. Based on this definition, the method allows for the assessment of the rework required to implement a change without complex calculations and consequently, it is able to address the limitations of the existing methods. The versatility of this method enables the assessment of the rework to be carried out at any stage of the development process, not just at the beginning. One of the key benefits is that the method is able to compare rework between multiple possibilities of a single change and between multiple changes. It is also a precursor to change cost estimation as the identification of the implementation option with minimal rework reduces the number of times the estimation has to be carried out. The method is not dependent on the development technique and therefore can be applied to both traditional and agile development techniques.

Limitations

The rework assessed is an initial look at how much the system needs to be changed to accommodate the change. It does not calculate the cost or time required to implement the change. The method is ideally suited to collocated development and would need some adjustments to be considered for global software development projects.

6.3 Future work

In this section, the remaining challenges in the discipline and the possible dimensions of work that have opened up for future investigation are discussed.

6.3.1 Change cost/effort estimation

The systematic review revealed a lack of methods dedicated to change estimation. The literature also bears evidence that the cost and time related to implementing a change can be a significant factor affecting the budget as well as the successful completion of a project. The rework method introduced in chapter 5 is a precursor to change cost/effort estimation. However, we have not established a direct conversion of the assessment rework to represent change cost/effort estimation. It would therefore be beneficial to extend the rework method to be able to estimate the cost and effort of implementing a change.

6.3.2 Requirements change validation

Requirements change validation was included in some of the RCM processes identified in the literature review. The main intension of change validation is to ensure that the existing system is stable and functions as required after the implementation of the change. However, we believe that change validation should not be executed as testing after actual implementation but through mock implementation to avoid additional defect fixing. It wasn't evident from the literature found that much work has been done in the area of change validation. It would be ideal for change validation to occur after impact analysis and prior to change cost estimation, the rationale being that any change that cannot be validated can be rejected before any estimation is carried out.

6.3.3 Applying the methods in an industry case study

The findings presented in the thesis on all the methods are based on a case study related to academia and have the limitation of not being applicable to all possible development scenarios. We aim to apply the methods to a broader case study in industry that will test the merits of each method in a broader spectrum.

References

[1] A. Kobayashi and M. Maekawa, "Need-based requirements change management," in *Engineering of Computer Based Systems, 2001. ECBS 2001. Proceedings. Eighth Annual IEEE International Conference and Workshop on the*, 2001, pp. 171-178: IEEE.

[2] K. El Emam, D. Höltje, and N. H. Madhavji, "Causal analysis of the requirements change process for a large system," in *Software Maintenance, 1997. Proceedings, International Conference*, 1997, pp. 214-221: IEEE.

[3] I. Sommerville, "Software Engineering. International computer science series," *ed: Addison Wesley*, 2004.

[4] H. M. Sneed, "A cost model for software maintenance & evolution," in *Software Maintenance, 2004. Proceedings, 20th IEEE International Conference on*, 2004, pp. 264-273.

[5] B. W. Boehm, "Software engineering economics," in *Pioneers and Their Contributions to Software Engineering*: Springer, 2001, pp. 99-150.

[6] F. Brooks Jr, "No Silver Bullet Essence and Accidents of Software Engineering," *Computer*, no. 4, pp. 10-19, 1987.

[7] J. D. Procaccino, J. M. Verner, S. P. Overmyer, and M. E. Darter, "Case study: factors for early prediction of software development success," *Information and Software Technology*, vol. 44, no. 1, pp. 53-62, 2002.

[8] J. Dominguez, "The curious case of the chaos report 2009," *Project Smart*, 2009.

[9] I. Sommerville and G. Kotonya, *Requirements engineering: processes and techniques*. John Wiley & Sons, Inc., 1998.

[10] S. D. Harker, K. D. Eason, and J. E. Dobson, "The change and evolution of requirements as a challenge to the practice of software engineering," in *Requirements Engineering, 1993., Proceedings of IEEE International Symposium*, 1993, pp. 266-272: IEEE.

[11] S. Bohner, "Impact analysis in the software change process: A year 2000 perspective," in *Software Maintenance 1996, Proceedings, International Conference*, 1996, pp. 42-51: IEEE.

[12] S. Nurcan, J. Barrios, G. Grosz, and C. Rolland, "Change process modelling using the EKD-Change Management Method," in *European Conference on Information Systems*, 1999, pp. 513-529.

[13] R. Chitchyan, A. Rashid, P. Rayson, and R. Waters, "Semantics-based composition for aspect-oriented requirements engineering," in *Proceedings of the 6th international conference on Aspect-oriented software development*, 2007, pp. 36-48: ACM.

[14] M. R. Basirati, H. Femmer, S. Eder, M. Fritzsche, and A. Widera, "Understanding Changes in Use Cases: A Case Study," in *Requirements Engineering, 2015., Proceedings of IEEE International Symposium on*, 2015, pp. 352-361.

[15] J. Buckley, T. Mens, M. Zenger, A. Rashid, and G. Kniesel, "Towards a taxonomy of software change," *Journal of Software Maintenance and Evolution: Research and Practice,* vol. 17, no. 5, pp. 309-332, 2005.

[16] S. McGee and D. Greer, "A software requirements change source taxonomy," in *Software Engineering Advances, 2009. ICSEA'09. Fourth International Conference*, 2009, pp. 51-58: IEEE.

[17] S. Ghosh, S. Ramaswamy, and R. P. Jetley, "Towards requirements change decision support," in *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*, 2013, vol. 1, pp. 148-155: IEEE.

[18] L. C. Briand, Y. Labiche, and L. Sullivan, "Impact analysis and change management of UML models," in *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference*, 2003, pp. 256-265: IEEE.

[19] N. Nurmuliani, D. Zowghi, and S. Fowell, "Analysis of Requirements Volatility during Software Development Life Cycle," in *Australian Software Engineering Conference*, 2004, p. 28.

[20] N. Nurmuliani, D. Zowghi, and S. P. Williams, "Using card sorting technique to classify requirements change," in *Requirements Engineering Conference*, 2004, pp. 240-248.

[21] X. Hua, Q. Jin, and Z. Ying, "Supporting Change Impact Analysis for Service Oriented Business Applications," in *Systems Development in SOA Environments, 2007. SDSOA '07: ICSE Workshops 2007*, 2007, pp. 6-6.

[22] G. E. Stark, P. Oman, A. Skillicorn, and A. Ameele, "An examination of the effects of requirements changes on software maintenance releases," *Journal of Software Maintenance,* vol. 11, no. 5, pp. 293-309, 1999.

[23] C. Gupta, Y. Singh, and D. S. Chauhan, "A Dynamic Approach to Estimate Change Impact using Type of Change Propagation," *Journal of Information Processing,* vol. 6, no. 4, pp. 597-608, 2010.

[24] Å. Dahlstedt and A. Persson, "Requirements Interdependencies: State of the Art and Future Challenges," in *Engineering and Managing Software Requirements*, A. Aurum and C. Wohlin, Eds.: Springer Berlin Heidelberg, 2005, pp. 95-116.

[25] B. Regnell, B. Paech, A. Aurum, C. Wohlin, A. Dutoit, and Johan, "Requirements Mean Decisions! - Research issues for understanding and supporting decision-making in Requirements Engineering," *Proceedings, First Swedish Conference on Software Engineering Research and Practise*, 2001, pp.49-59.

[26] P. Carlshamre, K. Sandahl, M. Lindvall, B. Regnell, and J. Natt och Dag, "An industrial survey of requirements interdependencies in software product release planning," in *Requirements Engineering, 2001. Proceedings, Fifth IEEE International Symposium*, 2001, pp. 84-91.

[27] I. Sommerville and G. Kotonya, *Requirements Engineering: Processes and Techniques*. John Wiley \&amp; Sons, Inc., 1998, p. 282.

[28] K. Pohl, *Process-Centered Requirements Engineering*. John Wiley & Sons, Inc., 1996, p. 342.

[29] S. Ferreira, J. Collofello, D. Shunk, G. Mackulak, and P. Wolfe, "Utilization of process modeling and simulation in understanding the effects of requirements volatility in software development," in *International Workshop on Software Process Simulation and Modeling, Portland, Oregon*, 2003.

[30] D. Pfahl and K. Lebsanft, "Using simulation to analyse the impact of software requirement volatility on project performance," *Information and Software Technology,* vol. 42, no. 14, pp. 1001-1008, 2000.

[31] N. Nurmuliani, D. Zowghi, and S. Powell, "Analysis of requirements volatility during software development life cycle," in *Software Engineering Conference, 2004. Proceedings, 2004 Australian*, 2004, pp. 28-37: IEEE.

[32] N. Nurmuliani, D. Zowghi, and S. P. Williams, "Requirements volatility and its impact on change effort: Evidence-based research in software development projects," in *Proceedings of the Eleventh Australian Workshop on Requirements Engineering*, 2006.

[33] S. Ramzan and N. Ikram, "Making decision in requirement change management," in *2005 International Conference on Information and Communication Technologies*, 2005, pp. 309-312: IEEE.

[34] W. Lam and V. Shankararaman, "Requirements change: a dissection of management issues," in 25th *EUROMICRO Conference, 1999. Proceedings.* 1999, vol. 2, pp. 244-251: IEEE.

[35] M. Strens and R. Sugden, "Change analysis: a step towards meeting the challenge of changing requirements," in *Engineering of Computer-Based Systems, 1996. Proceedings, IEEE Symposium and Workshopn*, 1996, pp. 278-283: IEEE.

[36] J. Tomyim and A. Pohthong, "Requirements change management based on object-oriented software engineering with unified modeling language," in *Software Engineering and Service Science (ICSESS), 2016 7th IEEE International Conference*, 2016, pp. 7-10: IEEE.

[37] L. Lavazza and G. Valetto, "Enhancing requirements and change management through process modelling and measurement," in *Requirements engineering, 2000. Proceedings, 4th International Conference*, 2000, pp. 106-115: IEEE.

[38] B. Nuseibeh and S. Easterbrook, "Requirements engineering: a roadmap," in *Proceedings of the Conference on the Future of Software Engineering*, 2000, pp. 35-46: ACM.

[39] N. Ikram, "The management of risk in information systems development," Diss. 2000.

[40] B. R. Butler, P. K. Blair, A. J. Fox, I. A. Hall, K. N. Henry, J. A. McDermid, J. Parnaby, H. Sillem, and M. Rodd, "The challenges of complex IT projects," *Relatório técnico, Royal Academy of Engineering. British Computer Society,* 2004.

[41] B. Curtis, H. Krasner, and N. Iscoe, "A field study of the software design process for large systems," *Communications of the ACM,* vol. 31, no. 11, pp. 1268-1287, 1988.

[42] S. Ramzan and N. Ikram, "Requirement change management process models: Activities, artifacts and roles," in *2006 IEEE International Multitopic Conference*, 2006, pp. 219-223: IEEE.

[43] B. W. Boehm, "Understanding and controlling software costs," *Journal of Parametrics,* vol. 8, no. 1, pp. 32-68, 1988.

[44] D. Firesmith, "Common Requirements Problems, Their Negative Consequences, and the Industry Best Practices to Help Solve Them," *Journal of Object Technology,* vol. 6, no. 1, pp. 17-33, 2007.

[45] S. Lock and G. Kotonya, "An integrated framework for requirement change impact analysis," Proceedings, Fourth Australian Conference on Requirements Engineering, 1999, pp. 29–42.

[46] I. Sommerville and P. Sawyer, *Requirements engineering: a good practice guide*. John Wiley & Sons, Inc., 1997.

[47] A. Taylor, "IT projects: sink or swim," *The Computer Bulletin,* vol. 42, no. 1, pp. 24-26, 2000.

[48] E. Oz, "When professional standards are lax: the CONFIRM failure and its lessons," *Communications of the ACM,* vol. 37, no. 10, pp. 29-43, 1994.

[49] S. Lock and G. Kotonya, "Requirement level change management and impact analysis," Cooperative Systems Engineering Group, Technical Report Ref:CSEG/21/1998.

[50] B. Kitchenham, "Procedures for performing systematic reviews," *Keele, UK, Keele University,* vol. 33, no. 2004, pp. 1-26, 2004.

[51] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering–a systematic literature review," *Information and software technology,* vol. 51, no. 1, pp. 7-15, 2009.

[52] A. Ullah and R. Lai, "A systematic review of business and information technology alignment," *ACM Transactions on Management Information Systems (TMIS),* vol. 4, no. 1, p. 4-30, 2013.

[53] D. Wickramaarachchi and R. Lai, "Effort Estimation in Global Software Development - A systematic Review," *Computer Science and Information Systems*, vol. 14, no. 2, pp. 393-421, 2017.

[54] M. B. Miles and A. M. Huberman, *Qualitative data analysis: An expanded sourcebook*. sage, 1994.

[55] B. J. Williams, J. Carver, and R. B. Vaughn, "Change Risk Assessment: Understanding Risks Involved in Changing Software Requirements," in *Software Engineering Research and Practice*, 2006, pp. 966-971: Citeseer.

[56] S. McGee and D. Greer, "Software requirements change taxonomy: Evaluation by case study," in *Requirements Engineering Conference (RE), 2011 19th IEEE International*, 2011, pp. 25-34: IEEE.

[57] S. McGee and D. Greer, "Towards an understanding of the causes and effects of software requirements change: two case studies," *Requirements Engineering,* vol. 17, no. 2, pp. 133-155, 2012.

[58] B. Boehm, "Industrial software metrics top 10 list," *IEEE Software,* vol. 4, no. 5, 1987, pp. 84-85.

[59] S. L. Pfleeger, "Software metrics: progress after 25 years?," *IEEE Software,* vol. 25, no. 6, 2008, pp. 32-34.

[60] D. M. Weiss and V. R. Basili, "Evaluating software development by analysis of changes: Some data from the software engineering laboratory," *IEEE Transactions on Software Engineering,* no. 2, pp. 157-168, 1985.

[61] M. Bano, S. Imtiaz, N. Ikram, M. Niazi, and M. Usman, "Causes of requirement change-a systematic literature review," in *Evaluation & Assessment in Software Engineering (EASE 2012), 16th International Conference*, 2012, pp. 22-31: IET.

[62] A. Van Lamsweerde, *Requirements engineering: from system goals to UML models to software specifications*. Wiley Publishing, 2009.

[63] K. Wiegers and J. Beatty, *Software requirements*. Pearson Education, 2013.

[64] R. S. Pressman, *Software engineering: a practitioner's approach*. Palgrave Macmillan, 2005.

[65] C. Rolland, C. Salinesi, and A. Etien, "Eliciting gaps in requirements change," *Requirements Engineering,* vol. 9, no. 1, pp. 1-15, 2004.

[66] E. Fricke, B. Gebhard, H. Negele, and E. Igenbergs, "Coping with changes: causes, findings, and strategies," *Systems Engineering,* vol. 3, no. 4, pp. 169-179, 2000.

[67] A. M. Davis and K. V. Nori, "Requirements, Plato's Cave, and Perceptions of Reality," in *Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International*, 2007, vol. 2, pp. 487-492: IEEE.

[68] B. Boehm, "Requirements that handle IKIWISI, COTS, and rapid change," *Computer,* vol. 33, no. 7, pp. 99-102, 2000.

[69] M. G. Christel and K. C. Kang, "Issues in requirements elicitation," Carnegie Mellon University Technical report, CMU/SEI-92-TR-012, 1992.

[70] C. Ebert and J. De Man, "Requirements uncertainty: influencing factors and concrete improvements," in *Proceedings of the 27th international conference on Software engineering*, 2005, pp. 553-560: ACM.

[71] T. Moynihan, "How experienced project managers assess risk," *IEEE software,* vol. 14, no. 3, pp. 35-41, 1997.

[72] T. Moynihan, "Requirements-uncertainty': should it be a latent, aggregate or profile construct?," in *Software Engineering Conference, 2000. Proceedings, 2000 Australian*, 2000, pp. 181-188: IEEE.

[73] L. Mathiassen, T. Saarinen, T. Tuunanen, and M. Rossi, "Managing requirements engineering risks: an analysis and synthesis of the literature," *Helsinki School of Economics,* p. 63, 2004.

[74] C. Jones, "Strategies for managing requirements creep," *Computer,* vol. 29, no. 6, pp. 92-94, 1996.

[75] N. Nurmuliani, D. Zowghi, and S. P. Williams, "Using card sorting technique to classify requirements change," in *Requirements Engineering Conference, 2004. Proceedings. 12th IEEE International*, 2004, pp. 240-248: IEEE.

[76] S. Jayatilleke and R. Lai, "A Method of Specifying and Classifying Requirements Change," in *Software Engineering Conference (ASWEC), 2013 22nd Australian*, 2013, pp. 175-180: IEEE.

[77] J. Kramer and J. Magee, "The evolving philosophers problem: Dynamic change management," *IEEE Transactions on software engineering,* vol. 16, no. 11, pp. 1293-1306, 1990.

[78] J. S. O'Neal, "Analyzing the impact of changing software requirements: A traceability-based methodology," *Ph.D. dissertation of Louisiana State University,* 2003.

[79] S. Lock and G. Kotonya, "Tool support for requirement level change management and impact analysis," in *Doctoral Symposium Proceedings*, 1998: Citeseer.

[80] K. El Emam, D. Holtje, and N. H. Madhavji, "Causal analysis of the requirements change process for a large system," in *Software Maintenance, 1997. Proceedings, International Conference*, 1997, pp. 214-221: IEEE.

[81] D. Leffingwell and D. Widrig, *Managing software requirements: a unified approach*. Addison-Wesley Professional, 2003.

[82] G. Kotonya and I. Sommerville, *Requirements engineering: processes and techniques*. Wiley Publishing, 1998.

[83] D. Pandey, U. Suman, and A. K. Ramani, "An Effective Requirement Engineering Process Model for Software Development and Requirements Management," in *2010 International Conference on Advances in Recent Technologies in Communication and Computing*, 2010, pp. 287-291.

[84] W. Hussain, D. Zowghi, T. Clear, S. MacDonell, and K. Blincoe, "Managing Requirements Change the Informal Way: When Saying 'No'is Not an Option," in *Requirements Engineering Conference (RE), 2016 IEEE 24th International*, 2016, pp. 126-135: IEEE.

[85] D. M. Berry, K. Czarnecki, M. Antkiewicz, and M. AbdElRazik, "Requirements determination is unstoppable: an experience report," in *Requirements Engineering Conference (RE), 2010 18th IEEE International*, 2010, pp. 311-316: IEEE.

[86] M. Bommer, R. DeLaPorte, and J. Higgins, "Skunkworks approach to project management," *Journal of Management in Engineering*, vol. 18, no. 1, pp. 21-28, 2002.

[87] K. Skytte, "Engineering a small system," *IEEE Spectrum*, vol. 31, no. 3, pp. 63-65, 1994.

[88] B. Curtis, M. I. Kellner, and J. Over, "Process modeling," *Communications of the ACM*, vol. 35, no. 9, pp. 75-90, 1992.

[89] S. T. Acuña and X. Ferré, "Software Process Modelling," in *ISAS-SCI (1)*, 2001, pp. 237-242.

[90] J. Lonchamp, "A structured conceptual and terminological framework for software process engineering," in *Software Process, 1993. Continuous Software Process Improvement, Second International Conference*, 1993, pp. 41-53: IEEE.

[91] N. C. Olsen, "The software rush hour (software engineering)," *IEEE Software*, vol. 10, no. 5, pp. 29-37, 1993.

[92] M. Makarainen, "Software change management processes in the development of embedded software," *VTT PUBLICATIONS*, vol. 4, no. 1, p. 6, 2000.

[93] W. Lam, V. Shankararaman, S. Jones, J. Hewitt, and C. Britton, "Change analysis and management: a process model and its application within a commercial setting," in *Application-Specific Software Engineering Technology, 1998. ASSET-98. Proceedings, 1998 IEEE Workshop*, 1998, pp. 34-39: IEEE.

[94] S. A. Ajila, "Change management: Modeling software product lines evolution," in *Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics, Orlando, Florida*, 2002, pp. 492-497.

[95] S. A. Bohner, "Impact analysis in the software change process: a year 2000 perspective," in *International Conference on Software Maintenance*, 1996, vol. 96, pp. 42-51.

[96] A. Eberlein and J. Leite, "Agile requirements definition: A view from requirements engineering," in *Proceedings of the International Workshop on Time-Constrained Requirements Engineering (TCRE'02)*, 2002, pp. 4-8.

[97] L. Cao and B. Ramesh, "Agile requirements engineering practices: An empirical study," *IEEE Software*, vol. 25, no. 1, 2008, pp. 60-67.

[98] I. Inayat, S. S. Salim, S. Marczak, M. Daneva, and S. Shamshirband, "A systematic literature review on agile requirements engineering practices and challenges," *Computers in Human Behavior,* vol. 51, pp. 915-929, 2015.

[99] S. Bilgaiyan, S. Mishra, and M. Das, "A review of software cost estimation in agile software development using soft computing techniques," in *Computational Intelligence and Networks (CINE), 2016 2nd International Conference on*, 2016, pp. 112-117: IEEE.

[100] Y. Zhu, "Requirements engineering in an agile environment. Uppsala University J. Inayat et al," *Computers in Human Behavior,* vol. 30, no. 2014, p. 15, 2009.

[101] B. Ramesh, L. Cao, and R. Baskerville, "Agile requirements engineering practices and challenges: an empirical study," *Information Systems Journal,* vol. 20, no. 5, pp. 449-480, 2010.

[102] L. Jun, W. Qiuzhen, and G. Lin, "Application of agile requirement engineering in modest-sized information systems development," in *Software Engineering (WCSE), 2010 Second World Congress*, 2010, vol. 2, pp. 207-210: IEEE.

[103] M. Daneva *et al.*, "Agile requirements prioritization in large-scale outsourced system projects: An empirical study," *Journal of Systems and Software,* vol. 86, no. 5, pp. 1333-1353, 2013.

[104] A. De Lucia and A. Qusef, "Requirements engineering in agile software development," *Journal of Emerging Technologies in Web Intelligence,* vol. 2, no. 3, pp. 212-220, 2010.

[105] N. A. Ernst, A. Borgida, I. J. Jureta, and J. Mylopoulos, "Agile requirements engineering via paraconsistent reasoning," *Information Systems,* vol. 43, pp. 100-116, 2014.

[106] K. Boness and R. Harrison, "Goal sketching: Towards agile requirements engineering," in *Software Engineering Advances, 2007. ICSEA 2007. International Conference on*, 2007, pp. 1-6: IEEE.

[107] D. Carlson and P. Matuzic, "Practical agile requirements engineering," in *Proceedings of the 13th Annual Systems Engineering Conference*, 2010.

[108] D. M. Berry, "The inevitable pain of software development, including of extreme programming, caused by requirements volatility," *Eberlein and Leite,* 2002, pp. 1-11.

[109] M. Fowler, "Refactoring: Improving the Design of Existing Code. 2000," *DOI= http://www. martinfowler. com/books. html/refactoring,* 2003.

[110] R. Carlson, P. Matuzic, and R. Simons, "Applying scrum to stabilize systems engineering execution," *CrossTalk*, pp. 1-6, 2012.

[111] S. D. P. Harker, K. D. Eason, and J. E. Dobson, "The change and evolution of requirements as a challenge to the practice of software engineering," in *IEEE International Symposium on Requirements Engineering*, 1993, pp. 266-272.

[112] B. Nuseibeh and S. Easterbrook, "Requirements engineering: a roadmap," presented at the Proceedings of the Conference on The Future of Software Engineering, Limerick, Ireland, 2000, pp. 35-46.

[113] W. Lam and M. Loomes, "Requirements evolution in the midst of environmental change: a managed approach," in *Proceedings of the Second Euromicro Conference on Software Maintenance and Reengineering*, 1998, pp. 121-127.

[114] E. F. Ecklund Jr, L. M. Delcambre, and M. J. Freiling, "Change cases: use cases that identify future requirements," in *ACM SIGPLAN Notices*, 1996, vol. 31, no. 10, pp. 342-358: ACM.

[115] M. Pichler, H. Rumetshofer, and W. Wahler, "Agile requirements engineering for a social insurance for occupational risks organization: A case study," in *Requirements Engineering, 14th IEEE International Conference*, 2006, pp. 251-256: IEEE.

[116] Z. Racheva, M. Daneva, and A. Herrmann, "A conceptual model of client-driven agile requirements prioritization: Results of a case study," in *Proceedings of the 2010 acm-ieee international symposium on empirical software engineering and measurement*, 2010, p. 39: ACM.

[117] S. Ibrahim, N. B. Idris, M. Munro, and A. Deraman, "A requirements traceability to support change impact analysis," *Asian Journal of Information Technology,* vol. 4, no. 4, pp. 345-355, 2005.

[118] S. A. Bohner and R. S. Arnold, "An introduction to software change impact analysis," in *Software Change Impact Analysis*, 1996, pp. 1-26: IEEE Computer Society Press.

[119] O. Gotel and A. Finkelstein, "Extended requirements traceability: results of an industrial case study," in *Requirements Engineering, 1997., Proceedings of the Third IEEE International Symposium on*, 1997, pp. 169-178: IEEE.

[120] M. F. Bashir and M. A. Qadir, "Traceability techniques: A critical study," in *2006 IEEE International Multitopic Conference,* 2006, pp. 265-268: IEEE.

[121] F. Bouquet, E. Jaffuel, B. Legeard, F. Peureux, and M. Utting, "Requirements traceability in automated test generation: application to smart card software

validation," in *ACM SIGSOFT Software Engineering Notes*, 2005, vol. 30, no. 4, pp. 1-7: ACM.

[122] J. Dick, "Design traceability," *IEEE software*, vol. 22, no. 6, pp. 14-16, 2005.

[123] A. Egyed and P. Grunbacher, "Automating requirements traceability: Beyond the record & replay paradigm," in *Automated Software Engineering, 2002. Proceedings, ASE 2002. 17th IEEE International Conference*, 2002, pp. 163-171: IEEE.

[124] M. Heindl and S. Biffl, "A case study on value-based requirements tracing," in *Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*, 2005, pp. 60-69: ACM.

[125] M. Jarke, "Requirements tracing," *Communications of the ACM*, vol. 41, no. 12, pp. 32-36, 1998.

[126] B. Ramesh and M. Jarke, "Toward reference models for requirements traceability," *IEEE Transactions on Software Engineering*, vol. 27, no. 1, pp. 58-93, 2001.

[127] R. Ravichandar, J. D. Arthur, and M. Pérez-Quiñones, "Pre-requirement specification traceability: bridging the complexity gap through capabilities," *arXiv preprint cs/0703012*, 2007.

[128] S. Rochimah, W. M. Wan-Kadir, and A. H. Abdullah, "An Evaluation of Traceability Approaches to Support Software Evolution," in *Internationl conference on Software Engineering Advances (ICSEA)*, 2007, p. 19.

[129] T. Verhanneman, F. Piessens, B. De Win, and W. Joosen, "Requirements traceability to support evolution of access control," in *ACM SIGSOFT Software Engineering Notes*, 2005, vol. 30, no. 4, pp. 1-7: ACM.

[130] P. Arkley and S. Riddle, "Overcoming the traceability benefit problem," in *13th IEEE International Conference on Requirements Engineering (RE'05)*, 2005, pp. 385-389: IEEE.

[131] J. Cleland-Huang, C. K. Chang, and M. Christensen, "Event-based traceability for managing evolutionary change," *IEEE Transactions on Software Engineering*, vol. 29, no. 9, pp. 796-810, 2003.

[132] J. Cleland-Huang, G. Zemont, and W. Lukasik, "A heterogeneous solution for improving the return on investment of requirements traceability," in *Requirements Engineering Conference, 2004. Proceedings, 12th IEEE International*, 2004, pp. 230-239: IEEE.

[133] J. Cleland-Huang, R. Settimi, C. Duan, and X. Zou, "Utilizing supporting evidence to improve dynamic requirements traceability," in *Requirements Engineering, 2005. Proceedings, 13th IEEE International Conference*, 2005, pp. 135-144: IEEE.

[134] O. Gotel and S. Morris, "Crafting the requirements record with the informed use of media," in *Proceedings of the First International Workshop on Multimedia Requirements Engineering (MeRE'06)*, 2006, p. 5: Citeseer.

[135] F. Blaauboer, K. Sikkel, and M. N. Aydin, "Deciding to adopt requirements traceability in practice," in *International Conference on Advanced Information Systems Engineering*, 2007, pp. 294-308: Springer.

[136] J. Cleland-Huang, "Toward improved traceability of non-functional requirements," in *Proceedings of the 3rd international workshop on Traceability in Emerging Forms of Software Engineering*, 2005, pp. 14-19: ACM.

[137] B. Ramesh, "Factors influencing requirements traceability practice," *Communications of the ACM*, vol. 41, no. 12, pp. 37-44, 1998.

[138] R. S. Arnold and S. A. Bohner, "Impact Analysis-Towards a Framework for Comparison," in *Conference on Software Maintenance*, 1993, vol. 93, pp. 292-301.

[139] G. Antoniol, G. Canfora, G. Casazza, and A. D. Lucia, "Identifying the starting impact set of a maintenance request: A case study," in *Software Maintenance and Reengineering, 2000. Proceedings of the Fourth European*, 2000, pp. 227-230: IEEE.

[140] Y. Li, J. Li, Y. Yang, and M. Li, "Requirement-centric traceability for change impact analysis: a case study," in *Making Globally Distributed Software Development a Success Story*, 2008, pp. 100-111: Springer.

[141] S. Ibrahim, N. B. Idris, M. Munro, and A. Deraman, "Integrating Software Traceability for Change Impact Analysis," *International Arab Journal of Information Technology*, vol. 2, no. 4, pp. 301-308, 2005.

[142] A. Göknil, I. Kurtev, and K. van den Berg, "Change impact analysis based on formalization of trace relations for requirements," in ECMDA Traceability Workshop (ECMDA-TW), SINTEF Report, 2008.

[143] A. Von Knethen, "Change-oriented requirements traceability. Support for evolution of embedded systems," in *Software Maintenance, 2002. Proceedings, International Conference*, 2002, pp. 482-485: IEEE.

[144] N. Ali and R. Lai, "A method of requirements change management for global software development," *Information and Software Technology*, vol. 70, pp. 49-67, 2016.

[145] J. Hassine, J. Rilling, J. Hewitt, and R. Dssouli, "Change impact analysis for requirement evolution using use case maps," in *Principles of Software Evolution, Eighth International Workshop*, 2005, pp. 81-90: IEEE.

[146] J. Hewitt and J. Rilling, "A light-weight proactive software change impact analysis using use case maps," in *Software Evolvability, 2005. IEEE International Workshop*, 2005, pp. 41-46: IEEE.

[147] L. Shi, Q. Wang, and M. Li, "Learning from evolution history to predict future requirement changes," in *Requirements Engineering Conference (RE), 2013 21st IEEE International*, 2013, pp. 135-144: IEEE.

[148] J. C. Maxwell, A. I. Antón, and P. Swire, "Managing changing compliance requirements by predicting regulatory evolution," in *Requirements Engineering Conference (RE), 2012 20th IEEE International*, 2012, pp. 101-110: IEEE.

[149] R. Malhotra and M. Khanna, "Mining the impact of object oriented metrics for change prediction using Machine Learning and Search-based techniques," in *Advances in Computing, Communications and Informatics (ICACCI), 2015 International Conference*, 2015, pp. 228-234: IEEE.

[150] C. Ingram and S. Riddle, "Using early stage project data to predict change-proneness," in *Proceedings of the 3rd International Workshop on Emerging Trends in Software Metrics*, 2012, pp. 42-48: IEEE Press.

[151] N. Tsantalis, A. Chatzigeorgiou, and G. Stephanides, "Predicting the probability of change in object-oriented systems," *IEEE Transactions on Software Engineering*, vol. 31, no. 7, pp. 601-614, 2005.

[152] S. Mirarab, A. Hassouna, and L. Tahvildari, "Using bayesian belief networks to predict change propagation in software systems," in *Program Comprehension, 2007. ICPC'07. 15th IEEE International Conference*, 2007, pp. 177-188: IEEE.

[153] N. N. B. Abdullah, S. Honiden, H. Sharp, B. Nuseibeh, and D. Notkin, "Communication patterns of agile requirements engineering," in *Proceedings of the 1st workshop on agile requirements engineering*, 2011, p. 1: ACM.

[154] B. Haugset and T. Stalhane, "Automated acceptance testing as an agile requirements engineering practice," in *System Science (HICSS), 2012 45th Hawaii International Conference*, 2012, pp. 5289-5298: IEEE.

[155] R. Popli, P. Malhotra, and N. Chauhan, "Estimating Regression Effort in Agile Environment," *International Journal of Computer Science and Communication,* vol. 5, pp. 23-28, 2014.

[156] M. Cohn, *User stories applied: For agile software development*. Addison-Wesley Professional, 2004.

[157] H. Leung and Z. Fan, "Software cost estimation," *Handbook of Software Engineering, Hong Kong Polytechnic University,* pp. 1-14, 2002.

[158] S. Rajper and Z. A. Shaikh, "Software Development Cost Estimation: A Survey," *Indian Journal of Science and Technology,* vol. 9, no. 31, 2016, pp.1-5.

[159] N. Fenton and J. Bieman, *Software metrics: a rigorous and practical approach*. CRC Press, 2014.

[160] M. H. Halstead, *Elements of software science*. Elsevier New York, 1977.

[161] P. G. Hamer and G. D. Frewin, "MH Halstead's Software Science-a critical examination," in *Proceedings of the 6th international conference on Software engineering*, 1982, pp. 197-206: IEEE Computer Society Press.

[162] V. Y. Shen, S. D. Conte, and H. E. Dunsmore, "Software science revisited: A critical analysis of the theory and its empirical support," *IEEE Transactions on Software Engineering,* no. 2, pp. 155-165, 1983.

[163] A. J. Albrecht and J. E. Gaffney, "Software function, source lines of code, and development effort prediction: a software science validation," *IEEE Transactions on Software Engineering,* no. 6, pp. 639-648, 1983.

[164] C. Jones, *Applied Software Measurement: Assuring Productivity and Quality,* 2nd edn McGraw-Hill, NewYork 1997.

[165] S. Kumari and S. Pushkar, "Performance analysis of the software cost estimation methods: a review," *International Journal of Advanced Research in Computer Science and Software Engineering,* vol. 3, no. 7, pp. 229-238, 2013.

[166] P. Abrahamsson, I. Fronza, R. Moser, J. Vlasenko, and W. Pedrycz, "Predicting development effort from user stories," in *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium*, 2011, pp. 400-403: IEEE.

[167] M. Ceschi, A. Sillitti, G. Succi, and S. De Panfilis, "Project management in plan-based and agile companies," *IEEE software,* vol. 22, no. 3, pp. 21-27, 2005.

[168] N. C. Haugen, "An empirical study of using planning poker for user story estimation," in *Agile Conference, 2006*, 2006, pp. 9 pp.-34: IEEE.

[169] V. Mahnič and T. Hovelja, "On using planning poker for estimating user stories," *Journal of Systems and Software,* vol. 85, no. 9, pp. 2086-2095, 2012.

[170] S. K. Khatri, S. Malhotra, and P. Johri, "Use case point estimation technique in software development," in *Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO), 2016 5th International Conference,* 2016, pp. 123-128: IEEE.

[171] N. Nunes, L. Constantine, and R. Kazman, "IUCP: Estimating interactive-software project size with enhanced use-case points," *IEEE software,* vol. 28, no. 4, pp. 64-73, 2011.

[172] E. Coelho and A. Basu, "Effort estimation in agile software development using story points," *International Journal of Applied Information Systems (IJAIS),* vol. 3, no. 7, pp. 7-10, 2012.

[173] P. R. Hill, *Practical project estimation: a toolkit for estimating software development effort and duration*. International Software Benchmarking Standards Group, 2010.

[174] A. Panda, S. M. Satapathy, and S. K. Rath, "Empirical validation of neural network models for agile software effort estimation based on story points," *Procedia Computer Science,* vol. 57, pp. 772-781, 2015.

[175] A. G. Silvius, "Business & IT Alignment in theory and practice," in *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, 2007, pp. 211b-211b: IEEE.

[176] B. Campbell, "Alignment: Resolving ambiguity within bounded choices," in *Proceedings of the Pacific Asia Conference on Information Systems (PACIS),* p. 54, 2005.

[177] P. Tallon and K. L. Kraemer, "A process-oriented assessment of the alignment of information systems and business strategy: implications for IT business value," *Center for Research on Information Technology and Organizations,* 1999.

[178] A. Fuggetta and A. L. Wolf, *Software process*. John Wiley & Sons, Inc., 1996.

[179] E. J. Barry, T. Mukhopadhyay, and S. A. Slaughter, "Software project duration and effort: an empirical study," *Information Technology and Management,* vol. 3, no. 1-2, pp. 113-136, 2002.

[180] V. Basili, J. Heidrich, M. Lindvall, J. Münch, M. Regardie, D. Rombach, C. Seaman, and A Trendowicz, "Bridging the gap between business strategy and software development," *ICIS 2007 Proceedings,* p. 25, 2007.

[181] T. Goradia, "Dynamic impact analysis: A cost-effective technique to enforce error-propagation," in *ACM SIGSOFT Software Engineering Notes*, 1993, vol. 18, no. 3, pp. 171-181: ACM.

[182] J. Law and G. Rothermel, "Whole program path-based dynamic impact analysis," in *Software Engineering, 2003. Proceedings. 25th International Conference*, 2003, pp. 308-318: IEEE.

[183] P. Tonella, "Using a concept lattice of decomposition slices for program understanding and impact analysis," *IEEE Transactions on Software Engineering,* vol. 29, no. 6, pp. 495-509, 2003.

[184] M. Aoyama, "Agile software process and its experience," in *Software Engineering, 1998. Proceedings of the 1998 International Conference*, 1998, pp. 3-12: IEEE.

[185] S. Nerur, R. Mahapatra, and G. Mangalaraj, "Challenges of migrating to agile methodologies," *Communications of the ACM,* vol. 48, no. 5, pp. 72-78, 2005.

[186] P. Karesma, "Scaling Agile Methods," Diss 2016.

[187] D. J. Reifer, F. Maurer, and H. Erdogmus, "Scaling agile methods," *IEEE software,* vol. 20, no. 4, pp. 12-14, 2003.

[188] F. J. Pino, O. Pedreira, F. García, M. R. Luaces, and M. Piattini, "Using Scrum to guide the execution of software process improvement in small organizations," *Journal of Systems and Software,* vol. 83, no. 10, pp. 1662-1677, 2010.

[189] D. E. Strode, S. L. Huff, and A. Tretiakov, "The impact of organizational culture on agile method use," in *System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference*, 2009, pp. 1-9: IEEE.

[190] E. Bjarnason, K. Wnuk, and B. Regnell, "A case study on benefits and side-effects of agile practices in large-scale requirements engineering," in *Proceedings of the 1st Workshop on Agile Requirements Engineering*, 2011, p. 3: ACM.

[191] B. A. Rajabi and S. P. Lee, "Change management in business process modeling survey," in *Information Management and Engineering, 2009. ICIME'09. International Conference*, 2009, pp. 37-41: IEEE.

[192] C. André and F. Mallet, "Specification and verification of time requirements with CCSL and esterel," in *ACM Sigplan Notices*, 2009, vol. 44, no. 7, pp. 167-176: ACM.

[193] G. Dillingham, "Air traffic control: evolution and status of FAA’s Automation Program," Technical Report GAO/T-RCED/AIMD-98-85, *Washington, DC: United States General Accounting Office,* 1998.

[194] K. Pohl, *Requirements engineering: fundamentals, principles, and techniques*. Springer Publishing Company, Incorporated, 2010.

[195] J. V. Guttag and J. J. Horning, *Larch: languages and tools for formal specification*. Springer Science & Business Media, 2012.

[196] I. J. Jureta, A. Borgida, N. A. Ernst, and J. Mylopoulos, "Techne: Towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling," in *2010 18th IEEE International Requirements Engineering Conference*, 2010, pp. 115-124: IEEE.

[197] R. Van Solingen, V. Basili, G. Caldiera, and H. D. Rombach, "Goal question metric (gqm) approach," *Encyclopedia of Software Engineering*, pp. 578-583, 2002.

[198] M. Weiss, "Resource description framework," in *Encyclopedia of Database Systems*, 2009, pp. 2423-2425: Springer

[199] H. Koziolek, "Goal, question, metric," in *Dependability metrics*: Springer, 2008, pp. 39-42.

[200] P. Berander and P. Jönsson, "A goal question metric based approach for efficient measurement framework definition," in *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, 2006, pp. 316-325: ACM.

[201] D. V. Subramanian and A. Geetha, "Adaptation of goal question metric technique for evaluation of knowledge management systems," *Review of Knowledge Management*, vol. 1, no. 2, p. 4-11, 2011.

[202] V. Basili, J. Heidrich, M. Lindvall, J. Munch, M. Regardie, and A. Trendowicz, "GQM^{+} Strategies - Aligning Business Strategies with Software Measurement," in *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, 2007, pp. 488-490.

[203] H.-J. Happel and S. Seedorf, "Applications of ontologies in software engineering," in *Proc. of Workshop on Sematic Web Enabled Software Engineering"(SWESE) on the ISWC*, 2006, pp. 5-9: Citeseer.

[204] T. Tse and L. Pong, "An examination of requirements specification languages," *The Computer Journal*, vol. 34, no. 2, pp. 143-152, 1991.

[205] P. B. Checkland, "Information systems and systems thinking: Time to unite?," *International Journal of Information Management*, vol. 8, no. 4, pp. 239-248, 1988.

[206] E. Mumford, *Redesigning human systems*. IGI Global, 2003.

[207] A. M. Davis, "The design of a family of application-oriented requirements languages," *Computer,* vol. 5, no. 15, pp. 21-28, 1982.

[208] C. Roche, "Ontoterminology: How to unify terminology and ontology into a single paradigm," in *LREC 2012, Eighth International Conference on Language Resources and Evaluation,* 2012, pp. p. 2626-2630: European Language Resources Association.

[209] L. Gillam, M. Tariq, and K. Ahmad, "Terminology and the construction of ontology," *Terminology,* vol. 11, no. 1, pp. 55-81, 2005.

[210] V. Castañeda, L. Ballejos, M. L. Caliusco, and M. R. Galli, "The use of ontologies in requirements engineering," *Global Journal of Researches in Engineering,* vol. 10, no. 6, pp. 2-8, 2010.

[211] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition,* vol. 5, no. 2, pp. 199-220, 1993.

[212] W. Lam and M. Loomes, "Requirements evolution in the midst of environmental change: a managed approach," in *Software Maintenance and Reengineering, 1998. Proceedings of the Second Euromicro Conference on*, 1998, pp. 121-127: IEEE.

[213] W. Lam and V. Shankararaman, "Managing change in software development using a process improvement approach," in 24th *Euromicro Conference, 1998. Proceedings,* 1998, vol. 2, pp. 779-786: IEEE.

[214] H. Xiao, J. Quo, and Y. Zou, "Supporting change impact analysis for service oriented business applications," in *Systems Development in SOA Environments, 2007. SDSOA'07: ICSE Workshops 2007. International Workshop on,* 2007, pp. 6-12: IEEE.

[215] R. K. Yin, "Discovering the future of the case study method in evaluation research," *Evaluation practice,* vol. 15, no. 3, pp. 283-290, 1994.

[216] R. K. Yin, "Case study methods," in J. L. Green, G. Camilli, & P. B. Elmore (Eds.), *Handbook of complementary methods in education research*, 2006, pp. 111-122.

[217] B. W. Boehm, *Software Engineering Economics*. Prentice Hall PTR, 1981, p. 768.

[218] P. Lago, H. Muccini, and H. v. Vliet, "A scoped approach to traceability management," *J. Syst. Softw.*, vol. 82, no. 1, pp. 168-182, 2009.

[219] D. Zowghi and R. Offen, "A logical framework for modeling and reasoning about the evolution of requirements," in *Requirements Engineering, 1997., Proceedings of the Third IEEE International Symposium on*, 1997, pp. 247-257: IEEE.

[220] R. Sugden and M. Strens, "Strategies, tactics and methods for handling change," in *Engineering of Computer-Based Systems, 1996. Proceedings., IEEE Symposium and Workshop on*, 1996, pp. 457-463: IEEE.

[221] O. C. Gotel and A. C. Finkelstein, "An analysis of the requirements traceability problem," in *Requirements Engineering, 1994., Proceedings of the First International Conference on*, 1994, pp. 94-101: IEEE.

[222] R. Torkar, T. Gorschek, R. Feldt, M. Svahnberg, U. A. Raja, and K. Kamran, "Requirements traceability: a systematic review and industry case study," *International Journal of Software Engineering and Knowledge Engineering*, vol. 22, no. 03, pp. 385-433, 2012.

[223] E. Brynjolfsson, A. A. Renshaw, and M. van Alstyne, "The Matrix of Change: A Tool for Business Process Reengineering," *Working paper 189, Centre for Coordinating Sciences, Massachusetts Institute of Technology*, 1996.

[224] S. Wang and M. A. M. Capretz, "A Dependency Impact Analysis Model for Web Services Evolution," in *Web Services, 2009. ICWS 2009. IEEE International Conference*, 2009, pp. 359-365.

[225] J. Zhang, Y. C. Chang, and K. J. Lin, "A dependency matrix based framework for QoS diagnosis in SOA," in *Service-Oriented Computing and Applications (SOCA), 2009 IEEE International Conference*, 2009, pp. 1-8.

[226] A. M. Omer and A. Schill, "Web service composition using input/output dependency matrix," in *Proceedings of the 3rd Workshop on Agent-oriented Software Engineering Challenges for Ubiquitous and Pervasive Computing*, 2009, pp. 21-26: ACM.

[227] K. van den Berg, "Change impact analysis of crosscutting in software architectural design," in *Proceedings of Workshop on Architecture-Centric Evolution at 20th ECOOP*, 2006.

[228] B. Li, "Managing dependencies in component-based systems based on matrix model," in *proceedings of Net.objectdays(NODE) 2003 conference*, 2003, pp. 22-25.

[229] M. Ruiz, V. Mejia, and A. Kaplan, "Information system comprised of synchronized software application moduless with individual databases for implementing and changing business requirements to be automated," ed: Google Patents, Patent Application 10/633,959, 2005.

[230] R. H. Katz and E. E. Chang, *Managing change in a computer-aided design database*. in *Proceedings of 13th VLDB Conference*, 1987, pp. 445-462.

[231] S. Maadawy and A. Salah, "Measuring Change Complexity from Requirements: A Proposed Methodology," *International Magazine on Advances in Computer Science and Telecommunications*, vol. 3, no. 1, 2012.

[232] M. W. Dickinson, A. C. Thornton, and S. Graves, "Technology portfolio management: optimizing interdependent projects over multiple time periods," *IEEE Transactions on Engineering Management*, vol. 48, no. 4, pp. 518-527, 2001.

[233] L. Maciaszek, *Requirement analysis and system design*. Pearson Education Limited, Edinburgh Gate, England, 2007.

[234] P. Selonen, K. Koskimies, and M. Sakkinen, "Transformations between UML diagrams," *Journal of Database Management*, vol. 14, no. 3, pp. 37-55, 2003.

[235] M. S. Kilpinen, "The Emergence of Change at the Systems Engineering and Software Design Interface," PhD Diss, University of Cambridge, 2008.

[236] S. Jayatilleke and R. Lai, "A systematic review on Requirement Change Management," *Information and Software Technology*, vol. 93, pp. 163-185, 2018. DOI: 10.1016/j.infsof.2017.09.004.

[237] D. Kiritsis, K.-P. Neuendorf, and P. Xirouchakis, "Petri net techniques for process planning cost estimation," *Advances in Engineering Software*, vol. 30, no. 6, pp. 375-387, 1999.

[238] P. E. D. Love, D. J. Edwards, H. Watson, and P. Davis, "Rework in Civil Infrastructure Projects: Determination of Cost Predictors," *Journal of Construction Engineering and Management*, vol. 136, no. 3, pp. 275-282, 2010.

[239] P. E. D. Love, "Influence of Project Type and Procurement Method on Rework Costs in Building Construction Projects," *Journal of Construction Engineering and Management*, vol. 128, no. 1, pp. 18-29, 2002.

[240] K. Butler and W. Lipke, "Software process achievement at tinker air force base," Technical Report CMU/SEI-2000-TR-014, Carnegie-Mellon Software Engineering Institute, 2000.

[241] A. G. Cass, S. M. Sutton, and L. J. Osterweil, "Formalizing rework in software processes," in P*roceedings of European Workshop on Software Process Technology (EWSPT)*, 2003, vol. 2786, pp. 16-31: Springer.

[242] First CeBASE eWorkshop, "Focusing on the cost and effort due to software defects," *NSF Center for Empirically Based Software Engineering*, 2001. http://www.cebase.org/www/researchActivities/defectReduction/eworkshop1

[243] V. R. Basili, S. E. Condon, K. E. Emam, R. B. Hendrick, and W. Melo, "Characterizing and modeling the cost of rework in a library of reusable software components," in *Proceedings of the 19th international conference on Software Engineering*, Boston, Massachusetts, USA, 1997, pp. 282-291: ACM.

[244] U. T. Raja, M.J., "Defining and Evaluating a Measure of Open Source Project Survivability," *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 169-174, 2012.

[245] R. N. Charette, "Why software fails [software failure]," *IEEE Spectrum*, vol. 42, no. 9, pp. 42-49, 2005.

[246] X. O. Zhao, L.J., "An approach to modeling and supporting the rework process in refactoring," in *International Conference on Software and System Process (ICSSP)*, 2012, pp. 110-119.

[247] J. Highsmith and A. Cockburn, "Agile software development: The business of innovation," *Computer*, vol. 34, no. 9, pp. 120-127, 2001.

[248] C. Ghezzi, M. Jazayeri, and D. Mandrioli, *Fundamentals of software engineering*. Prentice Hall PTR, 2002.

[249] P. E. Love and J. Smith, "Benchmarking, benchaction, and benchlearning: rework mitigation in projects," *Journal of Management in Engineering*, vol. 19, no. 4, pp. 147-159, 2003.

[250] J. Radatz, A. Geraci, and F. Katki, "IEEE standard glossary of software engineering terminology," *IEEE Standard*, vol. 610121990, no. 121990, p. 3, 1990.

[251] K. H. Bennett and V. T. Rajlich, "Software maintenance and evolution: a roadmap," in *Proceedings of the Conference on the Future of Software Engineering*, 2000, pp. 73-87: ACM.

[252] *ISO12207 Information technology - Software life cycle processes*, 1995.

[253] G. Canfora and A. Cimitile, "Software maintenance," in *Handbook of Software Engineering and Knowledge Engineering: Volume I: Fundamentals*: World Scientific, 2001, pp. 91-120.

[254] S. Jayatilleke, R. Lai, and K. Reed, "Managing Software Requirements Changes through Change Specification and Classification," *To appear in Computer Science and Information Systems*, 2017. DOI: 10.2298/CSIS161130041J.

[255] S. Jayatilleke, R. Lai, and K. Reed, "A method of requirements change analysis," *Requirements Engineering,* pp. 1-16, 2017. DOI: 10.1007/s00766-017-0277-7.

[256] T. Wijayasiriwardhane and R. Lai, "Component Point: A system-level size measure for component-based software systems," *Journal of Systems and Software,* vol. 83, no. 12, pp. 2456-2470, 2010.

[257] S. Mahmood and R. Lai, "A complexity measure for UML component-based system specification," *Software: Practice and Experience,* vol. 38, no. 2, pp. 117-134, 2008.

[258] P. N. Jeziorek, "Cost estimation of functional and physical changes made to complex systems," Massachusetts Institute of Technology, PhD Diss, 2005.

[259] L. Lavazza and G. Valetto, "Requirements-based estimation of change costs," *Empirical Software Engineering,* vol. 5, no. 3, pp. 229-243, 2000.

Appendix A

Following is the copyright information of the papers published (online)/in press/ submitted in this thesis:

1. The paper entitled “A Systematic Review of Requirements Change Management” has been published in Information and Software Technology: As stated at https://www.elsevier.com/about/company-information/policies/copyright authors have the right to share their article for personal use, which encompasses inclusion in their theses.

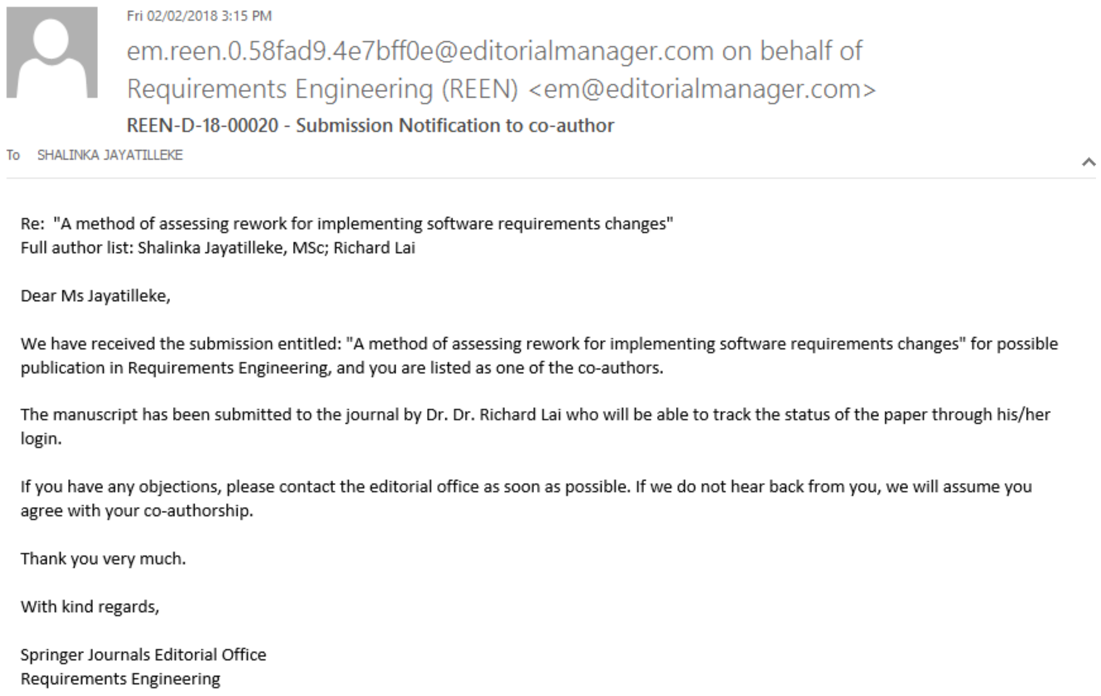
2. The paper entitled “Managing Software Requirements Changes through Change Specification and Classification” has been published online in ComSIS: http://www.comsis.org/copyright.php “Authors/employers retain all proprietary rights in any process, procedure, or article of manufacture described in the Work”.

3. The paper entitled “A Method of Requirements Change Analysis” has been published online in Requirements Engineering: As stated at http://www.springernature.com/gp/researchers/sharedit “As part of the Springer Nature SharedIt initiative, you can now publicly share a full-text view-only version of your paper by using the link below. If you have selected an Open Access option for your paper, or where an individual can view content via a personal or institutional subscription, recipients of the link will also be able to download and print the PDF. All readers of your article via the shared link will also be able to use Enhanced PDF features such as annotation tools, one-click supplements, citation file exports and article metrics.” http://rdcu.be/ub4S

4. The paper entitled “A Method of Analysing Rework for Implementing Software Requirements Changes” has been submitted to Requirements Engineering. Awaiting outcome of submission.

Appendix B

Following is the proof of submission of “A Method of Analysing Rework for Implementing Software Requirements Changes” in Requirements Engineering:



Fri 02/02/2018 3:15 PM

em.reen.0.58fad9.4e7bff0e@editorialmanager.com on behalf of Requirements Engineering (REEN) <em@editorialmanager.com>

REEN-D-18-00020 - Submission Notification to co-author

To SHALINKA JAYATILLEKE

Re: "A method of assessing rework for implementing software requirements changes"
Full author list: Shalinka Jayatilleke, MSc; Richard Lai

Dear Ms Jayatilleke,

We have received the submission entitled: "A method of assessing rework for implementing software requirements changes" for possible publication in Requirements Engineering, and you are listed as one of the co-authors.

The manuscript has been submitted to the journal by Dr. Dr. Richard Lai who will be able to track the status of the paper through his/her login.

If you have any objections, please contact the editorial office as soon as possible. If we do not hear back from you, we will assume you agree with your co-authorship.

Thank you very much.

With kind regards,

Springer Journals Editorial Office
Requirements Engineering