



Behaviour Engineering based Approaches to Requirements Change Management and Requirements Defects Detection

Sajid Anwer

Bachelor of Science in Computer Science (BSCS) from COMSATS Institute of
Information Technology, Lahore, Pakistan

Master of Science in Computer Science (MSCS) from King Fahd University of
Petroleum & Minerals, Saudi Arabia

**School of Information and Communication Technology
Griffith University
Australia**

*SUBMITTED IN FULFILMENT OF THE REQUIREMENTS OF THE DEGREE OF
DOCTOR OF PHILOSOPHY*

August 2021

Abstract

The main goal of software development is to produce software that fulfils users' needs. Users' needs are usually expressed as system requirements that are elicited during the Requirements Engineering (RE) phase of the Software Development Life Cycle (SDLC). In real-world systems, it is difficult to elicit all the requirements early in development. As a result, in practice, changes to the initial requirements occur frequently. Moreover, some other factors may also trigger Requirements Changes (RCs) such as technological advancements and changes in stakeholders' needs. RCs can also be made even after the system has been deployed. To manage RCs, systematic approaches, which usually fall under the umbrella of Requirements Change Management (RCM), are required.

Efficient RCM is essential for the success of large software projects. To develop efficient RCM, challenges that might be faced during the execution of RCM processes need to be investigated and then suitable approaches must be designed to address them.

The objective of this thesis was to investigate RCM challenges and to provide approaches and guidelines to address these challenges. To achieve the objectives, we carried out an evidence-based study to identify RCM challenges and then developed a suitable RCM process, proposed in the format of an ISO/IEC standard. Subsequently, we used some semi-formal modelling languages and formal languages to tackle two key RCM challenges.

In particular, for the evidence-based study, we used a Systematic Literature Review (SLR) to identify challenges related to RCM in both in-house software development and Global Software Development (GSD). We then conducted a questionnaire-based survey to get industry practitioners' opinions of the findings of our literature review. Based on both data sets, a chi-square test was used to analyse the relative importance of identified RCM challenges in the context of in-house software development and GSD. The results reveal some insights into RCM key issues and will help software engineers and partitioner's to design more efficient RCM processes.

Based on our findings, we propose an RCM process in the format of an ISO/IEC standard. The proposed process is elaborated through a seven-step model. And then, Composition Trees (CTs) are used to compare the proposed process with the RCM related processes in an existing ISO/IEC standards. The comparison results show that the proposed process has addressed many key aspects of RCM that had previously been missed. In addition,

mapping is used to verify that most RCM challenges are covered by proposed RCM process outcomes. The results are further validated through an industry survey.

To increase the depth of our work related to RCM, we propose a systematic approach to estimate the impact of requirements changes on other software artefacts. In the SLR, change impact analysis is one of the most cited RCM challenges to have been identified. Our approach uses an Integrated Behaviour Tree (IBT) to model system requirements. We then convert the IBT to an Integrated Composition Tree (ICT), and the ICT into a Requirements Component Dependency Network (RCDN). All these models help to trace and visualize the change propagation from requirements to other software artefacts such as software architecture. Moreover, RCDN and component interface diagrams help to identify the impacted components and help to quantify the change impact, denoted as the change impact indicator, which could provide a more objective measure for calculating the development cost of the proposed change.

To address another RCM challenge, we propose a two-step approach to detect requirements defects. In the first step, a formal model based on Context-Free Grammar (CFG) is proposed to formalise the translation of software requirements described in natural language to a Behaviour Tree (BT). In this approach, we first define normal formed BTs based on a newly introduced Inertia Axiom, and then use CFG to generate or verify the normal formed BTs. A tool is developed to validate the proposed approach.

In the second step, we formalise the four most common requirements defects based on the normalised BTs and define algorithms to detect them. To detect those requirements defects, we use BTs to model system requirements and then develop an algorithm to translate BTs to Web Ontology Language (OWL). The process of translating natural language software requirements into BTs and then into a formal language helps to overcome the difficulty of verifying whether a formal representation is faithful to its original natural language form. And then SPARQL queries are used to query the OWL formulation of the requirements and to detect requirements defects. Lastly, a tool is developed and applied to a real-world system to validate the proposed approach.

An important point is that requirements defects also arise during requirements elicitation and analysis stages. Therefore, in this context, some results of this research are relevant to another problem in the RE phase of SDLC.

Statement of Originality

This work has not previously been submitted for a degree or diploma in any university. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made in the thesis itself.



Sajid Anwer

Acknowledgements

The work presented in this thesis would not have been possible without the support, guidance and encouragement of my two supervisors. I would also like to thank my friends and family for their constant support.

Foremost, It is a matter of great pleasure and honour to express my deep sense of gratitude for the continuous guidance and precious advice extended by my principal supervisor Dr. Lian Wen. He has been a constant source of enormous motivation and enthusiastic support throughout the span of the PhD. His constructive and kind comments have shown me the right direction to obtain valuable knowledge for my research. Due to his compassionate support and competent coaching, I have been able to finish my PhD. Thank you, Dr. Lian Wen, for your immense support regarding my work, and I hope we will continue to work together.

I want to express my appreciation and respect to my associate supervisor Dr. Zhe Wang, for his support, constructive comments, which were of utmost significance. His insightful discussions and friendly counselling backed me in achieving my objectives. I would also like to extend my gratitude to Professor Abdul Sattar for his encouragement.

I want to thank Griffith University for giving me the opportunity to pursue post-graduate studies and provides excellent research facilities and financial assistance during my PhD program. I am thankful to all the heads that have been appointed during my stay at Griffith and the administrative staff of the school of information and communication technology for their assistance. I am also thankful to the school of higher degree research of Griffith University for their support.

I want to thank my family and friends for supporting me throughout the years. I want to extend my deepest gratitude and respect to my parents and my siblings for believing in me and giving me immense moral support. I want to thank my wife, Maria Ayub and my children Abdur Rehman and Meerab Fatima, for their patience and support. I would also like to thank all my friends for their continued support.

Finally, I would like to thank Allah Almighty, who bestowed me the illumination and courage to fulfil my PhD commitment. He listens to my prayers and keeps me safe throughout my PhD, especially during the difficult time of COVID-19. He provided me with the opportunities and gave me the determination to do my research.

I want to end the acknowledgements section by leaving a few words for my readers. Following is the quote from one of the most accomplished scientists, Marie Curie.

"Life is not easy for any of us. But what of that? We must have perseverance and, above all, confidence in ourselves. We must believe that we are gifted for something and that this thing must be attained".

Contents

Abstract	i
Statement of Originality	iii
Acknowledgements	iv
Contents	vi
List of Figures	xi
List of Tables	xiv
Abbreviations	xvi
1 Introduction	1
1.1 Background	1
1.2 Motivation	2
1.2.1 Identifying RCM Challenges	3
1.2.2 Designing an RCM Process	4
1.2.3 Change Impact Analysis	5
1.2.4 Detect Requirements Defects	6
1.3 Aims and Objectives	8
1.4 Contributions of the Thesis	8
1.4.1 C1 - RCM Challenges	9
1.4.2 C2 - RCM Process in the Format of an ISO/IEC Standard	10
1.4.3 C3 - Change Impact Analysis	11
1.4.4 C4a & C4b - Requirements Defects Detection	12
1.5 Thesis Outline	14
2 Background and Literature Review	16
2.1 P1: Requirements Change Management	17
2.1.1 Overview of RE and RCM	17
2.1.2 C1: RCM Challenges through SLR and Questionnaire Survey	19
2.1.3 C2: RCM Processes	20
2.1.4 C3: Change Impact Analysis	21
2.2 P2: Requirements Defects Detection Overview	22
2.2.1 C4a: Modelling Language (BT) Formalisation	22
2.2.2 C4b: Requirements Defects Detection	23
2.3 Data Collection	24

2.3.1	T2: Systematic Literature Review	24
2.3.2	T3: Questionnaire Survey	26
2.4	T4: Software Processes and Standards	27
2.5	T5: Behaviour Engineering	29
2.5.1	T5a: Behaviour Tree	30
2.5.2	T5b: Composition Tree	33
2.6	T6: Context-Free Grammar	36
2.7	Ontologies and Reasoning	37
2.7.1	T7: Web Ontology Language (OWL)	38
2.7.2	T8: SPARQL	40
2.8	P1.1: Global Software Development	40
2.8.1	Project Management Structures in GSD	40
3	Requirements Change Management Challenges	44
3.1	Introduction	45
3.2	Related Work	48
3.3	Research Methodology	49
3.3.1	Data Collection via SLR	49
3.3.1.1	Search Strategy	50
3.3.1.2	Studies Selection	53
3.3.1.3	Data Extraction and Synthesis	55
3.3.2	Data Collection via Questionnaire Survey	56
3.3.2.1	Survey Design	56
3.3.2.2	Survey Instrument	57
3.3.2.3	Survey Execution and Data Pre-processing	57
3.4	SLR Results and Analysis	58
3.4.1	Overview of the Studies	58
3.4.2	SLR Findings of RCM in In-House Software Development Approach (RQ1)	61
3.4.3	SLR Findings of RCM in GSD Context (RQ3)	64
3.5	Questionnaire Survey Results and Analysis	66
3.5.1	Industry Survey Findings of RCM Process (RQ2)	67
3.5.2	Industry Survey Findings of RCM in GSD Context (RQ4)	69
3.5.3	Industry Survey Findings Analysis based on in-house and GSD Approach (RQ5)	70
3.5.4	Industry Survey Findings Analysis Based on Centralised and Distributed Global Project Structure (RQ6)	71
3.6	Comparison between SLR and Questionnaire Survey Data Sets (RQ7)	73
3.7	Discussion	77
3.7.1	Critical RCM Challenges	77
3.7.2	Software Development Approach-Based Analysis	78
3.7.3	Global Project Management Structure-Based Analysis	80
3.7.4	Implications of This Research	81
3.8	Limitations	83
3.9	Conclusion	84

4	Requirements Change Management Process in the Format of an ISO/IEC Standard	86
4.1	Introduction	87
4.2	Related Work	88
4.3	Proposed Requirements Change Management Process	89
4.3.1	Definition	89
4.3.2	Requirements Change Management Process Model Description . .	90
4.4	Comparison with Existing Processes	92
4.4.1	Composition Tree Modelling of Configuration Management Process	92
4.4.2	Composition Tree Modelling the Requirements Change Manage- ment Process	93
4.4.3	Comparison between RCM Process and Configuration Management Process	95
4.5	Mapping between RCM Challenges and RCM Process Outcomes	97
4.6	Conclusion	100
5	Change Impact Analysis	102
5.1	Introduction	103
5.2	Key Elements of BECIA	105
5.2.1	Conversion of an IBT into an ICT	106
5.2.2	Conversion of an ICT into RCDN	109
5.2.3	Transform IBT into CID	113
5.2.4	Change Impact Analysis Metric	114
5.3	BECIA Workflow with an Example	118
5.3.1	BECIA Workflow	118
5.3.2	Impact Analysis Example	119
5.3.2.1	Steps 2 and 3- Draw/modify RBTs and Integrate them with the IBT	120
5.3.2.2	Step 4- Convert an IBT into an ICT	120
5.3.2.3	Step 5- Convert an ICT into an RCDN	121
5.3.2.4	Steps 6 and 7- Project out CIDs from an IBT and Calcu- late Change Impact	122
5.3.3	Alternative Design Approach	125
5.4	Case Study	129
5.5	Related Work	137
5.5.1	Change Impact Analysis in System Requirements	138
5.5.2	Change Impact Analysis in System Design and Architecture	139
5.5.3	Change Impact Analysis in Source Code	139
5.5.4	Comparison with Existing Work	140
5.6	Conclusion	142
6	A Formal Model for Behaviour Trees based on Context-Free Grammar	143
6.1	Introduction	144
6.2	Related Work	146
6.3	Formal Model	147
6.3.1	Normal Form of Requirement Behaviour Tree	147
6.3.2	Context-Free Grammar for Normal Formed BT	149
6.4	An Example	155

6.5	Tool-BT Compiler	161
6.6	Conclusion	162
7	Formalisation of Requirements Defects Detection	164
7.1	Introduction	164
7.2	Related Work	167
7.2.1	Requirements Defects Detection using Formal Languages	167
7.2.2	Requirements Translation directly to Formal Languages	167
7.2.3	Requirements Translation using Semi-Formal Languages as a Bridge	168
7.2.4	Requirements Defects Analysis using Natural Language Processing	169
7.2.5	Application of SPARQL for Requirements Analysis	169
7.2.6	Limitations of Current Research related to Requirements Defects Detection	170
7.3	Requirements Defects Detection Framework	171
7.3.1	BERDD Structure	171
7.3.2	BERDD Workflow	172
7.4	Requirements Defects	173
7.4.1	Requirements Defects Classification	174
7.4.2	Normal Form of Requirements Behaviour Tree	175
7.4.3	Incompleteness Defect and its Detection	178
7.4.3.1	Definition	178
7.4.3.2	Example	178
7.4.4	Ambiguity Defect and its Detection	180
7.4.4.1	Definition	180
7.4.4.2	Example	181
7.4.5	5.4 Redundancy Defect and its Detection	181
7.4.5.1	Definition	181
7.4.5.2	Example	182
7.4.6	Inconsistency Defect and its Detection	183
7.4.6.1	Definition	183
7.4.6.2	Example	183
7.5	Tool (RDI) Environment	185
7.5.1	RDI Workflow	186
7.5.2	Behaviour Tree Representation in OWL	187
7.6	An Example	192
7.7	Discussion and Limitations	197
7.7.1	Comparison with Existing Studies	197
7.7.2	Limitations	199
7.8	Conclusion	200
8	Conclusion	201
8.1	Summary of Thesis	201
8.2	Future Work	204
A	Appendices for Chapter 3	206
A.1	Search Strings for Different Databases	206
A.1.1	IEEE Xplore	206

A.1.2	Science Direct	207
A.1.3	SpringerLink	207
A.1.4	ACM	208
A.2	List of Primary Studies in SLR	208
A.3	RCM Challenges Categories Identified via SLR	212
A.4	Questionnaire Survey	213
A.5	Ethical Clearance to Conduct Questionnaire Survey	216
A.6	Participants Demographic Details	216
A.7	SLR Primary Studies Quality Assessment Results	221
B	Appendices for Chapter 7	223
B.1	R1 OWL File	223
B.2	SPARQL Queries	224
B.3	AIP IBT File	225
	 Publications and Submitted Papers	 226
	 Bibliography	 227

List of Figures

1.1	Contributions overview	9
2.1	Overview of requirements engineering domain adapted from [1]	17
2.2	SLR Process	25
2.3	The behaviour engineering approach to creating a software design	30
2.4	BT example	32
2.5	BT node attributes	32
2.6	The IBT of R1	35
2.7	The IBT of R1 and R2	35
2.8	The IBT of CAR system	36
2.9	Chomsky hierarchy languages classification	37
2.10	GSD Overview	41
2.11	Centralised global project structure adapted from [2].	41
2.12	Distributed global project structure adapted from [2].	42
3.1	Research methodology	49
3.2	SLR process steps and number of studies at each step	54
3.3	Number of selected studies published per year and their distribution over source type	59
3.4	Bubble plot with year of publication and research method	59
4.1	Requirements change management model	91
4.2	The composition tree for CMP	93
4.3	The CT for RCMP	94
4.4	CCT for CMP of 12207 and proposed RCM process	96
5.1	Example IBT	107
5.2	Steps to model example ICT	108
5.3	Example ICT	109
5.4	The RCDN for the example ICT	111
5.5	The CID for component C1	113
5.6	BECIA workflow	118
5.7	R5 RBT	120
5.8	Updated IBT	120
5.9	Updated ICT	121
5.10	Updated RCDN	122
5.11	The CID for component C1	123
5.12	The CID for component C4	123
5.13	The CID for component C3	123

5.14	The CID for component C6	124
5.15	Second design RBT for R5	126
5.16	Updated IBT for the second design	126
5.17	Updated ICT for the second design	126
5.18	Updated RCDN for the second design	127
5.19	The CID for component C1 based on the second design	127
5.20	The CID for component C4 based on the second design	127
5.21	The CID for component C2 based on the second design	128
5.22	The CID for component C5 based on the second design	128
5.23	STLISMS updated IBT	131
5.24	STLISMS updated ICT	132
5.25	STLISMS updated RCDN	133
5.26	STLISMS-The CID for C1	134
5.27	STLISMS-The CID for C5	134
5.28	STLISMS-The CID for C10	135
5.29	STLISMS-The CID for C6	136
5.30	STLISMS-The CID for C2	137
5.31	STLISMS-The CID for C4	138
6.1	Oven IBT for (R1 & R4)	148
6.2	Normal formed RBT	150
6.3	An abstract IBT example	151
6.4	Parse tree for an abstract IBT	152
6.5	An IBT for SAS	154
6.6	The parse tree for SAS IBT	154
6.7	An IBT for original requirements of microwave oven	157
6.8	The parse tree for microwave oven original IBT	158
6.9	The modified IBT for microwave oven	159
6.10	The parse tree for microwave oven modified IBT	160
6.11	BTC workflow	161
6.12	BTC output for original requirements of microwave oven	162
6.13	BTC output for modified requirements of microwave oven	162
7.1	Key elements of BERDD	171
7.2	BERDD Workflow	173
7.3	Normal formed RBT	177
7.4	Oven IBT for incompleteness defects	179
7.5	R3-RBT	179
7.6	SAS IBT for ambiguity defects	182
7.7	Oven IBT for redundancy defects	184
7.8	Oven IBT for inconsistency defects	185
7.9	RDI Workflow	187
7.10	Steps to convert IBT-XML to OWL	188
7.11	OWL meta-classes and object properties declaration	190
7.12	SAS IBT	191
7.13	XML file for SAS-R1	191
7.14	OWL representation for node R1.1	192

7.15 RBT for R8 of AIP	194
7.16 RBT for R9 of AIP	194
7.17 Incompleteness defect detection of R8 of AIP	195
7.18 Incompleteness defect detection of R9 of AI	195
7.19 AIP - ambiguity detection	196
7.20 AIP - ambiguity detection	196
7.21 AIP - redundancy detection	197
7.22 AIP - inconsistency detection	197

List of Tables

2.1	Core elements of BT notation	33
2.2	The requirements of the CAR system	34
3.1	Keyword synonyms	51
3.2	Quality assessment criteria	55
3.3	Primary studies selection data	61
3.4	RCM Challenges identified via SLR for in-house software development . .	61
3.5	RCM - In-house Challenges analysis in the context of empirical studies . .	64
3.6	RCM-GSD Challenges identified via SLR	65
3.7	RCM-GSD Challenges analysis in the context of empirical studies	66
3.8	RCM - In-house Challenges analysis based on questionnaire survey	67
3.9	RCM-GSD Challenges analysis based on questionnaire survey data	69
3.10	Chi square test results of industry data (in-house vs GSD)	71
3.11	Chi square test results of industry data (centralised vs global project structure)	72
3.12	Comparison of two data sets of RCM Challenges in in-house software development	74
3.13	Comparison of two data sets of RCM-GSD Challenges	74
3.14	Group statistics of RCM Challenges in in-house software development . .	75
3.15	Group statistics of RCM-GSD Challenges	75
3.16	Independent sample <i>t</i> -test results for RCM Challenges in in-house software development	75
3.17	Independent sample <i>t</i> -test results for RCM-GSD Challenges	76
3.18	Summary of results	77
3.19	Comparison of two data sets for RCM challenges based on software development approaches	79
4.1	Mapping between RCM-In-house challenges and RCM process outcomes .	98
4.2	Mapping between RCM-GSD challenges and RCM process outcomes . . .	99
4.3	Best practices followed for the RCM process outcomes	100
5.1	The relationship types in RCDN and their interpretation rules for indirect impact analysis	112
5.2	STLISMS functional requirements	130
6.1	Functional requirements of microwave oven	155
6.2	Modified functional requirements for microwave oven	158
6.3	Mapping between original and modified functional requirements for microwave oven	160

7.1	Requirements defects classification	175
7.2	The requirements of microwave oven-Incompleteness	179
7.3	The SAS requirements	181
7.4	Oven requirements for redundancy defects	183
7.5	Oven requirements for inconsistency defects	185
7.6	AIP Functional requirements	193

Abbreviations

BE	B ehaviour E ngineering
BT	B ehaviour T ree
CT	C omposition T ree
CID	C omponent I nterface D iagram
CFG	C ontext- F ree G rammar
CMP	C onfiguration M anagement P rocess
CIA	C hange I mpact A nalysis
DBT	D esign B ehaviour T ree
DCT	D esign C omposition T ree
GSD	G lobal S oftware D evelopment
ICT	I ntegrated C omposition T ree
IBT	I ntegrated B ehaviour T ree
RE	R equirements E ngineering
RCM	R equirements C hange M anagement
RC	R equirements C hange
RBT	R equirements B ehaviour T ree
RCT	R equirements C omposition T ree
RCDN	R equirements- C omponents D ependency N etwork
RDI	R equirements D efects I dentifier
SDLC	S oftware D evelopment L ife C ycle
SE	S oftware E ngineering
SLR	S ystematic L iterature R evue
UML	U nified M odelling L anguage
OWL	W eb O ntology L anguage

Chapter 1

Introduction

1.1 Background

Software systems are making profound changes to every aspect of human existence. Nowadays, software is everywhere and plays a fundamental and increasing role in our society. Software now drives applications in virtually all areas of human endeavours in critical roles [3]. Therefore, it is of utmost importance that the software perform correctly and predictably in all domains where it is relied on because of its critical roles. Moreover, software produces big changes in the market and has a huge impact on the way we live. Therefore, it is necessary to understand the characteristics of software and other related factors.

It is necessary to define rules and processes for the development of software systems to ensure their correct functioning, as software impacts our daily lives to a huge extent. To fulfil this purpose, the term ‘software engineering’ was first introduced in the late 1950s and early 1960s [4]. Software engineering can be defined as “the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software” [5, 6]. The process that implements software engineering principles to produce quality software is called the Software Development Life Cycle (SDLC).

Software systems are designed for a specific purpose, such as solving a problem or fulfilling the needs of stakeholders such as end-users, customers, or business [6]. The stakeholder needs are usually expressed as system features that the system should possess, and these systems features are called software requirements. Software requirements can be formally

defined as “A condition or capability that must be met or possessed by a solution or solution component to satisfy a contract, standard, specification, or other formally imposed documents” [1, 5, 6]. The software requirements are managed through the Requirements Engineering (RE) phase of the SDLC.

RE is pivotal and central to every successful software development project. According to one study, it was found that of 268 cited software development challenges, 40% were related to the RE phase of SDLC [7]. RE includes both the process of specifying requirements by studying the needs of stakeholders and the process of systematically analysing and refining those specifications [8]. Therefore, requirements development and requirements management are the two main activities performed in the RE phase of SDLC [1, 9]. Requirements development mainly addresses the requirements elicitation and specification, while requirements management deals with the management of the current and changed requirements. According to the Rational Project Management survey, conducted by IBM [10], ineffective requirements management processes have been identified as a leading cause of project failure.

In the next sections, we introduce the motivation of the whole study; we define the scope of our work and present its main contributions.

1.2 Motivation

Requirements Change Management (RCM) is one of the key components of the requirements management activities of the RE phase. Software development is a very dynamic process, and change is inevitable and persistent in software engineering. Many factors have driven these changes, such as evolving customer needs or business goals, organisational policies, working or system operating environment, and government regulations [11]. Similarly, stakeholders’ varying ways of expressing the requirements can also result in ambiguity and inconsistency in requirements, which ultimately causes the requirements change [1].

Requirements Change (RC) or requirement evolution can be formally defined as “the tendency of requirements to change over time in response to evolving needs of customers, stakeholders, organisations and work environment” [12]. As a result, RCM can be formally

defined as "managing changing requirements during the requirements engineering process, system development and system maintenance" [1, 5, 9].

It is challenging to identify and specify a complete set of requirements for a system in real-world scenarios. The software requirements keep changing through the SDLC, and approximately 50% of the initial requirements are changed before the system has been deployed [5]. The continual requirements changes and their management remains an open challenge in the field of software engineering [13]. Bano et al. [14] and Arias et al. [15] specified that effective change management is one of the main factors determining software failure or success, and annually almost 8.7% of projects fail due to problems associated with requirements changes. Similarly, a study conducted by Standish (2019) revealed that requirements change is one of the top challenges faced in software development [16]. In another study of Virtual Case file [17], the project was abandoned after the engineers have spent five years and A\$170 million due to repeated changes in the requirements specifications.

After discussing the importance of RCM and its impact on software projects success and failure, it is imperative first to identify a set of challenges related to that problem because it helps investigate a problem in more detail. By considering this face, this thesis explores key challenges related to RCM domain indifferent software development approaches. By following this, to provide a formal set of guidelines to implement RCM, an RCM process is proposed. To move on and provide more in-depth work related to the RCM problem, two highly cited challenges identified initially are chosen and proposed approaches to address them. Although all these sub-problems are related to one problem, which is RCM, however, to improve readability, we provide motivation and key limitations of the existing research related to these problems in separate subsections.

In the following subsections, we introduce these problems related to RCM, and discuss the limitations of the existing research.

1.2.1 Identifying RCM Challenges

To provide qualitative and quantitative analysis related to software artefacts, a number of approaches, including evidence-based studies and empirical studies, have been conducted. Evidence-based studies such as a Systematic Literature Review (SLR) or mapping studies

are to investigate the current research related to the studied concept and to identify the limitations of the existing research [18, 19]. Empirical studies such as controlled experiments and case studies are needed to evaluate and validate the research results. For example, Jayatilleke and Lai [20] conducted an SLR to analyse current practices related to change impact analysis, requirements traceability, and cost estimation. They also analysed existing work related to these aspects in the context of the agile software development approach for software development. Similarly, a number of empirical studies has been undertaken to analyse RCM challenges in the context of the Global Software Development (GSD) approach [21–23].

In the past, a number of SLRs have been carried out to investigate different issues of the RCM domain. However, the existing research that uses SLR to investigate RCM challenges has the following limitations:

Limitations of existing approaches: Firstly, the existing research mostly focuses on the RCM challenges in the context of GSD, which are mostly related to the management perspective of the RCM process. They generally fail to investigate the challenges related to the core RCM process, which is usually same regardless of the development approach.

Secondly, they usually did not analyse the relative importance of RCM process challenges/factors in the context of software development approaches such as in-house software development and GSD.

Thirdly, the relative importance of RCM challenges in the context of different project management structures adopted in the GSD domain is still an open question [2, 24].

Therefore, there is a need for evidence based study to identify RCM challenges not only related to the GSD domain, but also related to the core RCM process. In addition, it would also be worthwhile to investigate RCM challenges in the context of the different project management structures usually used in GSD projects. This work is related to RCM, problem 1 studied in this thesis, as shown in Figure 1.1.

1.2.2 Designing an RCM Process

To provide a formal set of guidelines to implement RCM and align literature findings with industry practices, there is a need for RCM process in the format of an ISO/IEC standard [25]. Software standards are considered one of the fundamental parts of software

engineering, especially for developing and improving software systems [26, 27]. Due to the continuing increase in the complexity and size of software projects, the use of software standards has become more important. Software life cycle processes in standards provide guidance for the individuals involved in software development projects on how to carry out a particular activity [5]. Currently, ISO/IEC 12207 “software life cycle processes” [28] is the most widely used software standard that includes the complete software life cycle processes. ISO/IEC 12207 discusses change management not as a complete process, but as a part of other software life cycle processes.

Generally, the existing software processes have the following limitations:

Limitations of existing approaches: The existing standards have not fully addressed the RCM process. For example, ISO/IEC 12207: 2017 [28] and ISO/IEC 15288: 2015 [29] outline the configuration management process and discuss change management in the context of configuration management. However, in software systems, configuration management and requirements change are two different processes [30] and covers different aspects of software development. The goal of configuration management is to maintain consistency between all system components and control the overall execution of the system. On the other hand, the goal of change management is to manage and control change in individual products of SDLC.

As a result, there is a need for an RCM process that provides a structured set of formal activities required to ensure that software is produced of the right quality, within budget and on time [3]. This work is related to problem 1 and expands the range of work related to problem 1, as shown in Figure 1.1.

1.2.3 Change Impact Analysis

To further expand the scope of our work related to RCM, one of the critical challenges, Change Impact Analysis (CIA), needs to be thoroughly investigated. CIA is one of the highly cited challenges identified through our SLR. CIA is a process for identifying system artefacts that are indirectly or directly affected by a requirements change [31].

In real-world systems, the system components are linked with each other to collectively solve any problem [32]. Even though newly introduced requirements only occupy a small

portion of the entire system, the changes could affect some other requirements; they might also impact some critical design artefacts, including the architecture [33].

For many decades, various CIA techniques have been introduced and applied at different stages of the SDLC to different artefacts, including requirements, architecture, source code, and a combination of them. However, they have the following limitations:

Limitations of existing approaches: Firstly, previous research has only provided a rough set of candidate elements that might be impacted by requirements change instead of pinpointing the elements actually impacted [34].

Secondly, they usually define system artefacts at the abstract level without specifying them in more precise modelling languages [35]. The change impacts are described in natural languages rather than more formal and precise modelling language such as Unified Modelling Language (UML) and Behaviour Tree (BT). Because of this, they usually lack the capability to illustrate the propagation of the proposed change from requirements to other software artefacts in an easy to understand and verifiable way.

Thirdly, most of the approaches have not provided any suitable measure for quantifying the change impact. They usually only provide the list of potentially impacted software artefacts without considering their complexities. The correct understanding of the complexity of software artefacts helps estimate implementation cost more objectively.

Therefore, because of those limitations, there is a need to estimate the impact of proposed requirements change on other software artefacts by using a more objective measure. It is challenging to estimate the cost of implementing the proposed change without quantifying its impact [36].

1.2.4 Detect Requirements Defects

To move on, we took another key RCM challenge identified through SLR, requirements defects detection. It is important to note that requirements defects can arise during the two activities of the RE phase of SDLC. Firstly, they can arise during requirements elicitation and analysis, a key activity performed in the RE phase of SDLC. Therefore, in this context, this is another problem (problem 2, shown in Figure 1.1) related to the RE phase of SDLC. Secondly, they can arise during RCM, and this is the key reason to

study this problem along with RCM in this thesis. In this work, we divided requirements defects detection into two sub-parts.

The first part of requirements defects detection is related to the formalisation of the translation from natural language requirements to semi-formal modelling language, i.e. BT. The key motivation for this research is that natural languages are too flexible and have number of problems, such as ambiguity. This step is before the formalisation of semi-formal languages to formal language and is important because the problems due to the flexibility of natural languages can pass on to semi-formal language and then ultimately to formal languages and cause requirements defects [37]. In the past, a number of approaches have been proposed to perform formalisation of semi-formal modelling languages to some other formal languages. However, only a few studies have been conducted to formalise the translation of natural languages to semi-formal modelling languages, and they have some limitations such as only covering a few constructs of these languages [38].

The second part of requirements defects detection is related to the formalisation of requirements defects based on formalised BT and the detection of these requirements defects. Requirements defects pose a major threat to the success of systems [39]. Requirements defects may creep in at any stage of development and are more costly to fix at the later stage of development [40, 41]. Requirements defects formalisation is important in designing formal approaches that can help to detect these defects automatically [42]. In the past, a number of formal studies have been carried out to detect requirements defects; however, the existing approaches related to requirements defects detection have the following limitations:

Limitations of existing approaches: Firstly, most of the approaches translate natural language requirements directly to formal languages such as first-order or propositional logic [43]. However, the customers who provide the requirements may find it difficult to understand formal languages, while the engineers may not have the domain knowledge to correctly interpret the customers' requirements. As a result, it is not easy to verify if the formal representation is correct [44].

Therefore, there is a need to use some semi-formal languages to bridge natural languages and formal languages. Secondly, an expert-level understanding is required to implement formal languages. Moreover, there is a need for an automated tool to reduce the dependency on expert-level understanding to use formal languages [45, 46].

Lastly, most of the existing approaches cover only inconsistency defects but miss some other common defects such as incompleteness, redundancy, and ambiguity.

1.3 Aims and Objectives

As stated in the previous section, our main research goal is to provide approaches and guidelines for RCM and requirements defects detection to assist industry practitioners and researchers. To achieve our main research goal, the relevant objectives of this thesis are stated as follows:

1. To address the limitations of existing SLR approaches; perform an SLR to analyse the current literature related to RCM challenges in the context of in-house software development and GSD. This objective is fulfilled in contribution 1 (C1), as shown in Figure 1.1.
2. To address the limitations of existing RCM related processes in ISO/IEC standards, propose an RCM process in the format of an ISO/IEC standard to facilitate industry practitioners for implementing RC. This objective is fulfilled in contribution 2 (C2), as shown in Figure 1.1.
3. To estimate requirements change impact on other software artefacts, propose an approach to perform CIA, one of the highly cited RCM challenges identified in objective 1. This objective is fulfilled in contribution 3 (C3), as shown in Figure 1.1.
4. Propose an approach to perform requirements defects detection arising during RCM, another challenge identified in objective 1. This approach also detects requirements defects that arise during requirements elicitation and analysis. This objective is fulfilled in two contributions (C4a and C4b), as shown in Figure 1.1.

1.4 Contributions of the Thesis

This thesis investigates two key problems related to the RE phase of SDLC: RCM and requirements defects detection, as shown in Figure 1.1. Two starts, we identify challenges

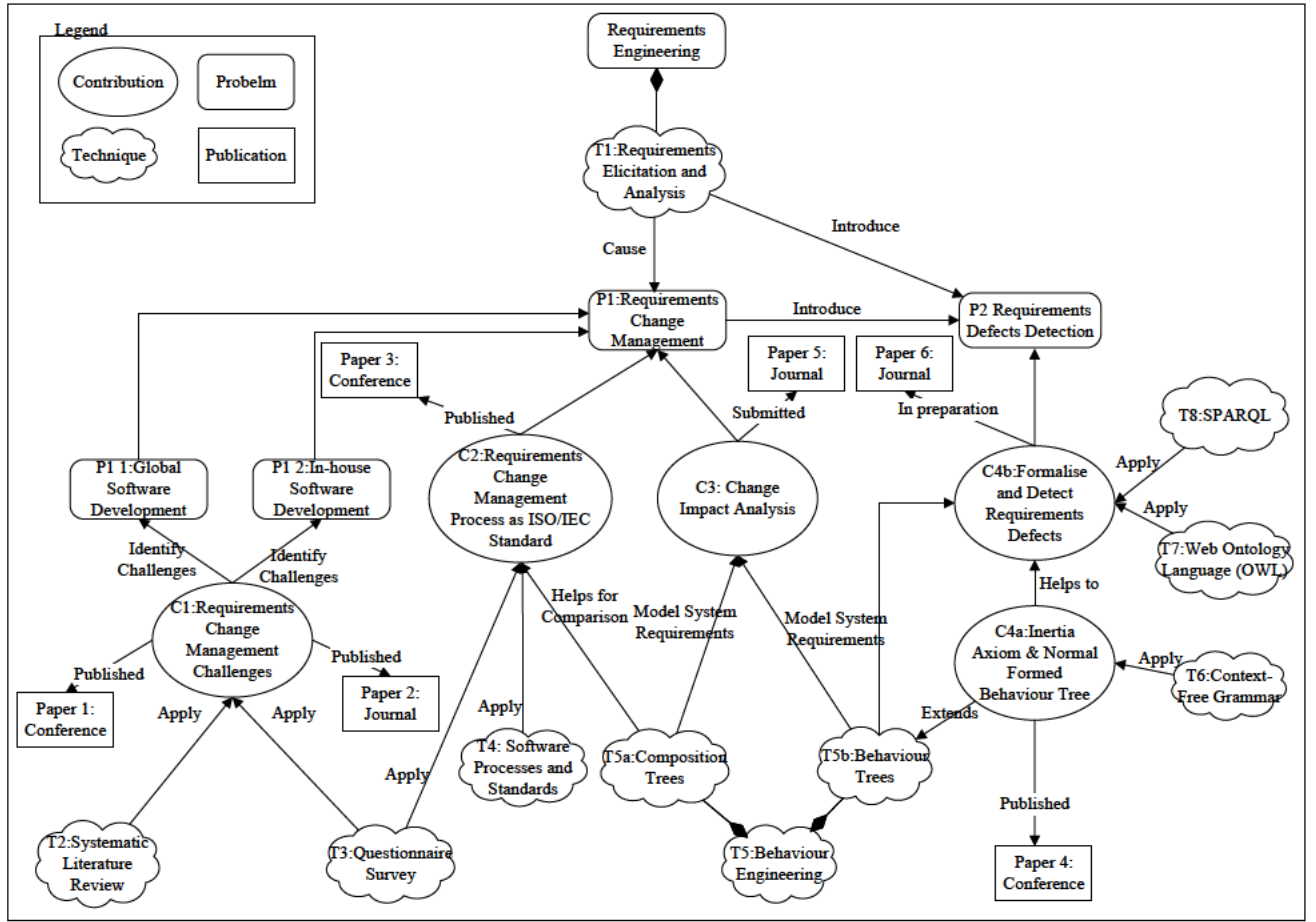


FIGURE 1.1: Contributions overview

related to RCM in in-house software development and the GSD domain, which provides an in-depth analysis of the RCM problem. And then, to provide a formal set of guidelines to implement RCM, we propose an RCM process in the format of an ISO/IEC standard. Following this, to further increase the scope of this thesis related to the RCM problem, we choose two highly cited RCM challenges: CIA and requirement defects and propose approaches to address them. Although all these sub-problems are related to one problem, which is RCM, however, to improve readability, we provide contribution details in separate subsections. An overview of these contributions is as follows:

1.4.1 C1 - RCM Challenges

In this work, we conducted an evidence-based study using SLR, to investigate the RCM challenges related to both in-house software development and GSD approaches, as shown in Figure 1.1. This work aims to identify RCM challenges and comparatively analyse RCM

challenges and development approaches. In this work, we first use SLR to identify RCM challenges related to both development approaches. Then we use a questionnaire survey to get industry practitioners' opinions about literature findings. After collecting data from both sources, we use statistical techniques such as the chi-square test to compare RCM challenges and development approaches. This comparative analysis helps in understanding the relative importance of each challenge related to a specific development approach. Furthermore, we use the chi-square test to analyse RCM challenges in the context of two prominent project management structures mostly adopted in GSD projects. This analysis helps in understanding the relative importance of each challenge in the context of a specific project management structure. In the end, based on the given analysis, we recommend some key future research directions. The detailed discussion related to contribution C1 is given in Chapter 3.

Techniques Used:

T2: SLR: SLR is one type of evidence-based studies used to investigate the current research related to the studied concept and identify future research direction. Further details of this technique is provided in subsection 2.3.1.

T3: Questionnaire survey: Questionnaire survey is one type of empirical studies used to gather opinions about a certain technology or process from a large population: A more detailed discussion related to this technique is provided in subsection 2.3.2.

Output: The following two papers are published as outputs of this contribution:

Paper 1: S. Anwer, L. Wen, and Z. Wang. "A Systematic Approach for Identifying Requirement Change Management Challenges: Preliminary Results," in *Proceedings of the Evaluation and Assessment on Software Engineering*, 230–235, 2019.

Paper 2: S. Anwer, L. Wen, Z. Wang and S. Mahmood, "Comparative Analysis of Requirement Change Management Challenges Between in-House and Global Software Development: Findings of Literature and Industry Survey" in *IEEE Access*, vol. 7, pp. 116585-116611, 2019,

1.4.2 C2 - RCM Process in the Format of an ISO/IEC Standard

The main goal of this contribution is to define an RCM process in the format of an ISO/IEC standard, which provides a set of guidelines as outcomes that ought to be followed to achieve a particular objective. In this work, we propose a novel RCM process in the format of an ISO/IEC standard and then use a composition tree to compare the

proposed RCM process with the existing configuration management process. This comparison helps in understanding the deficiencies of the existing RCM related processes in ISO/IEC standards. We also defined a seven-step RCM model to further elaborate the proposed RCM process. Moreover, to align RCM challenges identified in contribution 1 (C1) with the proposed process, we mapped RCM challenges to RCM process outcomes. Then, we used a questionnaire survey to get the opinions of industry practitioners about this mapping. The thorough analysis shows that more than 90% of the participants agreed with our developed mapping. This mapping helps industry practitioners to understand what challenges they could face while achieving a specific RCM process outcome. The detailed discussion related to this contribution is given in Chapter 4.

Techniques Used:

T4: Software Processes and Standards: Software processes in standards provide a structured set of activities to produce quality software. The detailed discussion related to software processes and standards is given in subsection 2.4.

T5a: Composition Tree (CT): CT is one type of modelling notation introduced in the Behaviour Engineering (BE) approach. CTs, similar to UML class diagrams, model static system aspects in terms of entities, relationships, attributes and component states and provide useful summary information related to these elements [47]. Further details related to CT are given in subsection 2.5.2.

T3: Questionnaire survey: This technique is already briefly discussed in contribution C1 and detailed discussion is provided in subsection 2.3.2.

Output: The following paper is published as an output of this contribution:

Paper 3: S. Anwer, L. Wen, T. Rout, Z. Wang. “Introducing Requirements Change Management Process into ISO/IEC 12207,” *in: Software Process Improvement and Capability Determination*. SPICE 2018.

1.4.3 C3 - Change Impact Analysis

To further expand the depth of our work related to RCM domain, we propose a BE-based approach for change impact analysis. The primary goal of this approach is to estimate the impact of the proposed changes on other software artefacts. In this work, first, we translate and integrate software requirements to an Integrated Behaviour Tree (IBT) and then convert the IBT into an Integrated Composition Tree (ICT). After that, we convert

the ICT to a Requirements Components Dependency Network (RCDN) to capture different relationships between requirements and components. Then, we use the RCDN and component interface diagrams retrieved from the IBT to identify which components are impacted due to the proposed change. In addition, we propose a Change Impact Indicator (CII) metric to quantify the proposed change impact. The CII helps to estimate the development cost required to implement the proposed changes and also helps to identify an optimal solution from a number of possible change proposals. A real-world system is used to demonstrate the applicability of our proposed approach. A case study verified that our approach works successfully to deliver expected results in a real-world software project. Further discussion related to this contribution is presented in Chapter 5.

Techniques Used:

T5a: Composition tree: CT is briefly discussed in contribution 2 and detailed discussion is provided in subsection 2.5.2.

T5b: Behaviour tree: BT is one type of modelling notation introduced in the BE approach. The BT is used to model the dynamic aspect of the system. It is a semi-formal, tree-like graphical structure that represents the behaviour of entities that realise or change states, make decisions, and respond to/cause events. Further details related to BT given in subsection 2.5.1.

Output: This contribution output is submitted as a journal paper.

Paper 5: S. Anwer, L. Wen, S. Zhang and Z. Wang, “BECIA: A Behaviour Engineering-based Approach for Change Impact Analysis” (Submitted to *Journal of Software: Evolution and Process*).

1.4.4 C4a & C4b - Requirements Defects Detection

To further expand the scope of our work related to RCM problem, we took another RCM challenge identified through SLR and proposed a two-step approach to detect requirements defects, including incompleteness, ambiguity, redundancy, and inconsistency. Requirements defects is one of the major factors for project failures [16].

In the first step, we propose a new axiom called the Inertia Axiom to formalise BT syntax that is subsequently named normal formed BT. After that, we use a context-free grammar technique to verify or generate the normal formed BT. A tool is developed to validate the proposed approach. This work lay the foundation for defining and formalising

requirements defects in the BT context. The detailed discussion related to this part is given in Chapter 6.

In the second step, we use the normal formed BT concept and formalise the four most common requirements defects in the BT context. To detect the formalised requirements defects, we use BT to model system requirements and then develop an algorithm to translate BTs into a formal logic language called Web Ontology Language(OWL). Moreover, as a proof of concept, we developed a tool and applied it to a real case study. This tool uses SPARQL, a semantic query language, to query the OWL knowledge base and retrieve data about requirements defects. This work helps to detect requirements defects raised during requirements elicitation & analysis and requirements change. The detailed discussion related to this part is presented in Chapter 7.

Techniques Used:

T5b: Composition tree: BT is briefly discussed in contribution 3 and detailed discussion is provided in subsection 2.5.1.

T6: Context-Free Grammar (CFG): CFG is used to define or verify the structure of a language, such as programming language, using a set of defined rules. Further discussion is provided in subsection 2.6.

T7: OWL: The OWL is a semantic web language designed to represent complex knowledge about things, groups of things, and relations between things. Further discussion is provided in subsection 2.7.1.

T8: SPARQL: SPARQL is a query language used to retrieve and manipulate data stored in Resource Description Framework (RDF) format or OWL knowledge base. Further discussion is provided in subsection 2.7.2.

Output:The first part results are published as a conference paper, and the second part work is in preparation to be as a journal paper.

Paper 4: S. Anwer, L. Wen and Z. Wang, “A Formal Model for Behaviour Trees Based on Context-Free Grammar,” *27th Asia-Pacific Software Engineering Conference (APSEC)*, Singapore, 2020, pp. 465-469.

Paper 6: S. Anwer, L. Wen and Z. Wang, “BERDD: A Behaviour Engineering based Approach for Requirements Defects Detection” (In preparation).

1.5 Thesis Outline

This thesis investigates different issues related to RCM and requirements defects, as shown in Figure 1.1. This thesis is divided into eight chapters. Following the introductory chapter, the rest of the thesis is organised as follows:

Chapter 2 is the background of the thesis. We first present an overview of the problems investigated in this thesis, such as RCM and requirements defects detection. While discussing the problems, we also present brief related work for the contributions made related to these problems in this thesis. After that, we present a brief introduction and discuss the relevance of techniques, including GSD, SLR, questionnaire survey, software processes and standards, BT, CT, context-free grammar, OWL, and SPARQL.

Chapter 3 is related to RCM challenges both in in-house software development and GSD. We used SLR to identify the key challenges of RCM and then use a questionnaire survey to get industry practitioners' feedback. We perform different analysis to understand different aspects related to these challenges in the RCM domain based on both data sets. We also recommend some future research directions based on the performed analysis.

Chapter 4 is related to the RCM process. We propose a novel RCM process in the format of an ISO/IEC standard and then use CT to compare the existing configuration management process with the proposed RCM process. We also develop a mapping between RCM challenges and RCM process outcome to understand what challenges are associated with each process outcome.

Chapter 5 is related to change impact analysis. This chapter expands the depth of RCM related work and address one of the RCM challenges identified in Chapter 3. We use different BE models to propose an approach to estimate the impact of the proposed change on other system artefacts.

Chapter 6 This chapter further expands the depth of RCM work and breadth of work related to RE. It proposes a theoretical approach for formalisation of the translation from natural language requirements to BT, which is called normal formed BT. This theoretical work laid a foundation for addressing some requirements defects, such as incompleteness, which is another challenge usually faced during RCM. We use CFG to verify and generate normal formed BT.

Chapter 7 is related to requirements defects detection. This chapter extends chapter 6 work and defines the four most common requirement defects such as incompleteness, inconsistency, redundancy, and ambiguity. To detect these defects, we translate BT into OWL and then use SPARQL query language to retrieve data related to requirements defects.

Chapter 8 finally presents a summary of the work done in this thesis, and outlines some future directions to extend this thesis.

Chapter 2

Background and Literature Review

This chapter presents the preliminaries for the essential concepts that support the work presented in this thesis. It investigates works related to RCM (problem 1), including RCM process, RCM challenges through a Systematic Literature Review (SLR), and Change Impact Analysis (CIA) in Section 2.1. Requirements defects detection related work, which is problem 2 studied in this thesis, is presented in Section 2.2.

After that, relevant techniques will be discussed briefly. Software processes and standards are introduced in Section 2.4. Behaviour Engineering (BE) concepts and notations are described in Section 2.5, while a brief introduction about Context-Free Grammar (CFG) is provided in Section 2.6. A brief overview of ontologies and languages, including Web Ontology Language (OWL) and SPARQL query language, is given in Section 2.7. Data collection techniques, including SLR and a questionnaire survey, are discussed in 2.3. Lastly, The Global Software Development (GSD) concept is discussed in Section 2.8.

It is important to note that this chapter does not provide a comprehensive study about these topics, but rather offers a brief introduction that is sufficient to understand the work presented in this thesis.

2.1 P1: Requirements Change Management

This section briefly introduces the background of Requirements Engineering (RE) and RCM, and reviews RCM work, including RCM processes, RCM challenges through SLRs, and CIA.

2.1.1 Overview of RE and RCM

Requirements engineering is the first, and one of the key phases of SDLC [5]. The requirements specification is necessary to ensure that the system performs what the adopting organisation intends, but it is also imperative to detail the scope of the implementation. Poorly written requirements specification or inconsistent understanding among different stakeholders can cause system failure and might cause human casualties. For example, in 1992, the London ambulance service suffered a disaster that brought their operations to a virtual standstill, and 20-30 people died. The detailed investigation revealed that poor requirements specification was one of the major reasons for system failure [48]. Similarly, the Queensland Health payroll system crashed just after putting in the live environment, which cost around A\$1.25 billion to the state government [49].

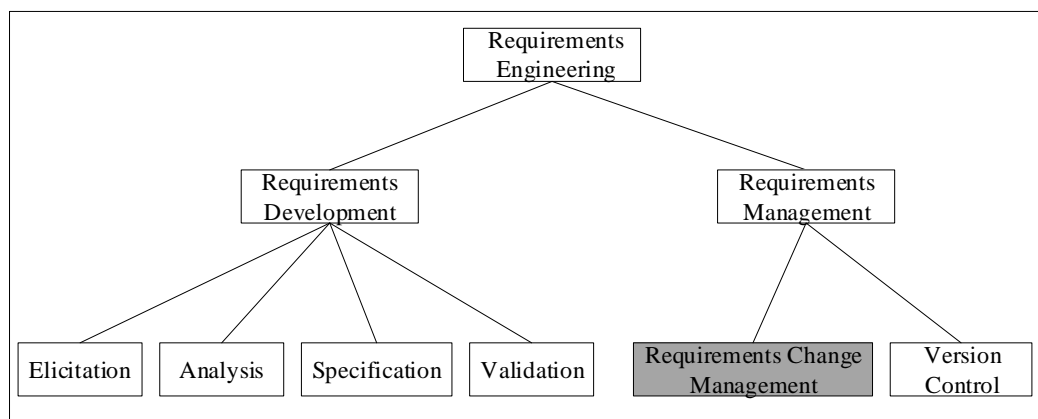


FIGURE 2.1: Overview of requirements engineering domain adapted from [1]

Requirements development and requirements management are two main activities performed in the RE phase, as shown in Figure 2.1. Requirements development deals with requirements elicitation, analysis, specification, and validation. Requirements elicitation and analysis is a process of interacting with customers and end-users to find out the

domain requirements, system functional requirements, and other constraints. Requirements engineers usually execute it, but it may also include other stakeholders, including managers, maintenance engineers, and testing engineers. It helps to gather requirements, analyse and find potential conflicts within collected requirements. However, at the same time, this process may also introduce some defects due to incorrect understanding of gathered requirements [50].

Requirements management usually deals with RCM and version control. There are some other activities usually performed in requirements management but are not listed here due to space limitation in the diagram. Our primary focus is to investigate different aspects of the RCM domain, which is shown in the shaded rectangle in Figure 2.1. Our second aim is to design an approach for requirements defects detection, which are usually raised due to RCM and requirement elicitation and analysis.

Software development is a very dynamic process, and change is inevitable and persistent in software engineering. It is challenging to identify and specify a complete set of requirements for a system in real-world scenarios. Many factors drive these changes, such as evolving customer needs or business goals, organisational policies, working or system operating environment, and government regulations. Requirements change is generally considered an undesirable event that usually creates unfavourable effects on the software development process [51]. A study conducted by Standish (2017) revealed that requirements change increased the project cost by three times and project time by two times [52]. Similarly, the estimated cost for the project *Shared Service Transformation Programme* designed by the Department of Transport in the UK [53], when approved, was £55.4 million, but due to multiple changes in the system requirements, the project cost increased to £121.2 million.

In summary, RCM is a very important activity performed throughout the SDLC, even during system maintenance. In this research, we studied different aspects of the RCM domain and designed some approaches to address some of this domain's key challenges.

After discussing the general background about RE and RCM, we briefly discuss the background of the sub-problems related to RCM that we investigate in this thesis. Firstly, we discuss SLRs and questionnaire surveys related to RCM challenges, followed by RCM related processes in the existing ISO/IEC standards and then change impact analysis, one of the key challenges for RCM. After that, we discuss requirements defects detection

as a separate problem that may arise due to requirements elicitation and analysis; however, requirements defects may also arise due to RCM. This is the reason to investigate requirements defects detection along with RCM in this thesis. The important consideration about the following subsections is that we only discuss the background of the sub-problems mentioned above and a brief literature review. A detailed literature review related to each problem and potential gaps is discussed in the corresponding chapters.

2.1.2 C1: RCM Challenges through SLR and Questionnaire Survey

Nowadays, the central role of software-intensive systems in everyday life emphasises the need for evidence-based software engineering (EBSE) [18, 19]. EBSE techniques such as SLR and mapping studies help researchers ensure that their work addresses the needs of industry practitioners and all other concerned stockholders, and they also help practitioners make rational decisions about new techniques and emerging technologies [54]. An SLR is a systematic way of surveying the literature and finding what has been done and what the shortcomings of a specific problem are. These shortcomings can be considered as future research directions related to that particular problem.

In the field of Software Engineering (SE), a number of SLRs have been conducted to study different SE concepts and to explore future research direction in the SE domain. For example, Schon et al. [55] conducted SLR to study requirements engineering literature with a focus on stakeholder and user involvement. Similarly, Hanssen et al. [56] conducted an SLR to review the application of agile methodologies in GSD projects. Furthermore, Albuquerque et al. [57] conducted a mapping study to explore best practices in implementing Requirements Changes (RC) in an agile software development context. Similarly, Batool and Inayat [58] conducted an empirical study to investigate RC best practices in the Pakistani agile-based industry.

In the domain of RCM, a number of SLRs have been conducted; for example, Jayatilleke and Lie [20] conducted an SLR to investigate existing research/literature related to RCM. It mainly focuses on the causes of changes, processes, and techniques to manage requirements change. They critically evaluate the formal and semi-formal processes related to RCM. Similarly, McGee and Greer [59], and Bano et al. [14] conducted SLRs to investigate the causes of change, and then they group them for better understanding. Moreover, some studies used empirical techniques to identify RCM challenges [23, 60].

Further to the above, a number of SLRs have been conducted to investigate RCM in the GSD paradigm. For example, Khan et al. [61, 62] used SLR and questionnaire to investigate the communications risks of the RCM process in GSD projects. They reported that, in the presence of geographical, sociocultural, and temporal differences, communication and coordination is crucial in the RCM process of globally distributed projects.

Regarding questionnaire survey, Shafiq et al. [63] presented an empirical study to explore different aspects of requirements management and requirement change management in GSD projects and proposed a specialised project management technique to handle RCM problems.

A detailed discussion about how we have conducted SLR and of the results is given in Chapter 3.

2.1.3 C2: RCM Processes

Software engineering processes in standards aim to ensure that systems are reliable and good for the environment. Software engineering processes have been proposed to standardise the development process so that software projects achieve the given objective within a specific time and budget.

Nowadays, a number of software processes as standards are used in industry to regulate the software development process. For example, ISO/IEC 12207: 2017 [28] provides a set of processes for the software life cycle. Similarly, ISO/IEC 29110:2011 [64] adapted a number of processes from ISO/IEC 12207 for very small entities. However, the existing standards did not provide a process for RCM, which is an important activity of the RE phase of SDLC. They discussed RCM as one of the activities of the configuration management process. For example, ISO/IEC 12207: 2017 [28] and ISO/IEC 15288: 2015 [29] outline the configuration management process and discuss the change management in the context of configuration management as an activity. However, in software systems, configuration management and requirements change are two different processes [30] and covers different aspects of software development. The configuration management goal is to control the overall execution of the system. In contrast, the change management process goal is to manage and control change in individual products of the project or system.

To address the limitations of the existing research, this work proposes a novel RCM process in the format of an ISO/IEC standard and theoretical model to further elaborate the RCM process.

Further discussion about the RCM process and RCM model is presented in Chapter 4.

2.1.4 C3: Change Impact Analysis

The majority of software systems are accompanied by frequent changes because they are required to keep the system operational [65]. Performing regular updates and addition of features is a necessary task when adapting to a new hardware/software or changing customer needs. However, the implementation of a single change can affect many different artefacts including design and architecture of the system. Therefore, it is necessary to understand and estimate the impact of the proposed change on other system artefacts.

CIA is a general process to identify system elements that are indirectly or directly affected by a change [31, 66]. CIA is usually studied in two types of application scenarios. Firstly, in requirements management, for a specific change request, the CIA identifies the files and models that might be affected by the requested change. Secondly, for an implemented change, the CIA role is to find the parts of the source code affected by the proposed change.

A number of CIA techniques have been proposed for different software artefacts such as requirements models [67], system design/architecture [68], and source code [69]. Similarly, a few studies have been conducted that trace back the changes from one software artifact to another artefacts. For example, from requirements to design, Sudin and Kristensen [70] presented an approach to understand how changes in requirements are carried out during the design process. From source code to software design, Hammad et al. [71] presented an approach that monitors the evolution of a design based on the changes in source code. However, both of these approaches manually identify the potentially impacted design elements based on requirements changes. In the context of requirements to source code, Ali and Lai [72] presented an approach to estimate impact in source code based on the proposed requirements change. They introduced a set of metrics to analyse the estimated impact from different perspectives.

Further discussion with more related work about the CIA is presented in Chapter 4.

2.2 P2: Requirements Defects Detection Overview

According to the Standish report (2019) [16], only 16.2% of software projects were deemed successful with all the promised functionality and completed on time and budget. There are many reasons for the failed (83.8%) projects, including lack of user involvement, poor planning, lack of executive support, and technical incompetence. In addition to these factors, defects or bugs also contribute towards project failure. Defects are defined as the deviation from the actual and expected result of system or software applications. Defects in software products show the software's inability or inefficiency to perform the expected and desired to work [73].

According to Lauesen and Vinter [74], software defects are usually classified into two groups: requirements defects and implementation defects. Implementation defects usually occur due to faulty development activities and result in program crashes or wrong results. In contrast, requirements defects usually happen due to incomplete requirements and inconsistent understanding.

The formal approaches related to requirements defects detection can be classified into two groups. Firstly, some studies first translate user requirements into semi-formal modelling language and then convert semi-formal modelling language to formal languages. Secondly, some studies translate user requirements from natural languages directly to formal languages. We opted for the first approach due to some obvious reasons. For example, by translating natural languages directly to formal languages, the customers who provide the requirements may find it difficult to understand formal languages, while the engineers may not have the domain knowledge to interpret the customers' requirements correctly [75]. In contrast, semi-formal languages usually produces graphical output, easily understandable to both customers and software engineers [76]. This approach is further divided into two parts, which are discussed in the following subsections.

2.2.1 C4a: Modelling Language (BT) Formalisation

The formalisation of translation from natural language to semi-formal Modelling languages is very vital to design formal approaches to detect requirements defects. It is also important to address natural languages inherent issues such as ambiguity. However, In

the past, most of the studies have been carried to formalise the translation from modelling languages to some formal languages.

Regarding UML to other formal languages, Khan and Porres [77] proposed a technique to check the consistency of different UML models (state and class diagrams) using OWL reasoning. They translated UML models to OWL and then performed consistency analysis using pallet and HerMiT reasoning engines. Similarly, Kaneiwa and Satoh [78] introduced an approach for checking the consistency of the class diagram by translating it into first-order logic. They introduced an optimised algorithm and restricted the basic elements of a class diagram. In another study, Mens et al. [79] implemented an approach to check the consistency between UML models (state chart diagram and sequence diagram) through using DL. They transformed the UML model to DL format using an off-the-shelf tool named Poseidon and then performed horizontal (between the same version of different models) and evolutionary (different versions of the same model) consistency checking by using a DL reasoning engine.

Similarly, a number of studies have been carried out to formalise BT to other formal languages. For example, Zafar et al. [80] developed a tool to map BT models into datalog and performed requirements early validation and analysis.

Modelling notations formalisation is very important to detect requirements defects. However, these types of approaches can detect requirements defects that can occur due to the flexibility of modelling languages and do not consider the issues that arise due to the flexibility of natural languages. The approach to formalise the translation of natural languages requirements into semi-formal modelling languages helps to detect the inherent problems of natural languages such as ambiguity, context-dependency, and incompleteness.

The detailed discussion related to formalising the translation of software requirements described in natural language to BT, with more related work, is presented in Chapter 6.

2.2.2 C4b: Requirements Defects Detection

In the past, a number of techniques have been proposed to deal with requirements defects, including defects prevention, defects detection, and minimising the impact of the defects. Defects detection and removal is the most popular approach in most organisations.

A decent amount of work has been undertaken regarding formal approaches for requirements defects detection. For example, Chen et al. [81] proposed a formal approach to check the consistency of the safety requirements of the interlocking railway system. They defined their language as SafeNL to specify safety requirements written in natural language and then automatically transformed them into formal constraints called clock constraint specification language. They used off the shelf available tools for model checking and to check for inconsistency defects. Similarly, Nenwitch et al. [82] developed a framework called Xlinkit using first-order logic to check inconsistency defects between heterogeneous requirement specifications. In another study, Weitzl et al. [83] implemented an approach by combining temporal logic and DL based ontology. They aimed to check the inconsistency of abstract level requirements documents by comparing them with the developed knowledge base.

Moreover, some studies have used Natural Language Processing (NLP) based approaches to detect requirements defects. In these approaches, firstly, the natural languages requirements are systematically translated to some other formal language using some NLP techniques and they then performed requirements defects detection. for example, Ferrari et al. [84] proposed an NLP pattern-based approach to defects requirements defects. They applied their approach to a large-scale system in the railway domain and achieved 83% precision.

Requirements defects detection with more related work is presented in Chapter 7.

2.3 Data Collection

This section briefly discusses the data collection techniques such as SLR and questionnaire survey used in this research.

2.3.1 T2: Systematic Literature Review

SLR, one type of EBSE, was originally used in the medicine domain and introduced in the field of software engineering in 2004 [54]. SLR can be defined as “A systematic literature review (often referred to as a systematic review) is a means of identifying, evaluating and

interpreting all available research relevant to a particular research question, or topic area, or phenomenon of interest.” The process of undertaking an SLR is shown in 2.2.

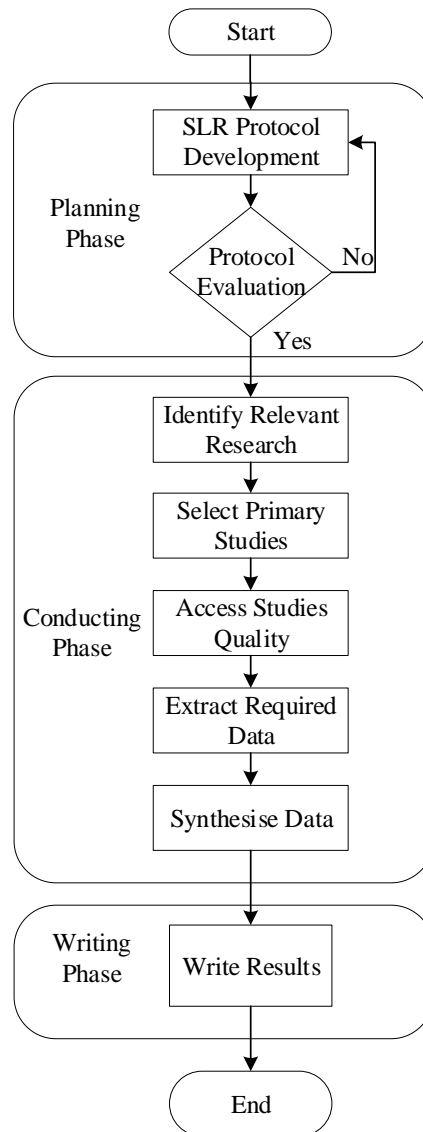


FIGURE 2.2: SLR Process

SLR is a three-phase process that involves planning, conducting, and documenting. Development of an SLR protocol and protocol evaluation are the two main activities performed in the planning phase. In the SLR protocol development activity, firstly, the need for review will be analysed, and secondly, the research questions will be specified based on the review need. Thirdly, research strings will be formalised based on the major search terms and relevant keywords. Research strings might be different for the different search

databases like IEEE, Science Direct. Lastly, the inclusion and exclusion criteria for primary studies will also be defined. Evaluation of developed protocol from independent researchers will be performed as the second activity of the planning phase.

In the conducting phase, the series of activities will include relevant research identification, primary studies selection, studies quality assessment, data extraction and data synthesis. Research strings developed in the planning phase will be executed in selected databases to find relevant studies.

In the second activity, the primary studies will be selected based on the inclusion and exclusion criteria. The primary studies selection process can be further divided into two sub-activities. In the first cycle, the studies selection will be performed based on the paper title and abstract. In the next cycle, the primary studies will be selected based on the paper's complete content. In the next activity, the selected primary studies will be accessed based on the defined quality criteria. Studies that fail to fulfil the quality threshold will be dropped from the primary studies list. Relevant data, such as challenges, factors etc., will be extracted in the fourth activity of the conducting phase. The extracted data will be peer reviewed based on some random selections from the primary studies. A manual review of primary studies will be performed to evaluate the effectiveness of the protocol and study selection process. After that, the extracted data will be further analysed and synthesised.

Lastly, the synthesised data will be analysed using statistical techniques according to the research problem in the reporting phase. In the end, the results will be summarised and published as scientific research papers.

Further details related to SLR and how we have conducted is presented in Chapter 3.

2.3.2 T3: Questionnaire Survey

The survey is an appropriate empirical research methodology tool for collecting qualitative and quantitative data from a large group of participants by using techniques such as questionnaires or interviews [85]. Surveys are conducted when the given technology or tool has already been used or before it is introduced. It is usually used to understand the situation of the current system [19].

The survey is designed and conducted by following these steps: firstly, the research objective is defined. It will help to understand the research scope and context for framing the research questions. The important point is that the research objectives must capture the survey goal. In the next step, the target audience and sampling frame are identified. The target audience is usually selected from the overall population; then, a sample is selected from the target audience. To select a target sample, random, systematic, and snowballing sampling techniques can be used.

In the third step, the survey instrument is designed. The survey instrument is generally the questionnaire and usually contains open-ended and quantitative value-based questions. Survey outcomes primarily depend on how a questionnaire has been designed. In the next step, the survey instrument is evaluated by experts and through experiments. In the last step, survey data is collected and analysed to obtain valuable outcomes.

Further details related to the questionnaire survey and how we have conducted are presented in Chapter 3.

2.4 T4: Software Processes and Standards

Software development is not always a straightforward task. A number of factors such as development team structure, market trends, technological advancements, and other situational factors affect the development process and quality of software systems [86]. To minimise the impact of these factors, the first software engineering standard was proposed in 1968, barely eight years after the term ‘software engineering’ was coined. Approximately, after two decades, in 1987, the joint ISO and IEC standard subcommittee (ISO/IEC JTC 1/SC7, or SC7) was created [46]. Standards form the fundamental building blocks for product development by establishing consistent protocols that can be universally understood and adopted.

The software standards provide a set of processes for managing the life cycle of software. For example, ISO/IEC 12207 provides 30 processes for four major groups related to software development. Software processes take certain inputs and transform them into some outputs through a defined set of activities. The software process provides a common structure so that the buyers, suppliers, developers, maintainers, operators, managers and technicians involved in the software development use a common language [87]. Software

processes help organisations to regulate their development processes, optimise development cost, increase revenue and create new business opportunities.

Sometimes, the term software development process is used synonymously with software process; however, the software process does not include activities directly related to software development like quality and configuration. Software development process definitions are usually not very formal and are left to intuition. People with formal backgrounds criticised these methods because they proved to be a major reason for failures of large and complex projects. The software process aims to manage and transform user requirements into a final software product within a specified budget and time. In a nutshell, the software process provides a roadmap to construct a software product with the required functionality and within a given budget and timeline[88].

Nowadays, GSD is widely used to gain benefit from an extended skill set and maximum use of clock hours. In the GSD environment, teams are dispersed across different countries with different time zones and cultures. Due to these differences, they may have a different understanding of different software products; however, software processes in standards helps to integrate their individual efforts at any level through a well-defined set of activities [89].

Software processes are defined in ISO software standards. Currently, many software standards are used in the industry to regulate development processes such as ISO/IEC 15288: 2015 [29], ISO/IEC 12207: 2008 [90], and 12207:2017 [28]. ISO/IEC 12207 provides a common framework for software life cycle processes that the software industry uses for software development, supply, operation, and maintenance. The software process in standards is defined as: title, purpose statement, outcomes, activities and tasks.

- Purpose: It is expressed as a high-level goal to achieve in performing the process.
- Outcomes: These usually provide the expected results but measurable and tangible from the successful performance of the process. Outcomes should express a single result. Use of the word ‘and’ or ‘and/or’ to conjoin clauses should be avoided.
- Activity: This aspect of the standard provides the list of actions that might be used to achieve the particular outcomes.
- Tasks: This aspect of the standard provides the list of actions that are required to perform a specific activity.

A detailed discussion about we use ISO/IEC standards format to propose a new RCM process is given in Chapter 4.

2.5 T5: Behaviour Engineering

Despite the improvements made in software engineering since 1968, software complexity, faulty or incomplete requirements and a need to satisfy a set of the developed software requirements still threats to software quality [91, 92]. This situation is intensified in conventional SE methods that try to build software systems that should satisfy their requirements. Such SE methods are not only complex, but also fail to deliver quality software within schedule and budget.

On the other hand, BE, first introduced in 2001 with the name of genetic software engineering [93], provides a systematic approach to designing a software system out of its requirements. Different from the more traditional SE approach, through which a software design is created to fulfil the requirements, BE retrieves a design through integrated views of the requirements [91]. BE introduces two families of graphic notations to capture software requirements from two different perspectives. The families are BTs, which capture the dynamic view of a system, and CTs, which capture the static view of a system. These two types of graphic notations will be introduced in the next subsections.

The overall behaviour engineering approach to creating a software design is shown in Figure 2.3. The problem domain contains the functional requirements of a software system expressed in natural languages. In the BE approach, each functional requirement is rigorously translated one at a time to either Requirements Behaviour Trees (RBTs) or Requirements Compositions Trees (RCTs). In the next step, all the individual trees are integrated into the Integrated behaviour Tree (IBT) or Integrated Composition Tree (ICT), which can later be transformed into the design behaviour tree. After that, the design behaviour tree can be translated into other design views such as component interaction diagram, component behaviour tree, and component interface diagram. Following this process, BE provides a clear, systematic and straightforward path from a set of functional requirements to a design that will satisfy those requirements [91].

As a formal modelling approach, the strength of BE lies in the novel way it addresses the problems of scale, complexity, and incomplete information associated with the large set

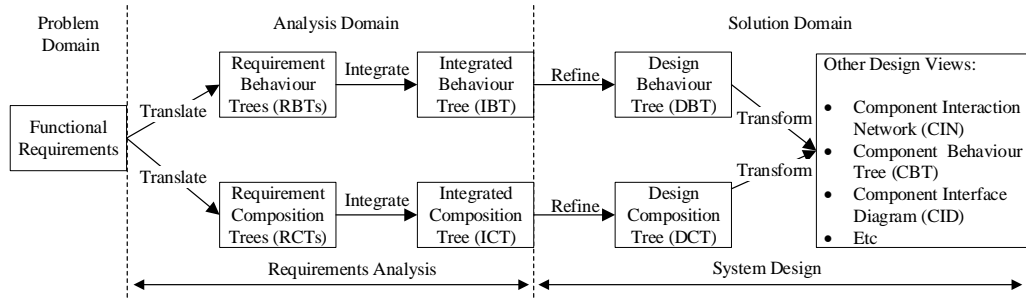


FIGURE 2.3: The behaviour engineering approach to creating a software design

of requirements needed to guide the development of challenging integrated systems [47]. While doing so, BE techniques remove ambiguity, redundancies and affect traceability. BE techniques enjoy a graphical, compositional, and structurally integrated view of the process or system being modelled [94]. This noval approach is arguably revolutionary [95]; it attracts interest in both academia and industry and has produced fruitful research results in the past two decades [96].

2.5.1 T5a: Behaviour Tree

The BT is one of the graphical notations introduced in BE. BT was created to capture and formalise dynamic information in natural language requirements specifications. BT can be formally defined as follows:

“A Behaviour Tree is a semi-formal, tree-like graphical form that represents the behaviour of individual or networks of entities which realise or change states, make decisions, respond-to/cause events, and interact by exchanging information and/or passing control” [96].

The unique tree-like syntax of BTs enables domain experts to view the flow of control easily and allows BTs to accommodate large amounts of information in network structures with a web of complex interconnections in a scalable way that would overwhelm humans [97]. In the last two decades, BT notation has gone through evolution and refinements to its current state. For example, Gonzalez-Perez, et al. [98] defined a meta-model for further understanding and clarity, especially by the object-oriented community. Winter, Grunske, and Colvin presented approaches to translate BT to other formal languages,

such as CSP [99], SAL [100], and UPPAAL [101]. Furthermore, an EBNF styled textual semantic language (BTSL) has been developed [102].

In addition to this, non-monotonic reasoning of BT was incorporated in [103]. Ahmed et al. [104] defined a semantic network to support interactive RE processes, and Colvin et al. [105] used probabilistic theory to enhance its reliability, performance, and other dependability properties. BT notation is well documented, explained, and discussed with supporting tools and successful industry case studies in [96].

Before introducing the other technical details of BT notation, we will explain how BT can be designed from natural language requirements with a simple case study. Consider the two requirements of a microwave oven case study used in already published research [106]. We have simplified the requirements description for easy understanding.

R1: *Originally, the oven is in an Idle state, and the Door is closed, and when the button is pushed, the Power-tube will be energised and the oven will start cooking.*

R2: *If the button is pushed while the oven is cooking, it will cause the oven to cook for an extra minute.*

Although we list only two oven system requirements, the objective here is to elucidate the BT modelling process with a set of requirements. The BTs are constructed in two steps, translation and integration, by using two axioms called precondition axiom and integration axiom, introduced in [91]. In the first step, each system requirement is translated into a separate BT called a RBT. For example, Figure 2.4 (a) shows the RBT for R1. For easy understanding, we have underlined the states/actions and made the **components** bold, i.e. Originally, the **OVEN** is in Idle state and the **DOOR** is closed and when the **BUTTON** is pushed, the **POWER-TUBE** will be energised and the **OVEN** will start cooking. Similarly, R2 will be translated, and the corresponding RBT is shown in Figure 2.4 (b). The directed arrows show the connection between individual nodes. In this translation, we have followed the convention of writing component names in the capital.

In the second step, the individual RBTs are integrated to get a complete BT of the system, which is called an IBT. In this step, nodes with the same behaviour in the RBTs are identified and integrated to form an IBT such as node with component name ‘OVEN’ and behaviour ‘Cooking’ used to combine RBT of R1 with RBT of R2, and an IBT based on these two requirements is shown in Figure 2.4 (c). The At sign ‘@’ is used to show the

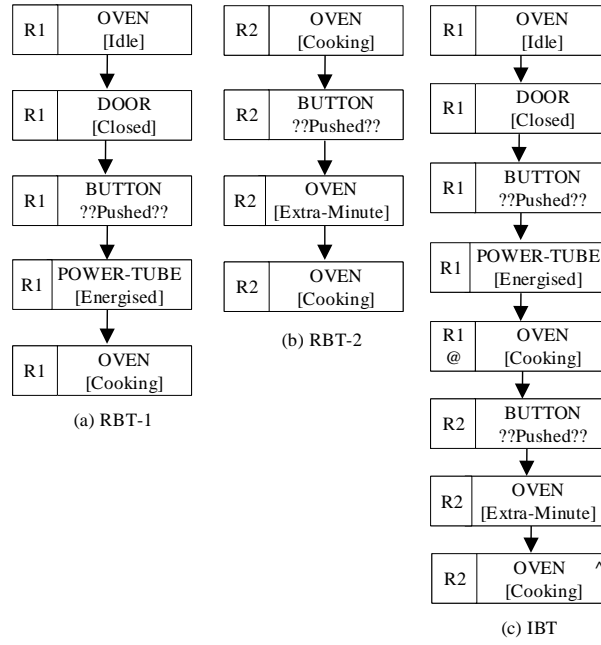


FIGURE 2.4: BT example

integration node in an IBT. Here, we explained the BT modelling process with a simple example; however, a more detailed discussion about this process can be found in [91].

After informally discussing the BT modelling process, we will now explain more technical details of BT notation. Figure 2.5 displays the attributes of a BT node. The key elements of a BT node consist of a component name and the behaviour it exhibits qualified by behaviour type. The other attributes include traceability link, traceability status, and node label as one optional parameter. The traceability link is used to associate a BT node with its corresponding system requirement, and traceability status indicates the status of that link with a set of values. The ‘+’ indicates that the BT node’s behaviour is not explicitly stated but is implied by the requirement, and ‘-’ specifies that the behaviour exhibits in the BT node are missing from the requirement.

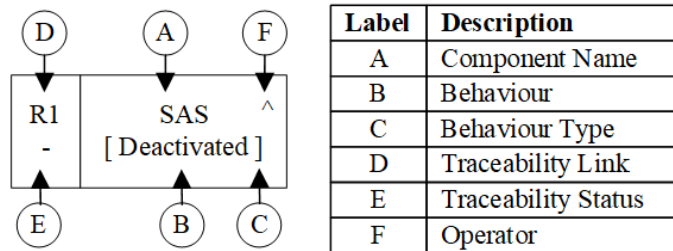
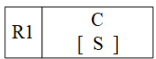
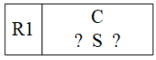
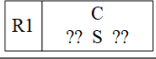
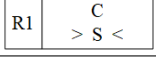
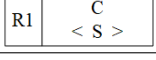



FIGURE 2.5: BT node attributes

Delimiters on both sides of the behaviour indicate the behaviour type of a BT node. The behaviour type may be a state realisation ($[\dots]$), an event ($?? \dots ??$), a guard ($??? \dots ???$), a selection ($? \dots ?$), an input ($< \dots >$), or an output ($> \dots <$). The node operators are defined in the source node, which matches a destination node with the same component, behaviour, and behaviour type. The reversion node (indicated by “ \wedge ”) indicates that the control of this node will be passed directly to its closest parent node with the same component name, behaviour name and type. The syntax and semantics of each behaviour type and the reversion node operator are shown in Table 2.1.

TABLE 2.1: Core elements of BT notation

Node	Label	Semantics
	State Realisation	This indicates that component C realises the state of S and then passes the control to its child node(s).
	Selection	This indicates that component C will pass the control to its child node(s) if C is in the state of S; otherwise, the process on this node will be terminated.
	Event	This indicates that component C will wait until it is in the state of S and then pass the control to its child node(s).
	Input	This indicates that component C will receive a message and passes the control to its child node(s).
	Output	This indicates that component C will generate a message and passes the control to the node(s) receives the message.
	Reversion	This indicates that the control of this node will be passed directly to its closest parent node with the same component name, behaviour name and type.

A further discussion about how BT is used to model system requirements is presented in Chapters 5, 6, and 7.

2.5.2 T5b: Composition Tree

Composition Tree is one of the graphical notations introduced in BE. CT is a formal graphical notation originally for modelling component-based systems [94], CTs are similar to UML class diagrams, model static system aspects in terms of entities, relationships, attributes and component states [47]. Compared to other modelling notations like UML with tens of diagrams to contend with, the created CT models are more intuitive, less ambiguous and easier to read and verify than the original natural language processes [47]. Non-experts can easily understand the CT flowchart-styled graphic notation, soft and

casual modelling. The notation has been found to be than other modelling notations to trace back and preserve the intentions of natural language processes [47].

CT, just like BT, is constructed through a careful step wise approach and later integrated into one complete tree like a graphical model. Individual RCTs are later integrated into an ICT more like a UML class diagram to encapsulate a structured view of the complete system vocabulary. Here we use a small example, which has been published in software engineering literature [47], to illustrate composition trees. Let us consider a small system, CAR, that has the following 6 requirements:

TABLE 2.2: The requirements of the CAR system

#	Requirement
R1	The car can only be started if it is in the park state when the driver inserts the key in the ignition and turns it on.
R2	A dashboard light remains on if the driver's seatbelt is not fastened when the driver is seated, and the ignition is on.
R3	If the handbrake is on when the ignition is on, the brake light turns on.
R4	The security alarm is on when the car is locked, and if anyone tries to break in by breaking a window or forcing a door the alarm will sound.
R5	When the driver, on approaching the car, presses the key-button it unlocks the door and turns the security alarm off.
R6	When the car is unlocked the driver may get in and put the car into the park state.

The first step is to translate individual requirements into RCTs. The RCT translated from requirement R1 is shown in Figure 2.6. To help readers understand how the requirement is translated into a CT, we repeat the original requirement in the figure and underline all keywords. The system is called CAR (in this example, we use all capital words to represent components and systems), expressed in a double-line box at the top of the figure. According to R1, the CAR could be in two different states called “started” and “park”, so the two states are drawn under CAR. R1 also mentions two components, “key” and “ignition,” drawn in single-line boxes under CAR. From R1, we also know that KEY could be in the state of “turned” and IGNITION could be in the state of “on”, so these states are drawn under corresponding components, respectively. We also note that KEY is inserted in IGNITION, which indicates a relationship between the two components, and this relationship is represented with a key label Relation under KEY. Similarly, we can draw other RCTs one by one and then integrate them.

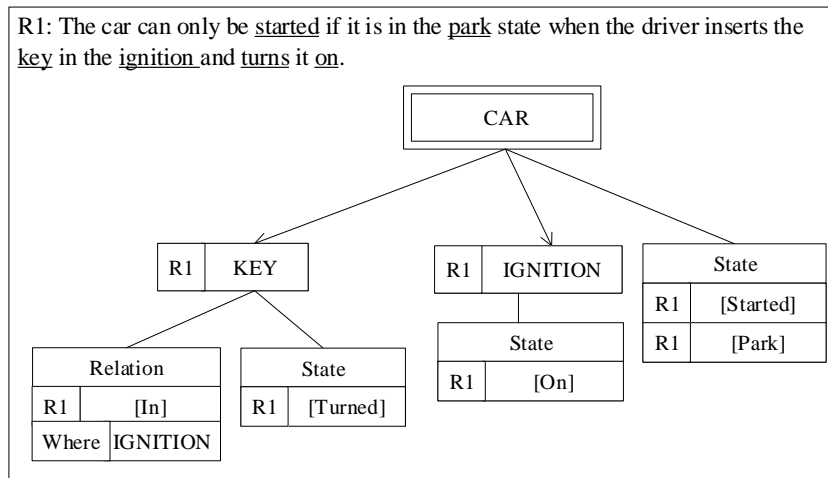


FIGURE 2.6: The IBT of R1

Figure 2.7 shows the integrated composition tree of R1 and R2. From this figure, apart from component KEY and IGNITION, three more components DLIGHT (dashboard light), SBELT (seatbelt), and SEAT (seat) are added with their corresponding states based on R2.

Figure 2.8 shows the ICT of all 6 requirements. There are 10 components under CAR and a component BUTTON under component KEY. Each state is associated with a requirement tag to trace this piece of information back to its original requirement. For example, component BUTTON has a state called “pressed”, the associated requirement tag is R5, so we can check requirement R5 to verify this piece of information.

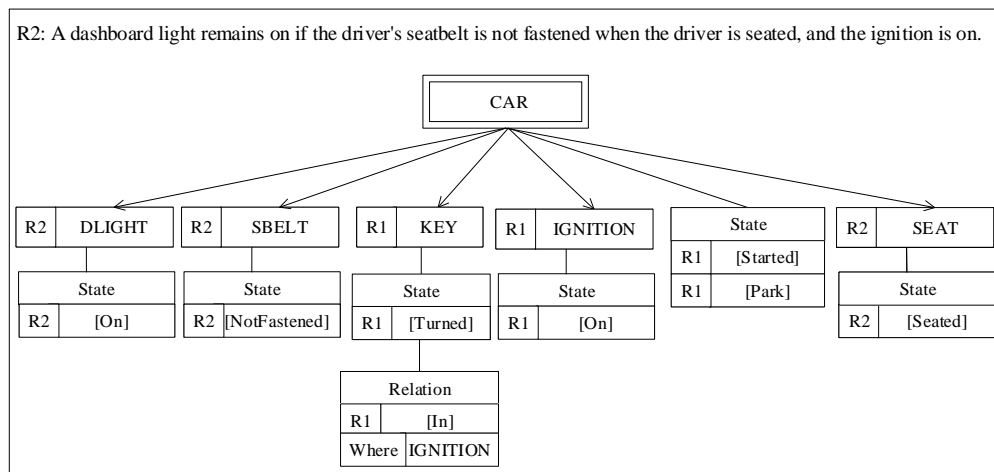


FIGURE 2.7: The IBT of R1 and R2

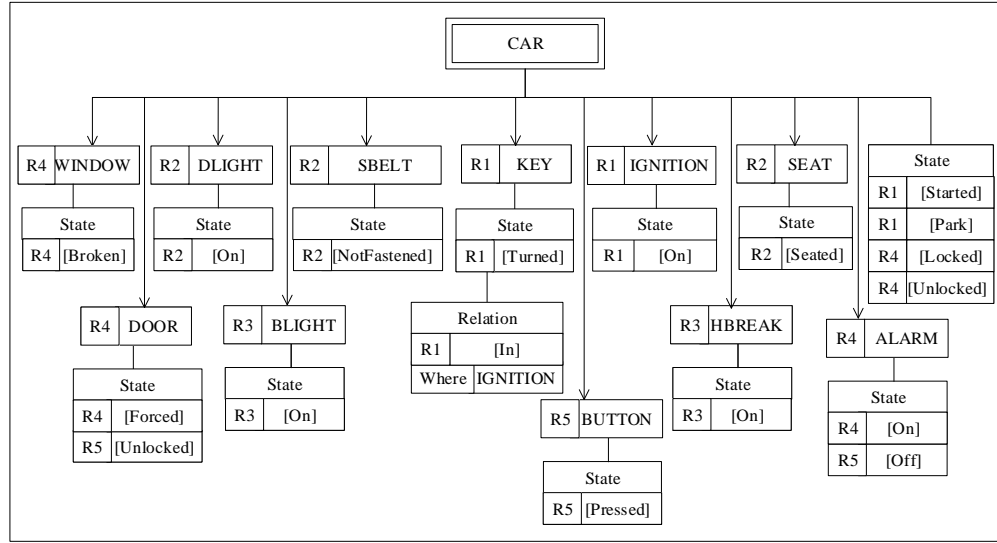


FIGURE 2.8: The IBT of CAR system

Comparing Figure 2.8 with the original set of requirements, we can see that the ICT shows the system’s component composition is much more clearly and visibly. Many questions such as what components are under the system, what states a component may have, can be easily answered by checking the diagram. The diagram can also help to identify incompleteness defects in the requirements. For example, we discover that the component DLIGHT has only one state “on”. Based on domain knowledge, we know that light should not be in the state of “on” all the time, which means we have missed some requirements to describe when the DLIGHT might be “off”. Generally, CTs plus BTs have proven to be useful tools for requirements analysis and systems design.

A further discussion about how CT is used in our research is presented in Chapters 4 and 5.

2.6 T6: Context-Free Grammar

Context-Free Grammar (CFG) is one of the formal grammars defined in the Chomsky hierarchy, as shown in Figure 2.9. CFG is less expressive than context-sensitive grammar and requires less computation effort to generate a language [107–109]. Moreover, the question of whether the given string is generated or verified by a given grammar is called a membership problem, and context-free languages are best suited for these problems. In

contrast, in natural languages where the context of letters/words within a string/sentence matters, along with the structure, context-sensitive languages become the best choice.

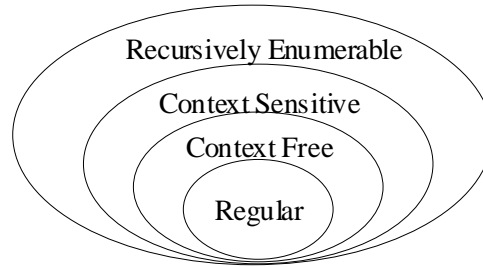


FIGURE 2.9: Chomsky hierarchy languages classification

A CFG helps define the structure of a language, such as a programming language, using a set of production rules. Alternatively, a CFG, by applying a set of production rules, can also be used to verify whether any given string belongs to some language. In general, a CFG is described by a tuple (S, N, T, P) where S , N , T , P represent start symbol, non-terminals, terminals, and production rules. Overall, a CFG generates a language L that means is the set of all sentences can be expressed using the CFG. For example, a CFG may specify the rules used to construct any sentence in the English language. If a sentence conforms to the rules specified in a CFG, it is called grammatically correct; otherwise, the sentence is ungrammatical.

This research uses CFG to formalise RBT structure and identifies requirements incompleteness defects by using that structure. We opted for CFG because, firstly, our problem relates to the membership problem, and CFG constructs are enough to verify and syntactically generate RBT structure. Secondly, a CFG provides a deterministic solution in contrast to a recursive enumerable one.

Further discussion related to CFG with more related work is presented in Chapter 6.

2.7 Ontologies and Reasoning

This section briefly discusses OWL, the relevance of OWL to this research, and the SPARQL query language used to query data from OWL ontologies.

2.7.1 T7: Web Ontology Language (OWL)

The increasing importance of ontologies and their processing in computers has led to the development of many ontology representation languages such as SHOE, OIL, DAML+OIL, RDF, RDF(S) and OWL [110]. OWL is a consolidation of its preceding languages with influence from Description Logics, frames paradigm and RDF. Its main representation and logical framework, including syntax and semantics, are based on Description Logics [110]. An ontology language is a vehicle to specify at an abstract level what is necessarily true in the domain of interest. It expresses constraints that declare what should necessarily hold in any possible concrete instantiation of the domain.

The OWL is a semantic web language designed and standardised by W3C to represent rich and complex knowledge about things, groups of things, and relations between things [111]. The first version of OWL called OWL 1 was announced in 2004. The OWL 1 language comprises three sub-languages such as OWL Lite, OWL DL, and OWL Full. OWL Lite is the least expressive, and supports classification hierarchy and simple constraint features. OWL DL provides maximum expressiveness to the users without losing computational completeness and decidability. OWL DL is so named due to its correspondence with description logic, which is a decidable variant of first-order-logic. Lastly, OWL Full provides maximum expressiveness and syntactic freedom but without computational guarantee, and for this reason, OWL DL is best suited for knowledge representation.

The current version of OWL is called OWL 2, which became a W3C standard in 2009, and it is more expressive than its predecessor OWL 1 (2004). The OWL 2 editors such as Protégé and semantic reasoners such as Pellet [112], and HermiT [113] were developed soon after the announcement of the standard. OWL 2 does not distinguish those OWL 1 sub-languages. In this research, we have used OWL 2 to represent software requirements.

OWL formalises domain knowledge by creating an ontology by defining classes and the properties of those classes. It also defines individuals and asserts properties about them and reasons about these classes and individuals to the degree permitted by its formal semantics [114]. A concrete syntax is needed to store OWL ontologies and exchange them among tools and applications in practice. The primary exchange syntax for OWL is RDF/XML, and it is supported by most of the OWL tools.

To retain upward compatibility with existing web language, OWL includes class and property features as already used by RDF and RDFS [115]. Because of all the influences on OWL and compatibility issues with other semantic web languages,

- OWL uses URI references as names just like RDF uses them. It is also usual in OWL to use qualified names such as `OWL: Thing`
- OWL presents information to ontologies which are stored as web documents written in RDF/XML.
- OWL allows RDF annotation properties to be used to attach information to classes, properties and ontologies.
- OWL uses RDF datatypes and XML schema datatypes to provides data types and data values.

The reasons for choosing OWL as the formal language to specify, analyse and reason about requirements defects in Chapter 7 are:

- Some other formal languages such as propositional logic or first-order logic can also be used for supporting software requirements analysis. However, propositional logic is less expressive and not suitable for reasoning with large data [116], while first-order logic, even though very expressive, is computationally undecidable.
- Based on the previous research [117], OWL is more suitable for creating and maintaining domain knowledge and semantics for requirements that require the concepts and relationships in the problem domain to be defined.
- Many OWL reasoners are available off the shelf [112, 113, 118].
- According to the systematic literature review [119], OWL is the most commonly used ontology language in the RE process.

We used the OWL concept in requirements defects detection, and more details about how we used it are presented in Chapter 7.

2.7.2 T8: SPARQL

SPARQL is a query language used to retrieve and manipulate data stored in RDF format. SPARQL can also be used to query the OWL knowledge base because an OWL ontology's underlying structure is a collection of triples, each consisting of a subject, a predicate, and an object. This structure is similar to the RDF graph and is stored in RDF/XML format, which is usually used to store RDF data. SPARQL expresses queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF such as OWL. SPARQL can also be used for querying required and optional graph patterns along with their conjunctions and disjunctions [120].

In this study, OWL is used to express system requirements, and different query languages such as DL and OWL can be used to query the OWL knowledge base. However, in this study, we use SPARQL because SPARQL offers many benefits such as graph pattern matching, availability of a rich set of functions for strings, numbers as compared to DL query language. SPARQL engines are also widely available and have significant adoptions [121].

Further details about how we used the SPARQL query to retrieve data related to requirements defects are presented in 7.

2.8 P1.1: Global Software Development

Global Software Development or software development outsourcing is an emerging software engineering paradigm with a focus on developing quality software at low development cost [122]. GSD can be defined as *a relationship between client and vendor organisations in which a client contracts out all or part of its software development activities to one or more vendors, who provide required services in return for agreed cost* [123]. The general overview of the GSD paradigm is shown in Figure 2.10.

2.8.1 Project Management Structures in GSD

The software development environment is continually changing because of globalisation, innovation, and market trends. The newly emerged software development paradigm



FIGURE 2.10: GSD Overview

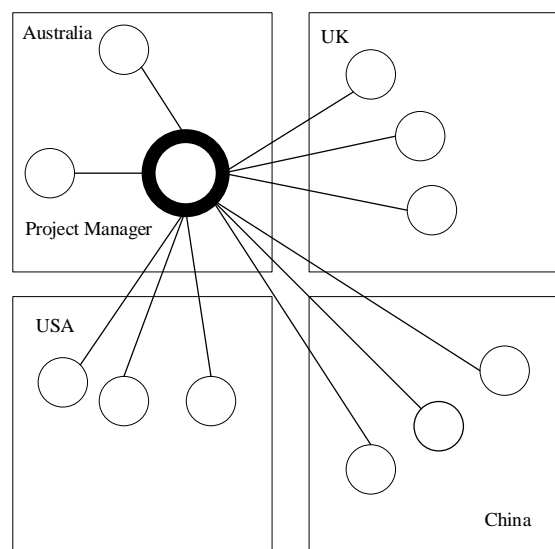


FIGURE 2.11: Centralised global project structure adapted from [2].

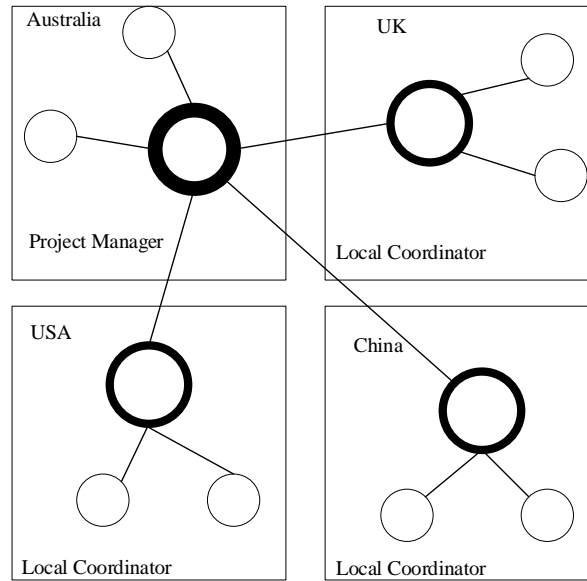


FIGURE 2.12: Distributed global project structure adapted from [2].

named GSD has the potential to reduce a project's time to market by using a highly skilled workforce at a relatively reduced cost, and by using different time zones to organise a 24/7 development model [2]. Because of these factors, the management of GSD projects becomes more challenging.

The project size, organisational structure, the maturity level of the organisation in undertaking GSD projects, and the experience of development team members working on GSD projects, dominate project structure selection from among different available projects structures. There are two main types of global project structures, namely, centralised project management structure and the distributed with local coordinators project management structure [2]. In centralised project management, as shown in 2.11, all or most of the team members report directly to a project manager who sits at one of the GSD sites and is responsible for most of the coordination and control tasks through collaborative tools. On the other hand, in distributed with a local coordinator, as shown in 2.12, the team members report directly to their local coordinators, who plan and execute the allocated task and report to the project manager at regular intervals.

In this research, we analysed identified RCM challenges in the context of the different project management structures mostly followed in the GSD paradigm.

Further details related to RCM challenges analysis based on the GSD project management structure are presented in Chapter [3](#).

Chapter 3

Requirements Change Management Challenges

Nowadays, technological advancements emphasise the need for evidence-based studies to investigate specific problems thoroughly. Evidence-based studies such as Systematic Literature Reviews (SLR) and mapping studies are the tools to collect data from published literature. They help researchers ensure that their research addresses the needs of industry practitioners and all concerned stockholders. By considering the fact that evidence-based studies are imperative to investigate a specific problem, in this chapter, we use SLR to identify the challenges associated with a Requirements Change Management (RCM) process both in in-house software development and Global Software Development (GSD) approaches. Furthermore, we use a questionnaire survey to get industry practitioners' feedback about our literature findings. The findings presented in this chapter would assist researchers and industry professionals by providing potential research directions to understand and implement RCM in different contexts more efficiently.

After identifying RCM challenges, we propose a process to provide a formal set of guidelines to implement RCM. Following this, we choose two highly cited RCM challenges identified through SLR and propose approaches to address them.

3.1 Introduction

RCM is a complex process triggered by factors such as organisational policies, market trends, and operational environments [124]. In recent years, several models have been proposed in the literature to improve RCM [72, 125, 126]. At the same time, a few reviews have been carried out to explore the different aspects of the RCM models [14, 20, 127]. However, the existing reviews have limited coverage and miss some important aspects related to software development approaches and the RCM process.

There are two major software development approaches in practice, namely in-house software development and GSD. In-house software development is carried out by a team of professionals, probably from the same country/city, with the same cultural and language background working within the same organisation [128].

A GSD project is carried out by multiple teams in various locations in the world [2, 123]. The GSD paradigm offers many benefits, including low-cost development, access to a skilled and quality workforce, and a follow-the-sun development approach [129]. However, the GSD paradigm has failed to realise the anticipated outcomes and has achieved a 45% project success rate compared to 61% for co-located teams [130]. There are many reasons for these failures, including cultural, temporal, and communication issues [131–133], particularly project management challenges across the borders.

Hence, the question arises: what are the differences and similarities between RCM in the two software development approaches? The comparative analysis between RCM challenges in in-house and GSD will assist practitioners to understand and implement RCM in different context more efficiently.

Furthermore, when people move from in-house software development to GSD, project management will become more challenging due to geographical and cultural differences [134]. To address this challenge, usually, two types of global project management structures, namely distributed (with local coordinators) structure and centralised structure [2] are used for GSD projects. To understand their impacts on RCM challenges is also interesting because doing so will help GSD practitioners to adopt or construct more suitable approaches to address these challenges.

Despite the importance of this problem, no detailed study has been found in the literature to explore the challenges associated with RCM and the two different software development

approaches. Similarly, little research has been reported to compare the impact of different project management structures in the context of RCM challenges for GSD projects. This study aims to identify and compare the challenges associated with RCM in both in-house and GSD approaches. To identify RCM related challenges, we use an SLR and then conduct a questionnaire survey with industry professionals to get feedback about SLR findings. To address the above-listed limitations, we compile the following research questions:

RQ1: What are the challenges of RCM in in-house software development as reported in the literature?

Motivation: This question provides the starting point of this study by identifying the RCM challenges related to the in-house software development approach reported in the literature.

RQ2: What are the challenges of RCM in in-house software development as identified in the industry?

Motivation: To support the findings of RQ1 and to analyse industry practices related to RCM in an in-house approach, a questionnaire survey was developed to collect data from industry professionals based on their experience.

RQ3: What are the challenges of RCM in the GSD approach, as studied in the literature?

Motivation: This question enhanced this study's scope and identified RCM challenges specifically related to GSD projects.

RQ4: What are the challenges of RCM in GSD projects as identified from industry?

Motivation: To support the findings of RQ3 and analyse industry practices related to the RCM process, a questionnaire survey was developed to collect data from industry professionals working on GSD projects based on their experience.

RQ5: What are the similarities and differences between RCM challenges in in-house software development and GSD?

Motivation: The literature hasn't discussed the relationships between RCM challenges and the two software development approaches. This research gap motivates us to tackle this question through an industry survey.

RQ6: What are the similarities and differences of RCM challenges between centralised and distributed project management structures followed in GSD projects?

Motivation: Similarly, this question hasn't been addressed in the literature; therefore, we try to find the answer through an industry survey.

RQ7: Are there any differences between the challenges identified from the literature and the industry survey?

Motivation: This question helps people realise the gaps between research and the industry by finding differences and similarities. Although the questionnaire survey was developed based on SLR results, However, there is a mismatch between both data sets results. This mismatch shows that there is a defence between academia and industry practices to investigate and implement RCM. We will apply a simple raking technique and a more advanced statistical technique, *t*-test of independence, to explore differences and similarities between both data sets.

In this work, we combined the SLR and questionnaire survey based approaches for the following reasons:

- The SLR process was used as a method for collecting RCM challenges from literature. To support our literature findings and to uncover state-of-the-art industrial practices related to RCM challenges, the survey was created to collect data from industry practitioners based on their experience.
- It is worth noting that the primary studies used in RQ3 do not show how different factors are taken into account in centralized and distributed GSD project management structures. This gap in the literature has motivated us to investigate industry practices in relation to RCM in different GSD project management structures. Hence, the questionnaire survey was used to collect data from GSD practitioners for centralized and distributed GSD project management structures.

The following part of this chapter is organised as: section 3.2 briefly presents the review of existing RCM related research. Section 3.3 describes the research methodology for this chapter. Section 3.4 presents the SLR results, and section 3.5 presents the questionnaire survey results. The comparison between SLR and questionnaire data sets is presented in section 3.6. Section 3.7 provides some discussions and implications of this work. Section 3.8 presents the limitations of this study. Finally, the conclusion is discussed in section 3.9.

3.2 Related Work

In the past, decent work has been done in the domain of RCM, including RCM models in both in-house and GSD domain, rework assessment during requirements change and empirical studies to explore different aspects of the RCM domain.

Mäkärräinen [135] proposed a requirements change process for embedded systems, and it is very similar to the spiral software development model and encompassed four cycles. In another study, Ren et al. [136] proposed prioritising critical requirements during the RCM process. Likewise, Alsanad et al. [137] developed an OWL-based domain ontology to implement requirements change in the GSD paradigm.

Regarding rework assessment, Chua and Verner [138] conducted a study to understand the effort estimation problem in the change management domain. They used case studies to validate their approach empirically. Similarly, Jayatilleke and Lai [139] investigated an approach to assess the rework required to implement a proposed change. Their approach assessed different options for a required change and suggested an option requiring less rework to implement a required change.

Regarding RCM challenges, Akbar et al. [140] proposed a readiness model for RCM in GSD context and in another study, Akbar et al. [60] conducted an empirical study to investigate the RCM challenges related to the GSD paradigm. In another study, Ahmed et al. [141] identified some challenges of RCM in GSD. However, they only used the ordinary literature review technique, which is not as systematic as SLR and they missed some relevant papers. More recently, Akbar et al. [127] conducted an SLR to investigate the success factors for RCM in the GSD domain. They found 23 success factors, including change acceptability, update requirements, information sharing. Similarly, in other studies [21, 142], conducted SLR to investigate the challenging factors that negatively impact RCM in GSD.

Despite the fact that various empirical studies to investigate the RCM problem have been reported. However, some limitations need attention.

- The existing studies have not discussed the different project management structures normally practised in GSD projects [2] and how the identified challenges can be addressed in a particular project management structure.

- The existing approach investigations mostly focused on project management-related challenges related to RCM and missed the core RCM process challenges.

3.3 Research Methodology

We use a two-step approach to conduct this research, as shown in Figure 3.1. In the first step, SLR is used to survey the literature published in the public domain and identify key challenges that impact RCM in in-house software development (RQ1) and some challenges that impact RCM in GSD (RQ3). In the second step, we use the first step results to develop a questionnaire survey and collect feedback from industry practitioners (RQ2, RQ4). After that, industry practitioners' feedback is analysed to explore different aspects of RCM challenges both in the in-house and GSD paradigms (RQ5 and RQ6). Finally, the data collected from the first two steps are compared (RQ7).

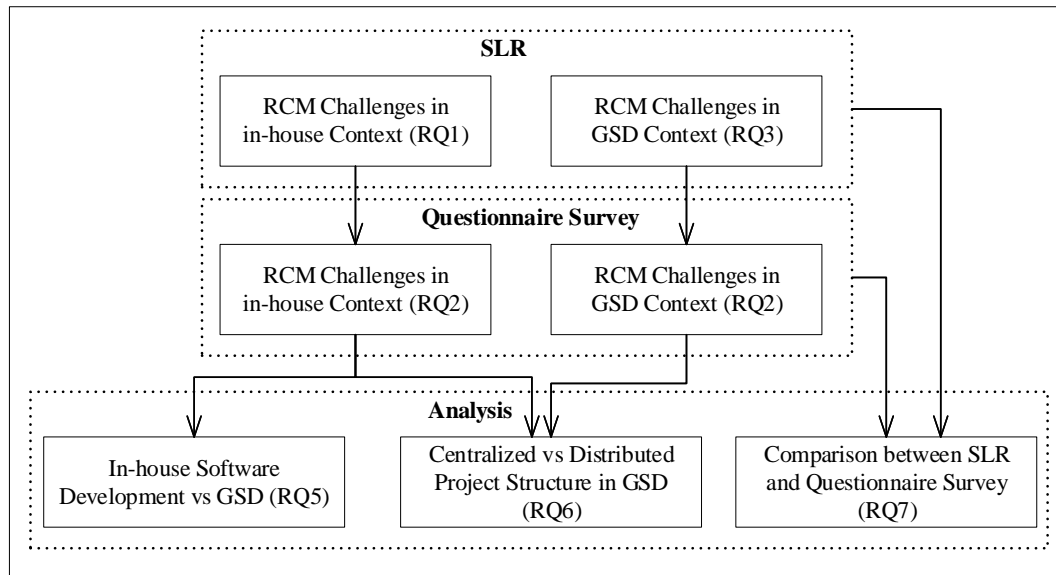


FIGURE 3.1: Research methodology

3.3.1 Data Collection via SLR

Systematic literature review is the most commonly used approach in evidence-based software engineering. SLR is formally planned and systematically executed, and it provides guidelines to identify, analyse and interpret all available evidence with reference to specific

research questions. SLR is recommended to review published literature; it helps to collect evidence and identify research gaps through a well-defined process.

In this research, we followed the Kitchenham and Charters [143] guidelines to execute an SLR process that contains three main phases: defining a protocol, conducting the protocol, and reviewing the protocol. In the first step, an SLR protocol was written to outline the complete process, and our protocol consisted of the following elements: (i) identification of research questions, (ii) search strategy, (iii) study selection, (iv) quality assessment, and (v) data extraction and synthesis. The first element was introduced in the introduction section, and the other elements are included in the following parts of this section. The SLR was undertaken by a team of three researchers, one student and two academic staff members. To reduce personal bias and improve SLR results reliability, an inter-rater reliability test (Kendall's coefficient of concordance (W)) was performed in all study selection phases.

3.3.1.1 Search Strategy

The search strategy for SLR in this study is based on the following four steps.

1. Construct search terms by identifying keywords from population, intervention, outcome and experimental design [143]. The results are:

Population: Global software development, In-house software development.

Intervention: Requirement change management challenges or barriers.

Outcome: List of challenges in the RCM of in-house and GSD projects.

Experimental design: Systematic literature review, empirical studies, expert opinion.

2. Find synonyms of keywords. Well reputed academic electronic databases are used to validate our keywords. The list of potential synonyms of each keyword is shown in Table 3.1.

3. Use boolean operators to connect major terms. In this step, we used Boolean operator OR to connect synonyms of each keyword and AND operator to connect major terms or keywords.

CHALLENGES: “Challenges” OR “problems” OR “difficulties” OR “complications” OR “obstacles” OR “barriers” OR “hurdles” OR “risks”

TABLE 3.1: Keyword synonyms

Keyword	Synonyms
Challenges	Challenges, problems, difficulties, complications, obstacles, barriers, hurdles, risks
Requirements Change Management	Requirements change, requirements volatility, requirements creep, requirements change management, requirements change difficulties, requirements change analysis, requirements change identification/type, requirements change models/processes
Global Software Development	Global software development, global project management, GSD, Offshore software development, distributed software development, offshore outsourcing global software engineering, distributed software engineering, GSE
In-house software development	In-house software development, Onshore software development, onsite software development

REQUIREMENTS CHANGE MANAGEMENT: “requirements change” OR “requirements volatility” OR “requirements creep” OR “requirements change management” OR “requirements change difficulties” OR “requirements change analysis” OR “requirements change identification/type” OR “requirements change models/processes”

GLOBAL SOFTWARE DEVELOPMENT: “global software development” OR “global project management” OR “GSD” OR “Offshore software development” OR “distributed software development” OR “offshore outsourcing” OR “global software engineering” OR “distributed software engineering” OR “GSE”

IN-HOUSE SOFTWARE DEVELOPMENT: “in-house software development” OR “onshore software development” OR “onsite software development”

By using AND operator, we defined search strings for RCM challenges in both in-house GSD approaches.

For RCM challenges in In-house software development:

“challenges” OR “problems” OR “difficulties” OR “complications” OR “obstacles” OR “barriers” OR “hurdles” OR “risks” AND

“requirements change” OR “requirements volatility” OR “requirements creep” OR “requirements change management” OR “Requirements change difficulties” OR “requirements change analysis” OR “requirements change identification/type” OR

“requirements change models/processes” AND

“in-house software development” OR onshore software development” OR “onsite software development”

For RCM challenges in the GSD paradigm:

“challenges” OR “problems” OR “difficulties” OR “complications” OR “obstacles” OR “barriers” OR “hurdles” OR “risks” AND

“requirements change” OR “requirements volatility” OR “requirements creep” OR “requirement change management” OR “requirements change difficulties” OR “requirements change analysis” OR “requirements change identification/type” OR “requirements change models/processes” AND

“global software development” OR “global project management” OR “GSD” OR “offshore software development” OR “distributed software development” OR “offshore outsourcing” OR “global software engineering” OR “distributed software engineering” OR “GSE”

4. Verify search terms in electronic databases. In this step, some papers that are relevant to our research questions were used to verify the search terms. The resources searched in this step included specific research databases, journals and conference proceedings.

Based on the available access, the following electronic academic databases were used to search relevant primary studies. Because these research sources differ in their search mechanisms, we customised the search strings listed in the previous step accordingly, which are given in Appendix A.1. It is a bit challenging to design a search string for each database that can give maximum coverage of the related papers. We designed search strings for this study with the help of two software engineering researchers who have published a good number of papers related to SLR. And then, we cross-check the search string results with a manual search. Therefore, we believe that our search strings help us to extract all the papers relevant to our studies topic.

- IEEE Xplore. <https://ieeexplore.ieee.org>
- Science Direct. <http://www.sciencedirect.com/>
- Springer Link. <http://link.springer.com/>

- ACM Digital Library. <http://dl.acm.org>
- Google Scholar. <https://scholar.google.com/>

3.3.1.2 Studies Selection

Inclusion and exclusion criteria were used to select the primary studies retrieved from the academic databases and other electronic resources. The primary studies published or available online before 30, June 2018 were included in this research. The criteria used for including and excluding the primary studies are as follows:

Inclusion criteria:

- Publications that directly linked to our research questions.
- In case of duplications, the most completed version is included.
- Publication written in English.

Exclusion criteria:

- Peer-reviewed papers only- we excluded position papers, keynotes, panel discussions, editorials etc.
- Publications written in non-English.
- Publications without bibliographic information.

The seven-step process was used to select primary studies. The number of studies selected at each step of this SLR is shown in Figure 3.2. The seven steps are as follows:

- In the first step, the search strings were executed on selected digital libraries, and 189 studies were retrieved.
- In the second step, 89 studies were selected based on the paper title and abstract. The studies that could not be decided based on their titles and abstracts were also retained for the next inspection round.
- In the next step, duplicate studies (13) were excluded.

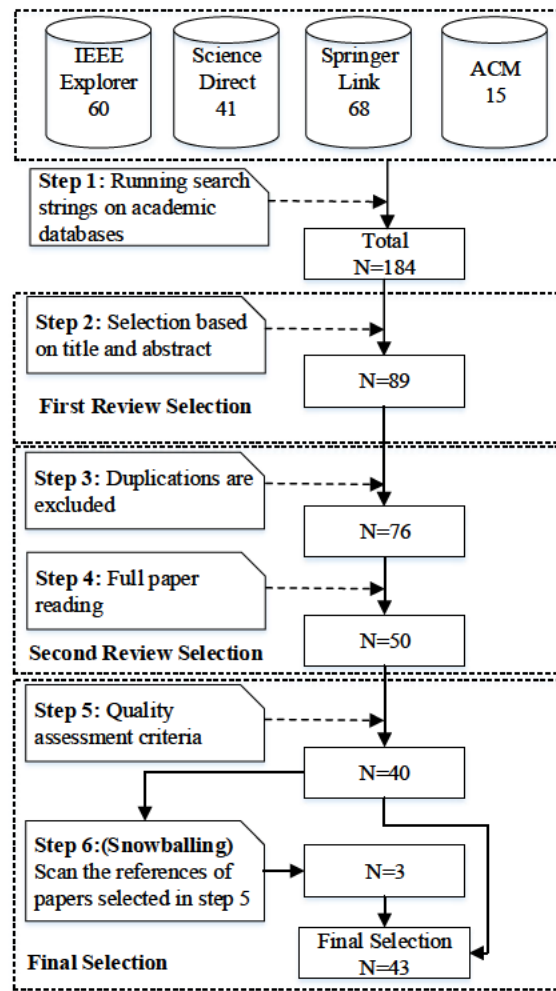


FIGURE 3.2: SLR process steps and number of studies at each step

- In the fourth step, 50 out of 76 primary studies were short-listed based on the full paper text. We only included papers that were relevant to our research questions.
- In the next step, 40 studies were short-listed based on quality assessment criteria, as shown in Table 3.2 and the papers that failed to satisfy the minimum quality score of 50% were excluded [119]. The quality assessment evaluated the credibility and relevance of primary studies. The complete list of selected papers from the SLR and their corresponding quality scores are given in Appendix A.7.
- In the sixth step, we applied the snowballing technique [144] to scan the references of the selected 40 papers to select more relevant studies. We found 15 more papers, and the same selection process was applied to them, and finally, three papers were

selected as primary studies. Unique identifiers were assigned to all papers, and these are listed in Appendix A.2.

TABLE 3.2: Quality assessment criteria

No.	Questions	Possible Answers
1	Is there a rationale for why the study was undertaken? [145]	Y=1 N=0 P=0.5
2	Are the research goals are clearly reported?[146]	Y=1 N=0 P=0.5
3	Is the proposed technique clearly described?[147]	Y=1 N=0 P=0.5
4	Are the research results clearly described?[119]	Y=1 N=0 P=0.5
5	Is there is explicit discussion about the limitations of this research?[148]	Y=1 N=0 P=0.5

3.3.1.3 Data Extraction and Synthesis

In the data extraction step (step 2), two authors extracted the data using a pre-designed data extraction form, and the third author validated the extracted data. A coding scheme based on grounded theory [149] was used to review the literature and conceptualise the RCM challenges. Although, grounded theory has been largely applied to qualitative research, however, labelled qualitative data can be processed for quantitative analysis [150]. We identified, labelled and grouped the related challenges to general categories and calculated the frequency. Furthermore, similar or related challenges were semantically compared and grouped under relevant categories.

Data Synthesis was performed, and a list of RCM challenges from the selected 43 studies was created. Initially, 18 challenges for RCM in in-house software development and additionally, 6 RCM challenges for GSD were identified, which are shown in Appendix A.3.

Three researchers carefully reviewed the identified list of challenges and tried to reduce biases and improve results validity independently. The initial list of 24 challenges was carefully reviewed and grouped into 12 main categories. The grouping of challenges was done based on the context in which those challenges were discussed in primary studies. For example, “impact analysis” and “change consequences” were grouped together in one category, as they were discussed in the same context of impact analysis.

Furthermore, to reduce researcher bias, an inter-rater reliability test was performed. In this process, three independent reviewers selected a random sample of five primary studies in the first selection round and conducted the initial selection process. Similarly, the same steps were followed in the next rounds of study selection.

We used non-parametric Kendall's coefficient of concordance (W) [151] to evaluate the inter-rater agreement between reviewers. Although grounded theory produces qualitative results but calculated frequencies for each challenge help us to perform this test. The W value ranged from 0 to 1; 1 indicates strong agreement, and 0 indicates perfect disagreement. The value of W for the randomly selected five studies from the first selection round was 0.84 ($P=0.002$). Similarly, The W value was 0.9 ($P=0.04$) and 0.95 ($P=0.03$) for the next two selection rounds, respectively. Moreover, in the snowballing process, the W value was 0.97 ($P=0.045$). These results indicate strong agreement between the findings of primary researchers and independent reviewers.

3.3.2 Data Collection via Questionnaire Survey

An empirical survey is an appropriate research methodology for collecting qualitative and quantitative data from a large group of participants by using techniques such as questionnaires or interviews [152]. In this work, we use a questionnaire survey to collect industry practitioners' opinions related to the SLR findings.

3.3.2.1 Survey Design

In this study, the target population of the survey is the software industry practitioners involved in managing GSD projects were the target population; however, it is always challenging to find a suitable population frame for a questionnaire survey [153, 154]. The population is divided into various continents. Snowballing technique [155] was used to recruit participants for this study's questionnaire survey. In our case, snowball sampling bias is not present because the population does not have privacy issues.

The snowballing sampling process was started by sending an email to personal contacts of the research team in Pakistan, China, Hong Kong, Saudi Arabia, United Arab Emirates, Australia, and the United Kingdom. The contacts were asked to send this survey to his personal contacts. Moreover, the survey link was posted on social media groups, including

Facebook, LinkedIn, Twitter. It was also posted on social media groups related to the Global software engineering conference of 2015-2018. A large number of emails are also to industry people working on GSD projects. The participants were informed that the data would only be accessible to the research team and only be used for research purposes.

3.3.2.2 Survey Instrument

In this study, we developed a questionnaire survey to get industry professionals' opinion about the RCM challenges identified through SLR. The questionnaire consisted of 3 main sections, including general information sections, RCM challenges in in-house software development and RCM challenges in GSD. Moreover, these three sections consisted of 2 open-ended questions and 21 close-ended questions. In close-ended questions, a 4-point Likert scale was used in terms of strongly agree, agree, disagree, or strongly disagree. In this work, we did not use neutral or do not know because we assume that our close-ended questions are very clear to the potential participants and they must have an opinion about it, either agree or disagree. Moreover, It is easy to discuss the finding and draw any conclusions in the absence of a neutral scale [156]. For example, one in five (20%) disagrees" correctly implies that four out of five agree. But if you have a five-point scale, you will probably have to qualify your statements in complicated ways. One in five (20%) disagrees doesn't mean the rest agrees because some of the participants may say neither agree nor disagree.

The questionnaire survey validity was assured in two steps. In the first step, face validity is confirmed by sending a draft survey to two independent researchers, one is from the software engineering domain and one is from the mathematics domain. In the second step, content validity was confirmed through a pilot study involving five professionals from different organisations. Based on their feedback, the final version of the questionnaire survey was developed, which is shown in Appendix A.4. We obtained ethical clearance from University to conduct this, which is shown in Appendix A.5.

3.3.2.3 Survey Execution and Data Pre-processing

The survey was uploaded on our personal website in 2018 and was active for a period of 4 months. We invited a total of 110 practitioners to participate in this research, and 80 of

them completed the survey, giving a response rate of 72%. The responses correctness and completeness were assured through a comprehensive review process. We use the related experience of the participants as a contingency question to filter the survey responses.

The respondents came from seven different countries: Australia, Pakistan, India, Ireland, Saudi Arabia, United Arab Emirates, and China. These respondents' organisations were involved in business intelligence, data processing, and embedded systems. The respondents' roles in their organisations ranged from software engineer to project manager with an average experience of 5 years in in-house software development and four years in GSD. The demographic information of all the participants is provided in Appendix [A.6](#).

In this survey, the data was collected in a relational database; therefore, we write simple SQL queries to perform data cleaning based on contingency questions and other factors. After doing the data cleaning, 69 out of 80 responses was usable.

The questionnaire survey consisted of three sections. The first section was designed to capture general information of participants, and this section information was stored in a relational database without any processing. Sections two and three were designed to capture participant opinions related to RCM challenges in the in-house and GSD domain. A 4-point Likert scale was used in terms of strongly agree, agree, disagree, or strongly disagree to get participants opinions. To get numeric values, we mapped strongly disagree to 1, disagree to 2, agree to 3 and strongly agree to 4, and this mapping helped us to perform statistical analysis.

3.4 SLR Results and Analysis

A total of 43 primary studies was selected from the SLR. Before discussing the SLR findings and the analysis of each research question, we give a thorough overview of the general characteristics of the primary studies .

3.4.1 Overview of the Studies

This subsection presents the general characteristics of the primary studies, including the year of publication, type of source, and research methods.

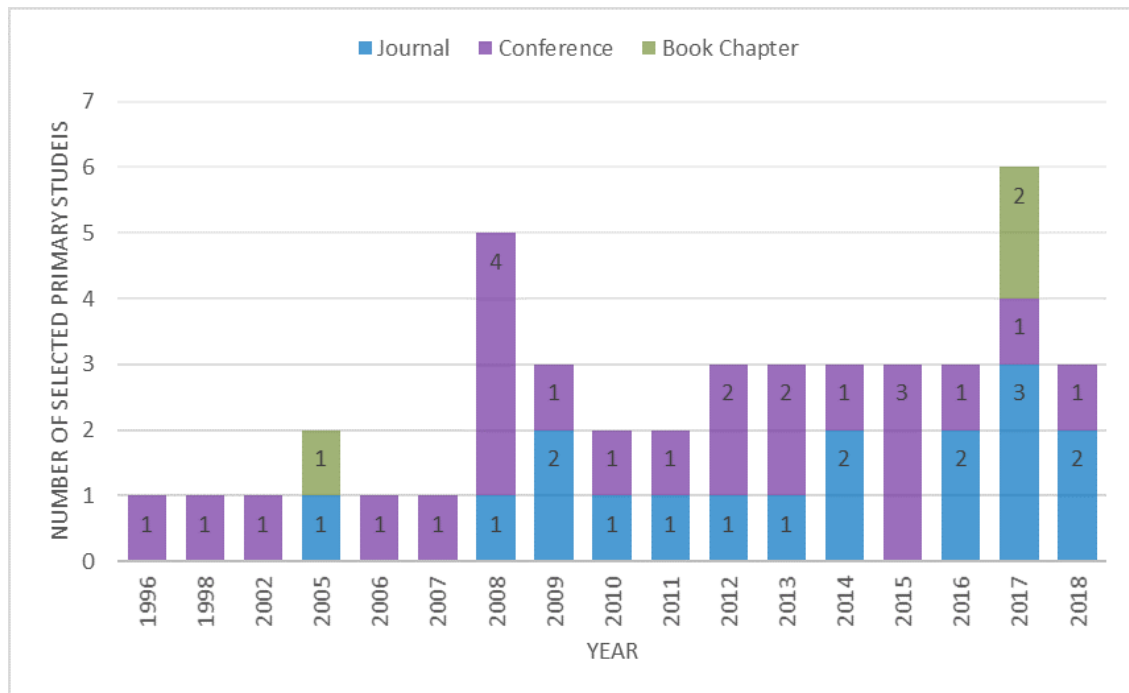


FIGURE 3.3: Number of selected studies published per year and their distribution over source type

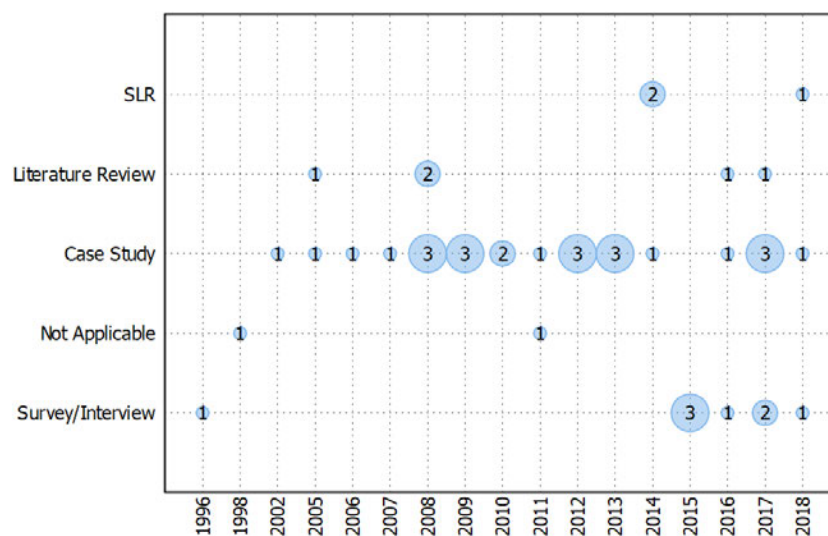


FIGURE 3.4: Bubble plot with year of publication and research method

Figure 3.3 shows the number of selected studies published per year from 1996 to 2018. In the context of publication years, it is noteworthy that research related to RCM gained attention after 2005. We could find only three papers published in this domain prior to 2005. Another worth mentioning point is that at least one journal paper was published every year after 2008 except 2015. Lastly, we found 6 studies in 2017 and 3 studies in the first six months of 2018, reflecting researchers' growing interest in the RCM domain. In the context of source type, the majority of the studies are conference papers (53%; 23 studies), followed by journal publications (40%; 17 studies), and book chapters (7%; 3 studies).

Figure 3.4 presents the distribution of published studies across empirical research methodologies. The different research strategies (literature review, case study, SLR, and survey/interview) were used in primary studies selected in this research, and these research strategies are commonly used in the empirical software engineering domain [157, 158]. The results depict that the methodology of the majority of studies was case study (63%; 27 studies), followed by survey/interviews (16%; 7 studies), literature review (14%; 6 studies), and SLR (7%; 3 studies). It is also worth mentioning that empirical investigation, through survey/interview with industry professionals, has gained attention during the past half a decade. This trend shows that academics have started appreciating the importance of industry practitioners' feedback in software engineering research. Another important point that should be emphasised is the absence of SLR because only three SLR have ever been conducted in RCM research.

The important point is that, theoretically, the primary studies in SLR should not be an SLR. However, we included three SLRs as primary studies in this SLR. The reason is that the focus of those three SLRs is different from our study. For example, Jayatilleke and Lai [20] [A20] in A.2 conducted an SLR to analyse the existing approaches related to each activity of the RCM process such as, change impact analysis, cost and time estimation, etc. In the other two SLRs, Khan et al. [61, 159] [A19 and A36] in A.2 aimed to identify and analyse communication-related challenges usually faced during RCM in the GSD domain. On the other hand, in this study, our aim was to identify RCM challenges usually faced during implementing RCM in the in-house and GSD domain.

3.4.2 SLR Findings of RCM in In-House Software Development Approach (RQ1)

This subsection discusses the SLR findings related to RQ1, in which we intend to explore the challenges that impact the RCM in in-house software development. The initial automated search resulted in 184 papers, and after a three-step selection process, we finally short-listed 43 papers shown in Table 3.3. Among them, 32 papers deal with in-house (general) RCM challenges and 11 of them with RCM in the GSD.

TABLE 3.3: Primary studies selection data

Resource	Total Results	First Review Section	Second Review Selection	Final Selection
IEEE Xplore	60	29	17	16
ACM	15	10	06	03
Science Direct	41	16	11	10
Springer	68	36	16	14
Total	184	89	50	43

TABLE 3.4: RCM Challenges identified via SLR for in-house software development

Challenge	Frequency ($n=32$)	Percentage
Impact Analysis	21	67
Cost/Time Estimation	8	25
Artefacts Documents Management	8	25
Requirements Traceability	7	22
Requirements Dependency	5	16
Requirements Consistency	4	12
Change Prioritisation	2	6
User Involvement	2	6
System Instability	1	3

In this research, we have identified 9 challenges (shown in Table 3.4) that impact RCM in in-house software development. Among the challenges, the most cited challenge for RCM in an in-house context is impact analysis (67%). In the RCM process, once the proposed change has been identified, further analysis is required to understand the consequences of the requested change on the software system [125, 160]. Bohner [31] defined impact

analysis as “the activity of identifying consequences, including the side effects and ripple effects of a change”. The impact analysis helps to understand the potential effects of requested changes before the actual change is implemented [161]. Misunderstanding of a proposed change could increase project cost or even leads to system failure.

Cost and time estimation cited by 25% of the primary studies, is usually carried out at the beginning of a project, and is a critical aspect of project management. However, the proposed change also impacts the defined project schedule and estimated cost. Cost and time estimation are collectively considered as effort estimation; however, the conversion of these aspects to each other is not a straight forward process [162, 163]. The first step of this process is to calculate the software size, which is the most crucial aspect of estimating effort impact. Different techniques, such as functional point analysis and line of code etc., can be used to calculate software size [164]. After that, man-hours are calculated based on the project size, and lastly, the number of man-hours is multiplied by an hourly rate to calculate the total effort required to implement the proposed change.

Artefact documents management is another key challenge, which is cited in 25% of the existing research. SDLC consists of several phases, and each phase output, such as specification document, and design document, is recorded as a phase product. In the RCM process, each phase product requires modification due to the proposed change to maintain consistency among all artefact documents [165]. The management of the SDLC phases product, such as requirement document, design document, source code, and testing document, is crucial, particularly if a change occurs in the late phases of SDLC, such as during testing.

Requirements traceability is another key challenge faced in the RCM process cited by 22% of primary studies. Requirements traceability can be formally defined as “the ability to describe and follow the life of a requirement in both forward, and backward direction” [166]. Traceability analysis is one of the efficient ways to understand the impact of the proposed change and is used for impact analysis [167]. Requirements traceability also helps to understand the dependency between requirements, which is another key challenge of the RCM process, and cited by 16% of the primary studies.

Requirements consistency is another key challenge that impacts the RCM process, cited by 12% of the primary studies. Consistency analysis happens during the change analysis phase, which is usually executed after change identification. Requirements consistency

can be defined in many ways, such as “not two or more requirements in a specification contradict with each other” [168], and “requirements should be understood precisely in the same way by every person who reads them” [169]. Researchers have used many techniques (including semi-formal i.e. using UML diagrams, formal i.e. first-order logic, and pure logic, to address this issue. In the process of requirements evolution, either new requirements or changes in existing requirements make requirements consistency one of the major issues [170].

Change prioritisation, cited by 6% of the primary studies, is crucial to meet the deadline and business goals. Every system requirement contributes to strategic business goals and delivers some financial value to the organisation. Change prioritisation is measured based on the urgency, impact, and risk involved with the proposed change. Prioritisation of the proposed changes is essential in RCM, mainly when strategic business goals are dependent on a given time frame [171].

6% cites that in existing research cite that user involvement is another key challenge of RCM. According to the Standish report (2014), from among 10 top-ranked software project success factors, user involvement is the top [172]. RCM requires user feedback, especially when the requested change is proposed by one of the system users. User involvement plays a critical role in successfully executing an RCM process and ultimately in project success [173]. Finally, system instability is another key challenge that impacts the RCM process and is referenced in 3% of the primary studies. A requested change can be easily handled before a system is put in the live environment; however, the RCM process becomes cumbersome when the system is already in a live environment. The key success indicator of an RCM process is the provision of uninterrupted services to the customers during the change implementation process.

Table 3.5 presents RCM challenges in in-house software development, source primary studies, and the corresponding empirical study strategies. In our study, the impact analysis is the highest cited challenge. We found 62% out of 21 papers that mentioned impact analysis as a key challenge faced in the RCM process using a case study for empirical investigation. Similarly, other empirical techniques, such as survey/interview, literature review, and SLR, were used by 14%, 19%, and 4% of the primary studies, respectively. Cost/time estimation is the second most cited challenge, and 38% of the primary studies that listed cost/time estimation as a key challenge of the RCM process used case

TABLE 3.5: RCM - In-house Challenges analysis in the context of empirical studies

Challenge	Empirical Studies Classification				Primary Studies
	Case Study	Survey/ Interview	SLR	Literature Review	
Impact Analysis	13	3	1	4	A2, A9, A10, A11, A12, A13, A14, A15, A17, A20, A22, A23, A24, A25, A30, A31, A32, A33, A34, A35, A39
Cost/Time Estimation	3	1	1	3	A2, A3, A10, A14, A20, A25, A33, A43
Artefacts Documents Management	5	1	0	2	A6, A10, A16, A23, A24, A28, A32, A43
Requirements Traceability	5	1	0	1	A1, A17, A27, A31, A33, A34, A35
Requirements Dependency	3	0	0	2	A2, A5, A9, A13, A25
Change Prioritisation	2	0	0	0	A1, A2
User Involvement	0	0	1	1	A16, A20
System Instability	0	0	1	0	A20

study and literature review for empirical investigation. Other empirical techniques, such as interview/survey and SLR, were used by 12% of the primary studies for empirical investigation.

3.4.3 SLR Findings of RCM in GSD Context (RQ3)

This subsection discusses the challenges that specifically impact RCM in GSD, as shown in Table 3.6. In the previous subsection, we have identified nine challenges, which are general RCM challenges and are related to in-house software development. In this subsection, we discuss three more challenges that are only relevant to RCM in GSD. In total, we need to consider twelve challenges while implementing proposed changes in GSD. The GSD has been increasingly used for developing software systems efficiently and effectively by capitalising on the talent pool across the world [174]. However, there are specific issues, such as time zone difference, that overshadow these benefits.

TABLE 3.6: RCM-GSD Challenges identified via SLR

Challenge	Frequency ($n=11$)	Percentage
Communication and Coordination	10	91
Knowledge Management and Sharing	8	73
Change Control Board Management	2	18

In our study, communication and coordination, cited by 91% of the primary studies, is the highest cited challenge that impacts RCM in GSD. Ineffective communication in the software development process is one of the main reasons for software project failure [175]. In GSD, communication and coordination are usually discussed in two different contexts: communication between various team members working on system development and communication between clients and development teams [176]. Geographical, cultural, and social differences make the communication and coordination process more difficult while implementing RCM in GSD projects [62].

Knowledge management and sharing, cited by 73% of the primary studies, is another key challenge that impacts RCM in GSD. In RCM, the development teams may reside in different parts of the globe and work on the same proposed change and, so accessing software artefacts with precise, accurate, and a common understating is very important. Geographical and cultural differences between development teams and clients make this process cumbersome [177]. The development teams also need to communicate and collaborate with clients who propose new requirements or modify existing requirements.

Furthermore, change control board management, cited by 18% of the primary studies, is another challenge that impacts an RCM process in GSD projects. In in-house software development, project managers and other members act as a change control board and accomplish the proposed change approval process. However, in GSD, projects are usually managed under two project management structures: centralised or distributed with local coordinators. The formation of a change control board would be different in each of them. The issues, such as who will be included in the CCB and how the CCB will work, need to be addressed for RCM to succeed in the GSD context.

Finally, Table 3.7 presents RCM challenges in GSD, their source of the primary studies, and corresponding empirical study strategies. In this research, communication and coordination is the highest cited challenge faced in RCM in GSD. We found 6 out of

10 papers used case study for empirical investigation, while 2 of the papers applied survey/interview and SLR. Knowledge management and sharing is the second most cited challenge, and 75% of the primary studies that listed knowledge management and sharing as a key challenge used case study for empirical investigation. Other empirical techniques, such as interviews/survey and SLR, were used by 12% of primary studies for empirical investigation.

TABLE 3.7: RCM-GSD Challenges analysis in the context of empirical studies

Challenge	Empirical Studies Classification				Primary Studies
	Case Study	Survey/ Interview	SLR	Literature Review	
Communication and Coordination	6	2	2	0	A4, A18, A19, A21, A26, A29, A36, A37, A41, A42
Knowledge Management and Sharing	6	1	1	0	A4, A18, A21, A36, A37, A38, A41, A42
Change Control Board Management	1	1	0	0	A21, A29

3.5 Questionnaire Survey Results and Analysis

This subsection presents the results of the questionnaire survey that we conducted to get industry practitioners' opinions regarding our SLR findings. After performing data cleaning and coding, we calculated frequency and percentage of each challenge. Frequencies were used to compare variables within and across the groups and are useful for ordinal, nominal and numeric data.

Moreover, we performed chi-square test to analyse the relative importance of RCM challenges for in-house and GSD domain, in subsection 3.5.3. Similarly, the chi-square test is used to analyse the relative importance of RCM challenges for project management structures followed in the GSD domain, in subsection 3.5.4. Chi-square test is a useful technique to analyse the association between categorical variables (i.e., whether the variables are independent or related) for given data [178, 179]. The hypothesis to interpret the results of the chi-square test are defined in respective subsections.

3.5.1 Industry Survey Findings of RCM Process (RQ2)

This subsection presents industry practitioners' opinions about RCM challenges. We received feedback from 69 industry practitioner, and a summary of the feedback is shown in Table 3.8. The participants' responses were divided into two groups: positives responses and negative responses. Positive feedback indicates that the listed challenges influence RCM, while negative feedback shows that the listed challenge has no impact on RCM. Because these challenges are common to both in-house and GSD development, we have considered all participants' observations in the frequency analysis.

TABLE 3.8: RCM - In-house Challenges analysis based on questionnaire survey

Challenge	Organisations' Observations ($n=69$)					
	Positive			Negative		
	SA	A	%	D	SD	%
Impact Analysis	23	46	100	0	0	0
Cost/Time Estimation	30	33	91	6	0	9
Requirements Traceability	16	47	91	6	0	9
System Instability	23	40	91	5	1	9
Requirements Dependency	18	43	88	8	0	12
Change Prioritisation	20	37	83	11	1	17
User Involvement	24	31	80	13	1	20
Requirements Consistency	23	31	78	15	0	22
Artefacts Documents Management	13	41	78	14	1	22

Note: Strongly Agree (SA); Agree (A); Disagree (D); Strongly Disagree (SD)

More than 90% of the respondents agreed that impact analysis, cost/time estimation, requirements traceability, and system instability are the key challenges that impact RCM process. For example, one of the participants supported his positive response for impact analysis with the following comment:

“The success/failure of an RCM process heavily depends upon the understanding of requested change impact on other baselines such as cost, time, artefacts documents and other requirements. We used number of different techniques such as cross-matrix, trees to understand the impact of proposed change.” Team Lead

Similarly, cost/time estimation is another key challenge, and received 91% of positive response from the participants. Cost and time estimations are normally used interchangeably in software engineering as a key factor that determines the project success/failure. One of the participants supported his response with the following comment:

“Cost/Time estimation is always a challenging task in software development process, and it becomes more difficult in the requirement change process. In RCM process, it is very challenging to estimate time for requested change with normally used techniques such as functional point analysis, line of code etc. Therefore, we normally use combination of different techniques to measure time for proposed change.” Project Manager

The above feedback indicates that there is need to develop a customised technique that can be used in the RCM process.

Requirements traceability is another key challenge, and 91% of respondents agreed that it impacts an RCM process. In industry, some participants considered it as a supportive element of impacts analysis. Still, most agreed that requirements traceability itself needs attention in RCM, and it helps to understand the requirement’s life and scope in both forward and backward directions.

Furthermore, system instability is another key challenge and received a 91% positive response. This challenge becomes more critical if the change request comes after the system is put in the client-side live environment. The vendor must keep the system functioning and provide uninterrupted services to the system users and other stakeholders. One of the participants supported his positive response with the following comment:

“It is very challenging for us to control the behaviour of the system during the RCM process. we normally try to implement proposed change without affecting system working, but we put system off-line if the requested change impacts key functionally or key requirements of the system.” Development Lead

Similarly, requirements dependency, change prioritisation, and user involvement received 88%, 83%, and 80% positive responses. One of the participants supported his positive response about change prioritisation with the following comments:

“It is very important to decide the implementation plan for the proposed change in the RCM process, and we usually use dependency maps to prioritise the requested changes.”
Requirements Manager

3.5.2 Industry Survey Findings of RCM in GSD Context (RQ4)

This subsection presents industry practitioners' opinions about the RCM challenges that are specific to GSD projects. The listed challenges are additional to the RCM process challenges discussed in the previous subsection as general RCM challenges. We received data from a total of 69 industry practitioners, and 45 of them were also involved in GSD projects. A summary of the data is presented in Table 3.9.

TABLE 3.9: RCM-GSD Challenges analysis based on questionnaire survey data

Challenge	Organisations' Observation (n=45)					
	Positive			Negative		
	SA	A	%	D	SD	%
Communication and Coordination	27	18	100	0	0	0
Knowledge Management and Sharing	13	31	98	1	0	2
Change Control Board Management	17	21	84	6	1	16

Note: Strongly Agree (SA); Agree (A); Disagree (D); Strongly Disagree (SD)

It is interesting to note that industry practitioners support our research findings from SLR, with two out of three challenges receiving more than 90% positive response. All the participants agreed that communication and coordination is the key challenge for RCM in the GSD paradigm. The key difference between in-house software development and GSD is the geographical or physical locations of development teams, which makes communication and coordination crucial for project success in GSD. One of the respondents supported his positive response with the following comment:

“Communication and coordination is a key success factor in GSD projects. We always try to minimise the impact of time zone and geographical difference by using various communication media such as teleconferences, instant messaging.” Project Manager

Similarly, knowledge management and sharing and change control board management received (98%) and (84%) positive response from industry practitioners. One interesting comment we received was:

“We usually struggled to share and convey a similar understanding of software artefacts between different development teams resides in a different part of the globe. We used

different cloud-based tools such as AWS cloud9 for sharing software artefacts.” Team Lead

3.5.3 Industry Survey Findings Analysis based on in-house and GSD Approach (RQ5)

This subsection discusses industry practitioners’ feedback analysis based on the two widely used software development approaches, in-house software development and GSD. In the survey, we asked about experience with RCM in both development approaches, in a demographic field. We applied the chi-square test of independence to compare the two categorical values (in-house and GSD) from a single population. The chi-square test results are shown in Table 3.10. We analysed the data based on the following hypothesis:

Null Hypothesis: There is no significant association between the identified list of RCM challenges and software development approaches.

The comparison of RCM challenges in the context of in-house software development and the GSD approach indicates more differences than similarities, as shown in Table 3.10. The P -value for impact analysis, requirements dependency, requirements traceability, and system instability is greater than 0.05, which indicates that there is no relationship between these RCM challenges and development approaches; therefore, we will accept the null hypothesis.

On the other hand, the P -value for cost/time estimation, artefacts documents management, requirements consistency, requirements prioritisation, and user involvement is less than 0.05, which indicates that these RCM challenges are different in both software development approaches; therefore in this case, we will reject the null hypothesis. The thorough analysis of industry practitioners feedback reveals that these RCM challenges require extra effort while working in the GSD context. The results show that 95% of the practitioners either strongly agreed or agreed that cost/time estimation is more challenging in GSD, while the same opinion was given by 83% of the practitioners working on RCM problems in in-house software development. Furthermore, 89% of the industry practitioners in GSD either strongly agreed or agreed that artefacts documents management is more challenging in GSD than in the in-house software development approach, where 58% of the practitioners gave the same feedback. Similarly, 89%, 90% and 89% of

TABLE 3.10: Chi square test results of industry data (in-house vs GSD)

Challenge	In-house ($n=24$)				GSD ($n=45$)				Chi-square test (linear-by-linear association) $\alpha=0.05$		
	SA	A	D	SD	SA	A	D	SD	X^2	Df	p - Value
Impact Analysis	10	14	0	0	13	32	0	0	1.952	1	0.162
Cost/Time Estimation	7	13	4	0	23	20	2	0	4.493	1	0.034
Artefacts Documents Management	4	10	9	1	9	31	5	0	4.993	1	0.025
Requirements Traceability	5	18	1	0	11	29	5	0	0.058	1	0.81
Requirements Dependency	7	12	5	0	11	31	3	0	0.387	1	0.534
Requirements Consistency	7	7	10	0	16	24	5	0	3.919	1	0.048
Change Prioritisation	6	9	8	1	14	28	3	0	5.246	1	0.022
User Involvement	6	9	8	1	18	22	5	0	5.546	1	0.019
System Instability	7	15	2	0	16	25	3	1	0.049	1	0.825

Note: Strongly Agree (SA); Agree (A); Disagree (D); Strongly Disagree (SD)

the industry practitioners either strongly agreed or agreed that requirements consistency, change prioritisation, and user involvement are more challenging while working on RCM in GSD projects, in contrast to in-house software development, where 58%, 63% and 63% of the industry practitioners gave the same opinion for these three RCM challenges.

3.5.4 Industry Survey Findings Analysis Based on Centralised and Distributed Global Project Structure (RQ6)

This subsection discusses industry practitioners' feedback related to global project management structures used for GSD projects. In the questionnaire survey, we asked for the corresponding organisational management structure (i.e. centralised or distributed) mostly followed in GSD projects, a demographic field. The gathered data reflects the practitioners' preference for centralised and distributed structured organisations. In the survey, we received data from a total of 69 participants, and 45 of them were working in GSD, 21 of them in a centralised structure, and 24 in a distributed with local coordinators project structure. We applied the chi square test of independence on the feedback of those

TABLE 3.11: Chi square test results of industry data (centralised vs global project structure)

Challenge	Centralised ($n=21$)				Distributed ($n=24$)				Chi-square test (linear-by-linear association) $\alpha=0.05$		
	SA	A	D	SD	SA	A	D	SD	X^2	Df	p – Value
Impact Analysis	8	13	0	0	5	19	0	0	1.588	1	0.208
Cost /Time Estimation	9	10	2	0	14	10	0	0	2.026	1	0.155
Artefact Documents Management	4	16	1	0	5	15	4	0	0.37	1	0.543
Requirements Traceability	5	14	2	0	6	15	3	0	0.01	1	0.919
Requirements Dependency	6	13	2	0	5	18	1	0	0.022	1	0.882
Requirements Consistency	7	12	2	0	9	12	3	0	0.004	1	0.951
Change Prioritisation	5	14	2	0	9	14	1	0	1.249	1	0.264
User Involvement	12	8	1	0	6	14	4	0	4.968	1	0.026
System Instability	8	11	2	0	8	14	1	1	0.145	1	0.703
Communication and Coordination	16	5	0	0	11	13	0	0	4.205	1	0.04
Knowledge Management and Sharing	4	17	0	0	9	14	1	0	0.931	1	0.335
Change Control Board Management	11	9	1	0	6	12	5	1	5.244	1	0.022

Note: Strongly Agree (SA); Agree (A); Disagree (D); Strongly Disagree (SD)

45 participants to compare the two categorical variables (centralised or distributed) from a single data set. The Chi square test results are shown in Table 3.11. We analysed our data based on the following hypothesis:

Null Hypothesis: There is no significant association between the identified list of RCM challenges and GSD project management structure.

A comparison between RCM challenges and project management structures indicates more similarities than differences between the two GSD project management structures. The P -value for impact analysis, cost/time estimation, requirements traceability, artefacts documents management, requirements dependency, requirements consistency, change prioritisation, system instability, and knowledge management and sharing is greater than

0.05. Therefore, we accept the null hypothesis and infer that these RCM challenges are independent of the two different project management structures.

On the other hand, the P -values for user involvement, communication and coordination, and change control board management are 0.026, 0.040, and 0.022, respectively. The P -value for user involvement, communication and coordination, and change control board management is less than 0.05, which indicates the significance of the results; therefore, we reject the null hypothesis. Of the industry practitioners who adopted a centralised project management structure, 95% strongly agreed or agreed that user involvement is more challenging than distributed project management structure, where 80% of the industry practitioners gave the same opinion.

Furthermore, 95% of the industry professionals who followed the centralised project management approach strongly agreed or agreed that change control board management is more challenging than management of distributed project structure, where 75% of the industry practitioners give the same opinion. Similarly, 100% of the industry practitioners strongly agreed or agreed that communication and coordination is challenging in both centralised and distributed project management structures followed in the GSD projects.

3.6 Comparison between SLR and Questionnaire Survey Data Sets (RQ7)

This subsection compares the two data sets retrieved from the SLR and questionnaire survey using the T-test of independence. In previous subsections, we discussed the challenges that impact the RCM process in in-house and GSD from both the published literature and the survey. In the survey, the participants were asked to give their opinion for each challenge by choosing one of the four options: strongly agree, agree, disagree, or strongly disagree. Table 3.12 presents the rank of each RCM process challenge based on the SLR and the survey results. We only take the percentage of the strongly agree option from the survey results. Furthermore, Table 3.13 shows each RCM challenge in the GSD context and the questionnaire survey results.

The comparison shows some similarities and differences between SLR and the survey results, as shown in Table 3.12. A critical analysis of two data sets shows that the

TABLE 3.12: Comparison of two data sets of RCM Challenges in in-house software development

Challenge	SLR (<i>n</i> =32)	%	Rank	SA (<i>n</i> =69)	%	Rank
Impact Analysis	21	67	1	23	33	3
Cost/Time Estimation	8	25	2	30	43	1
Artefacts Documents Management	8	25	3	13	19	9
Requirements Traceability	7	22	4	16	23	8
Requirements Dependency	5	16	5	18	26	7
Requirements Consistency	4	12	6	23	33	5
Change Prioritisation	2	6	7	20	29	6
User Involvement	2	6	8	24	35	2
System Instability	1	3	9	23	33	4

TABLE 3.13: Comparison of two data sets of RCM-GSD Challenges

Challenge	SLR (<i>n</i> =11)	%	Rank	SA (<i>n</i> =45)	%	Rank
Communication and Coordination	10	91	1	27	60	1
Knowledge Management and Sharing	8	73	2	13	29	3
Change Control Board Management	2	18	3	17	47	2

researchers and industry practitioners agree on the key challenges that impact the RCM process in in-house and GSD. Industry practitioners gave a similar response to most of the RCM challenges identified from the literature. However, artefacts documents management ranks third in the SLR data while it ranks ninth in the survey data. Furthermore, user involvement ranks 8th in the SLR data but ranks 2nd in the questionnaire data. Similarly, system instability ranked ninth in the SLR data but ranked fourth in the questionnaire data.

Furthermore, the list of challenges, particularly in GSD, received a similar response from data sets, as shown in Table 3.13. However, change control board management ranked third in the SLR data but ranked second in the questionnaire data.

We applied an independent t-test to quantify the significance of similarities between the both data sets, SLR and questionnaire survey. Our hypothesis is as follows:

Null hypothesis: The population variances of two data sets (SLR and questionnaire survey) are equal.

This study has two data sets (data from the SLR and the survey) for two categories (RCM challenges for in-house and challenges for GSD only). Therefore, we performed two different independent t-tests to compare both data sets for both categories. The descriptive statistics of the two data sets used for this study for both categories are shown in Table 3.14 and Table 3.15 respectively, whereas Table 3.16 and Table 3.17 show the independent sample t-test results.

TABLE 3.14: Group statistics of RCM Challenges in in-house software development

	Type	N	Mean	Std. Deviation	Std. Error Mean
Challenge	SLR	9	20.22	19.44	6.48
	Survey	9	30.44	7.13	2.38

TABLE 3.15: Group statistics of RCM-GSD Challenges

	Type	N	Mean	Std. Deviation	Std. Error Mean
Challenge	SLR	3	60.67	38.03	21.96
	Survey	3	45.33	15.57	8.99

TABLE 3.16: Independent sample *t*-test results for RCM Challenges in in-house software development

		Levene's Test for Equality of Variances		<i>t</i> -test for Equality of Means						
		<i>F</i>	<i>Sig.</i>	<i>t</i>	<i>df</i>	<i>Sig.</i> (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference	
									Lower	Upper
Challenge	Equal variances assumed	2.37	0.142	-1.48	16	0.158	-10.22	6.902	-24.85	4.41
	Equal variances not assumed			-1.48	10.1	0.169	-10.22	6.902	-25.58	5.13

TABLE 3.17: Independent sample *t*-test results for RCM-GSD Challenges

		Levene's Test for Equality of Variances		<i>t</i> -test for Equality of Means						
		<i>F</i>	<i>Sig.</i>	<i>t</i>	<i>df</i>	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference	
									Lower	Upper
Challenge	Equal variances assumed	3.11	0.153	0.646	4	0.553	15.33	23.73	-50.54	81.21
	Equal variances not assumed			0.646	2.65	0.570	15.33	23.73	-66.10	96.77

The *t*-test assumes that the variability of each group is approximately equal. This assumption will be verified by using Levene's test significant level. As the *P*-value of Levene's test is greater than 0.05 ($0.142 > 0.05$), this assumption is verified. Now we analyse the *p*-value of the *t*-test for equality of means against equal variance assumed, and the *P*-value of the test is 0.158 as shown in Table 3.16, which is greater than 0.05. As a result, we accept our null hypothesis and conclude that these two data sets (SLR and questionnaire survey) tend to be very close to each other, and the difference between both data sets is simply a result of statistical factors.

Similarly, the *P*-value of Levene's test for RCM-GSD challenges is greater than 0.05 ($0.153 > 0.05$): therefore, this assumption is verified. Now we analyse the *P*-value of the *t*-test for equality of means against equal variance assumed. The *P*-value of the *t*-test is 0.553, as shown in Table 3.17, which is greater than 0.05. As a result, we accept our null hypothesis and conclude that these two data sets (SLR and questionnaire survey) tend to be very close to each other. The difference between both data sets is simply a result of statistical factors.

TABLE 3.18: Summary of results

Research question	Summary of answers
RQ1: What are the challenges of Requirement Change Management process in in-house software development as commonly studied in the literature?	<ul style="list-style-type: none"> • Impact analysis
RQ3: What are the challenges of Requirement Change Management process in GSD projects, as commonly studied in the literature?	<ul style="list-style-type: none"> • Communication and coordination • Knowledge management and sharing
RQ7: Are there differences between the challenges identified from the literature and questionnaire survey?	Researches and practitioners agreed that impact analysis and cost/time estimation impact RCM process in in-house and additionally communication and coordination and knowledge management and sharing impact RCM process in GSD domain

3.7 Discussion

3.7.1 Critical RCM Challenges

In this research, we identified a list of challenges that impact RCM in both the in-house and GSD approaches. To analyse the significance of the challenges, we used the following criteria: the challenge is critical if the literature cites it with a frequency of greater than or equal to 50%, and similarly, a challenge is considered significant, if it is answered as strongly agree by more than 90% of the survey participants. Similar criteria are followed in existing research [180–182]. Table 3.18 summarises this research’s key findings based on the literature and industry feedback.

In RQ1, we identified only impact analysis as a critical challenge that impacts RCM in in-house software development. However, there are some other challenges such as cost/time estimation and artefacts documents management, which have a frequency of 25% and cannot fulfil the criticality criteria but are important for the RCM process [20]. Regarding RQ2, no RCM process challenge has a frequency greater than 90%; however, cost/time estimation has 43% positive response from industry practitioners and is important in RCM. It is also worth mentioning that user involvement received the second-highest positive response from industry practitioners compared to literature, where ranked at the 8th position. In summary, impact analysis, cost/time estimation, artefacts

documents management and user involvement are the key challenges that are important and should be managed with high priority in RCM in an in-house software development approach.

Furthermore, in RQ3, communication and coordination and knowledge management and sharing are cited in more than 50% of primary studies, and are important for RCM in the GSD context [61]. On the other hand, in the questionnaire survey (RQ4), although no challenge satisfies the criticality criteria, it is worth mentioning that change control board management received 47% of positive response and ranked 2nd compared to the literature review, where only two primary studies cited as a key challenge. In summary, all three challenges are important and should be considered while implementing RCM in the GSD context.

Regarding RQ7, the results indicate that industry practitioners are aligned with the research and reveal that impact analysis and cost/time estimation are the key challenges of the RCM process in the in-house domain. Additionally, communication and coordination and knowledge management and sharing are challenging for RCM in the GSD domain.

3.7.2 Software Development Approach-Based Analysis

In-house software development and GSD are the two most widely followed development approaches in the software development industry. It is interesting to note that the existing literature discusses the RCM process only in the context of in-house software development. There is a lack of research on how these challenges affect the RCM process in the GSD paradigm. The questionnaire-based survey presented in this research is the first attempt to address the important research gap identified in SLR.

In RQ5, we tried to understand the characteristics of different RCM challenges when they are implemented in the GSD paradigm. In the questionnaire survey, we received data from 24 participants using the in-house software development paradigm and 45 participants using the GSD paradigm. Accordingly, we compared the survey data between the two development approaches. The chi-square test results indicate that there are more differences than similarities between the RCM process development approaches. The influence of impact analysis, requirements dependency, requirements traceability, and system instability remains the same regardless of the development approach.

On the other hand, cost/time estimation, requirement consistency, change prioritisation, artefacts documents management and user involvement are more challenging in GSD than in in-house software development. We believe that this is because exchanging and synchronising information is much more difficult in GSD than in in-house software development. For example, cost/time estimation is heavily influenced by the different time zones of GSD teams, which is not the case for in-house software development. Furthermore, requirements consistency and artefacts documents management require proper coordination and precise understanding of the system requirements and other software artefacts. Therefore, it is more challenging in GSD due to cultural, geographical differences. Similarly, user involvement can be easily managed in an in-house software development paradigm where teams reside at one physical location compared to the GSD paradigm in which teams are working in different time zones.

TABLE 3.19: Comparison of two data sets for RCM challenges based on software development approaches

Challenge	In-house (n=24)	%	Rank	GSD (n=45)	%	Rank
Impact Analysis	11	46	1	13	29	6
Cost/Time Estimation	7	29	2	23	51	1
Requirements Dependency	7	29	3	11	24	7
Requirement Consistency	7	29	4	16	35	3
System Instability	7	29	5	16	35	4
Change Prioritisation	6	25	6	14	31	5
User Involvement	6	25	7	18	40	2
Requirements Traceability	5	21	8	11	24	8
Artefacts Documents Management	4	17	9	9	20	9

Moreover, we have ranked RCM challenges based on the feedback received from both types of participants, as shown in Table 3.19. It is worth mentioning that most challenges are relatively equally important and have the same rank in both development approaches, except for impact analysis, which ranked the 1st in in-house software development while ranking 6th in GSD. Similarly, user involvement ranked 7th in in-house software development, while it ranked 1st in the GSD paradigm data. In summary, these results present the relative importance of RCM challenges in both development approaches. Impact

analysis and cost/time estimation are more important in in-house software development as compared to GSD in which user involvement, cost/time estimation and requirement consistency are more important.

Furthermore, software methodologies such as lean, agile, iterative and waterfall are widely used in the software development industry. Lean and agile methodologies are more successful than iterative, and waterfall [183, 184]. This study performed a comparative analysis between RCM challenges and the most widely used software development approaches, namely in-house software development and GSD. However, there is a need to explore how RCM will be implemented, when these development methodologies (lean, agile, iterative, and waterfall) followed in in-house software development and GSD.

3.7.3 Global Project Management Structure-Based Analysis

In RQ5, we have investigated RCM process challenges and development approaches, namely in-house software development and the GSD paradigm. In RQ6, we have further analysed RCM challenges based on different project structures in GSD. The management structure of GSD projects can be either centralised or distributed with local coordinators based on the project size, complexity and other factors [2]. It is important to note that the existing literature discusses RCM for the general project management approach followed in GSD projects. There is a lack of information on how different project management structures impact these challenging factors. The industry practitioners feedback collected through our survey assists us in addressing this research gap identified in SLR.

In RQ6, the survey results indicate more similarities than differences between RCM challenges in different project management structures. Most challenges have the same impact on the RCM process regardless of management structure except for user involvement and change control board management, which are more challenging in the centralised project management structure than the distributed structure. We believe that this is due to the difference between the centralised and distributed project management structure followed in GSD projects. For example, in a distributed project structure, the user will communicate with one person at each site who works as the site coordinator and is responsible for communicating and collaborating with the project manager.

On the other hand, in a centralised project structure, all the team members working on different sites report directly to the project manager, who is solely responsible for all tasks. Therefore, all users need to communicate directly with a project manager, who may reside at a different development site with a different time zone. Furthermore, communication and coordination are equally important in both project management structures followed in GSD projects.

3.7.4 Implications of This Research

This study aims to identify RCM challenges similarities and differences in in-house software development and the GSD approach. Furthermore, we analyse RCM challenges in the context of project management structures followed in GSD. Based on these two-comparative analyses, the recommendations for researchers and practitioners are as follows:

1. In in-house software development, impact analysis, cost/time estimation, and artefacts documents management are more important, and project managers should give more attention to those challenges in RCM.
2. Impact analysis is one of the key challenges reported by the primary studies. The impact analysis helps to understand the consequences of proposed changes. Misunderstanding of proposed change consequences may increase project cost, postpone the delivery, and ultimately cause project failure. We believe that formal and semi-formal languages such as description logic and behaviour trees will assist researchers and practitioners in better understanding the requirements change and in developing tools/techniques for impact analysis.
3. Cost/time estimation is another key challenge that impacts the RCM process. Many techniques, like function point analysis and line of code, are used to calculate project time and cost at the start of the project. However, industry practitioners' data reveal that existing techniques such as function point analysis, line of code for software size and cost/time estimation are not suitable for the RCM process. Therefore, there is a need to understand and develop techniques that can be used to estimate and adjust project cost/time because of proposed requirement changes. Hence, researchers

should pay attention in developing customised techniques that can be used in RCM for cost/time adjustment and estimation.

4. Requirement consistency is another key challenge that impacts the RCM process. Requirement consistency may emerge due to changes in existing requirements or when proposing new requirements. The industry survey for this research reveals a need to develop techniques/tools using formal languages for this task. We believe that the use of formal and semi-formal languages such as description logic and behaviour trees could be a suitable solution.
5. Most of the studies did not consider system instability as a key challenge for RCM, which is contrary to industry opinion. Hence, we assert an important and urgent need for sufficient research to fully understand system instability in RCM.
6. Communication and coordination, knowledge management and sharing, and user involvement are the key challenges and must be managed with high priority in RCM in GSD projects.
7. Knowledge management and sharing are among the key challenges of the RCM process in the GSD context. The cultural, social and geographical differences make knowledge management more challenging and increase the need for suitable notation to record the requirements. As a semi-formal design notation, we believe that behaviour trees will assist practitioners in conveying a precise and clear understanding of system requirements among GSD teams.
8. The comparative analysis based on industry data between RCM challenges and software development approaches lays a foundation for future research directions. We strongly suggest that researchers should pay more attention to reporting the RCM process in the context of different software development approaches.
9. The comparative analysis between RCM challenges and project structures lays a foundation for GSD practitioners to make a better choice of project structure based on project size and the nature of a project. The GSD practitioners should consider user involvement and change control board management while using a centralised project structure in GSD projects. This analysis also asserts GSD researchers should report the RCM process in the context of these two project management structures in GSD projects.

10. The rank-based analysis lays a foundation for future research. It will help researchers focus and direct their research in RCM domains, for example, develop techniques for higher-ranked challenges in in-house software development and GSD.

3.8 Limitations

We applied an SLR to identify key challenges that impact the RCM process in in-house and GSD approaches. One limitation of SLR is incompleteness. The results depend upon the keywords used for key terms and publication databases (Science Direct, IEEE explorer, Springer Link, and ACM) to find primary studies relevant to our research questions.

However, we mitigated this risk of incompleteness in the search terms by using alternative synonyms to build search stings. Furthermore, with the increasing number of publications related to this topic, we may have missed some recent publications at the time of consolidating the SLR results.

Another possible limitation of SLR is the frequency calculation of identified challenges. We calculated each challenge's frequency using a grounded theory-based coding scheme, which provides an analytical approach to identifying, labelling, and grouping related challenges into one category. We used inter-rater reliability tests to reduce the impact of researchers bias. Nevertheless, we believe that our presented results are comprehensive and cover most of the relevant published literature.

Regarding questionnaire surveys, one possible limitation is that some participants may lack experience in responding to survey questions. In our study, we tried to choose participants who had either a higher degree in computer science or related fields and experience related to industry projects' requirement management to mitigate this risk. Another potential limitation of the questionnaire-based study is ambiguity in the survey questions.

To minimise this limitation, I was always available on Skype and by email during the study to clarify any potential ambiguities. This paper used Chi-square test and t-test of independence to either reject or accept the hypothesis. Furthermore, to mitigate any construct validity threat, we used a standard scale in the survey design that is largely used in reported research [185].

Another potential limitation of questionnaire-based studies lies in their external validity. This limitation is mainly due to the low participation rate and difficulty in choosing a true random sample.

We address this limitation by using LinkedIn, mailing list, and industrial contacts, and use the snowballing technique to engage more participants in this study [154]. Finally, we managed to receive 69 usable responses. The response rate is similar to other questionnaire-based studies reported by the software engineering community [186]. However, as indicated by Lethbridge et al. [85], questionnaire-based surveys with low participants rates can be used to understand trends. Therefore, we believe that our results will assist industry practitioners in making rational decisions during the RCM process.

3.9 Conclusion

The different aspects of RCM challenges have been explored in the existing research; however, most previous researchers have not addressed the difference between in-house software development and GSD.

In this study, we used an SLR to find out the RCM challenges related to in-house software development and GSD. Through SLR, we have identified 9 challenges related to RCM execution in in-house software development and 3 additional challenges that are specific to RCM in GSD projects.

To get industry practitioners' opinions related to our literature findings, a questionnaire survey was conducted. The chi-square based analysis of survey data indicates that there are four out of nine challenges, namely impact analysis, requirement traceability, requirement dependency, and system instability having the same impact in both in-house and GSD approaches. On the other hand, cost/time estimation, artefacts documents management, user involvement, requirement consistency, and requirement prioritisation need more attention when implemented in the GSD paradigm. Furthermore, regarding two important project management structures in GSD, centralised project structure and distributed project structure, the survey results reveal that all challenges have the same impact except user involvement and change control board management, which are more important in the centralized project structure.

The t-test of independence was used to compare both data sets. Thorough analysis reveals that research results and industry opinions are consistent regarding RCM challenges.

Chapter 4

Requirements Change Management Process in the Format of an ISO/IEC Standard

After identifying the RCM challenges through evidence based studies, there is a need to define a process to implement RCM and correlate the identified challenges with the process outcomes. The key motivation behind this work is that, software life cycle processes in ISO/IEC standards provide a formal set of guidelines to execute software development activities and help organisations to regulate their development processes to produce good quality software [25]. Furthermore, software processes provides a criteria to measure the success or failure of software system through well-defined guidelines [187].

By considering the fact that software life cycle processes in standards are imperative to produce good quality software, in this chapter, we propose a novel Requirements Change Management (RCM) process in the format of an ISO/IEC standard. After that, we proposed a theoretical model by considering the outcomes of of this process. Following this, Composition Trees (CTs) are used to compare our proposed process with the requirements change-related processes defined in existing standards. Finally, we map between RCM process outcomes and RCM challenges that have been identified by our research and introduced in Chapter 3.

By proposing a novel RCM process, this work expands the breadth of our work related to the RCM problem.

4.1 Introduction

In recent years, a number of RCM models have been proposed in the literature. There are also ISO/IEC standards proposed and practised in industry, but those standards have not fully addressed the problems of RCM. ISO/IEC 12207: 2008 [90] discusses some aspects of RCM, such as the requirements analysis activity of RCM.

ISO/IEC 12207: 2017 [28] and ISO/IEC 15288: 2015 [29] outline the Configuration Management Process (CMP) and discuss the change management in the context of configuration management as one activity. However, in Software Development Life Cycles (SDLC), configuration management and requirements change management are two different processes [30] and covers different aspects of SDLC. The goal of CMP is to maintain consistency between all components of the system and control the overall execution of the system. In contrast, the goal of the change management process goal is to manage and control change in individual products of the project or system.

To overcome the above-discussed limitation, we propose an RCM process in the format of an ISO/IEC standard and the theoretical model that addresses the deficiencies identified in existing research and industry practices. The proposed RCM process is defined in terms of process purpose and process outcomes. A set of activities has been identified and presented in a theoretical model to support and explain the process outcomes. Our proposed model consists of seven core activities: identification, analysis, negotiation and approval, implementation, verification, update deliverables, and communication.

Process outcomes are usually defined in natural languages and may have different interpretations for different users, which could cause ambiguity. A CT, as a semi-formal graphic notation [94], is used to model the proposed process to reduce ambiguity among different users. CTs are also used to compare the proposed process with the existing CMP defined in ISO/IEC 12207:2017. The comparison shows the similarities and highlights the differences between the two processes. Our proposed process sufficiently addresses the deficiencies of the existing CMP and addresses the core activities of RCM.

Thus, this chapter addresses the following research question:

RQ: *How can the proposed RCM process be compared with the existing RCM related processes, and does the proposed process cover every aspect of the RCM problem?*

Furthermore, to align industry practices with the literature findings, we also map between RCM process outcomes and RCM challenges that have been identified by our research and introduced in Chapter 3. To validate our developed mapping, we conducted a questionnaire survey with industry professionals and the results show that more than 90% of the industry professionals agreed with our mapping.

The rest of this chapter is organised as follows: Section 4.2 briefly discuss the previous research related to RCM models and software process in ISO/IEC standards. The proposed RCM process and a theoretical model are presented in Section 4.3. Section 4.4 introduces the CT modelling of our process and compares it with the existing process, and section 4.5 presents the developed mapping between RCM challenges and RCM process outcomes. Finally, the conclusion of this chapter is discussed in section 4.6.

4.2 Related Work

In the past, many studies have been undertaken to explore different aspects of the RCM domain, such as effort estimation [188], rework assessment [139, 163], and traceability analysis [189]. At the same time, a number of RCM models have been proposed [72, 126, 190].

Currently, many ISO standards are practised in the industry, but RCM issues are not fully addressed in these standards. Now industry has realised the importance of RCM in software development's success and stability, and this awareness has been reflected in some latest SDLC processes in standards. For example, ISO/IEC 12207: 2008 [90] discusses some aspects of RCM. Similarly, ISO/IEC 12207: 2017 [28] and ISO/IEC 15288: 2015 [29] outline the CMP and discuss change management as one of its activities; however, those solutions are not sufficient.

In SDLC, configuration management and software change management are two different processes. Configuration management aims to set and maintain consistency among project products and product versioning. In contrast, change management addresses the requested changes to individual products, e.g. system requirements, design, scope, etc. [191]. Configuration management can be considered a supporting process in software requirements change and mainly copes with the challenges of evolution and maintenance

[30, 124, 192] but cannot be used as a substitute for the RCM process. Similarly, configuration management and change management processes are defined separately in the Information Technology Infrastructure Library (ITIL). However, ITIL mainly focuses on delivering information technology services to companies instead of developing projects.

Despite the importance of RCM, existing ISO/IEC standards do not address this domain completely. To fill in this specific research gap, we proposed an RCM process and designed a theoretical model in light of the proposed process. To align with the industry practices, we mapped between RCM challenges and RCM process outcomes and received positive feedback from industry practitioners about our mapping.

4.3 Proposed Requirements Change Management Process

This section introduces the proposed RCM process in terms of its purpose and outcomes, which are commonly used to define a process in standards [193].

4.3.1 Definition

Process Purpose: The purpose of the RCM process is to manage and control requirement changes of system elements or items and make them available to concerned parties.

Process Outcomes:

As a result of the successful implementation of the software requirement change management process:

1. Items to be changed are identified and recorded.
2. Change Impacts are analysed.
3. The cost and schedule of changing items are estimated.
4. Changes to the items under requirements change are approved.
5. Changes to the items under requirements change are implemented.
6. Changes to the items under requirements change are verified and validated.
7. Changed items deliverables are updated and communicated to concerned parties.

4.3.2 Requirements Change Management Process Model Description

This subsection explains the theoretical model of the proposed RCM process. Figure 4.1 shows the model's activities and tasks. The source of change can be either internal or external, serving the purpose of RCM model input. The project management and maintenance team requests are considered internal requests, while external requests come from customers or other stakeholders. The change description and reasons for the change are also included in the change request. The change request will be saved in a change requirement pool for future reference, along with change identification. The Ince model [194] discussed this point as a core element of the RCM model. The use of appropriate notation to capture the changes is the most crucial element of this activity. Software requirement specifications usually apply UML [195], and behaviour trees [91] as a formal notation.

The second activity of the RCM model is to perform change analysis. This activity comprises many tasks, including change impact analysis, cost and schedule estimation. The primary function of change is to estimate the impact of the proposed change on other software artefacts, including design and architecture. Even risk analysis is included in this activity [31].

The next activity is negotiation and approval of change requests. A Change Control Board (CCB) is responsible for authorising the negotiation and approval process [126]. In small teams, normally, project managers act as the CCB, but in large teams, some other team members and system analysts are also included in the CCB [196]. The CCB discusses the change impact on cost & schedule with both the development team and the external stakeholders. After discussion with all the stakeholders, the CCB will decide whether the change request will be accepted, rejected, or sent back to the change identification stage for more information before the next iteration of change analysis. Rejected requests are stored in the change request pool for future reference, and accepted requests forwarded to the implementation stage.

The fourth activity is to implement the approved changes. Change implementation was identified as a core activity in previous research. Change implementation mainly depends on the type of documents used to identify the requested change [125, 197]. In this activity, changes will be recorded in all the impacted documents.

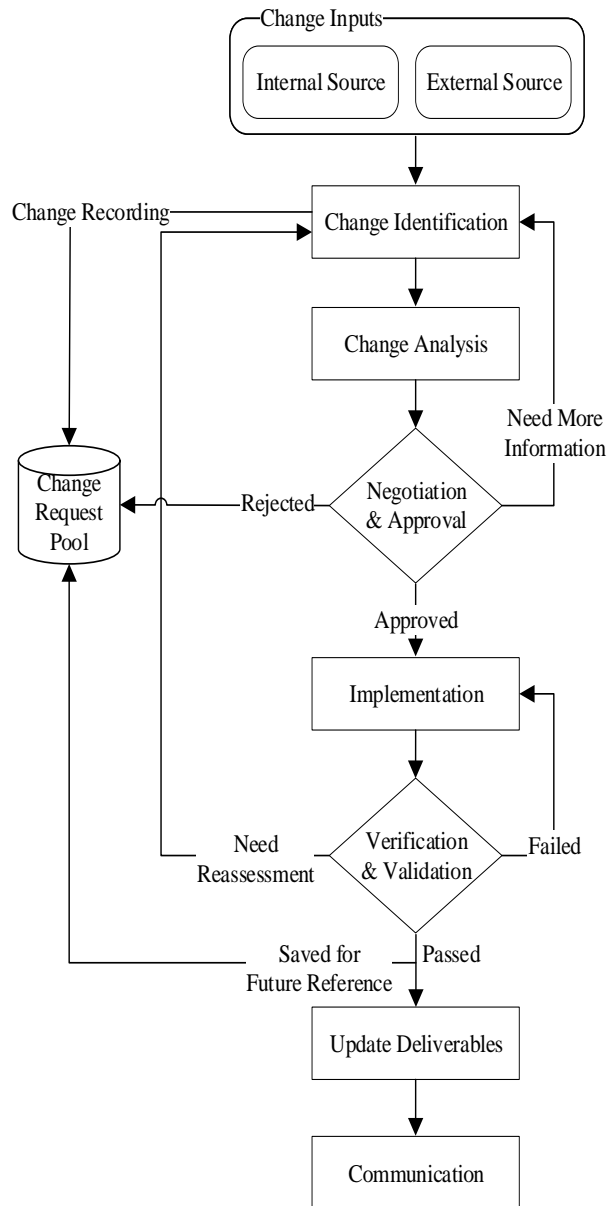


FIGURE 4.1: Requirements change management model

In fifth activity, implemented changes will be verified and validated according to the change request [198], and it is a critical part of an RCM process [199]. The failed changes due to implementation issues will be sent back to the implementation activity team. The successfully passed changes will move forward to the next activity and will be stored in the change request pool for future reference. The third possible output of this activity would be that the implemented changes are sent back to the change identification activity for reassessment.

In the next activity, all deliverables, including requirements, design, code, and testing products that have been impacted due to requested changes, will be updated. In the last stage, the modified deliverables will be communicated to all the stakeholders, so they become aware of this change and will use the updated version.

4.4 Comparison with Existing Processes

This section compares the proposed RCM process with the current CMP defined in the existing ISO/IEC standards. Initially, software processes are written in natural languages, and it is usually difficult to compare a process with its counterpart, also defined in natural languages. Previous research reveals that Behaviour Engineering (BE) notations can be used to model and verify process standards [200]. CT is a part of BE, and it is used to model and compare the software processes. In the next subsections, we first use CT to model the CMP defined in ISO/IEC 12207: 2017 and the RCM process proposed by us and then compare them to find the differences and similarities.

4.4.1 Composition Tree Modelling of Configuration Management Process

To model CMP defined in ISO/IEC 12207: 2017, firstly, the nouns and acronyms are identified from the process outcomes. The identified list includes components and attributes of components in CT. The process name is CT root, and then we go through each outcome one-by-one to identify the components, states, and their relationships and integrate them in the initial CT.

Process Name: *Configuration Management Process*

Process Purpose: *The purpose of CMP is to manage and control system elements and configurations over the system's life cycle. Configuration Management also manages consistency between a product and its associated configuration definition.*

Process Outcomes:

1. *Items requiring configuration management are identified and managed.*
2. *Configuration baselines are established.*

3. *Changes to the items under configuration management are controlled.*
4. *Configuration status information is available.*
5. *Required Configuration audits are completed.*
6. *System releases and deliveries are controlled and approved.*

CMP, item, configuration, and deliveries and releases (DRL) are the list of components identified based on process outcomes. Figure 4.2 shows the CT of CMP defined in ISO/IEC 12207: 2017. The component, Item, is the work product of individual phase, such as requirement specification document, design document, source code, testing document. The “*” sign indicates that the component may have more than one instance.

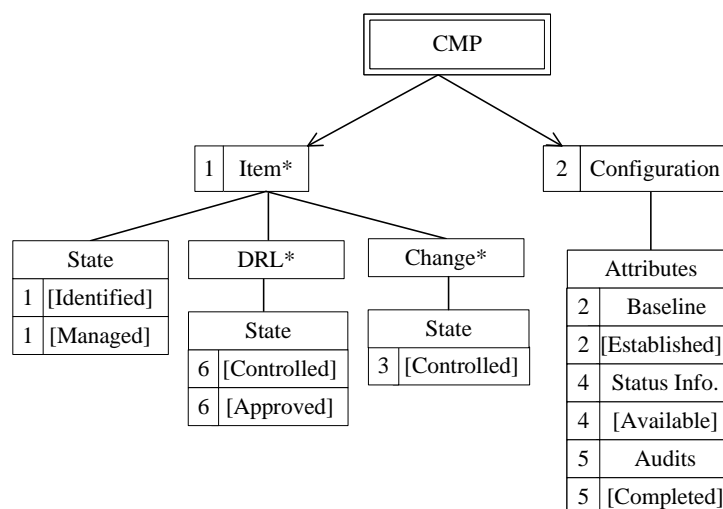


FIGURE 4.2: The composition tree for CMP

4.4.2 Composition Tree Modelling the Requirements Change Management Process

This subsection presents the composition tree of the proposed RCM process. A similar technique is used to construct a CT of the CMP process.

Process Name: Requirement Change Management Process (RCMP)

Process Purpose: The purpose of the RCM process is to manage and control a requirements change of systems elements or items and make them available to concerned parties.

Process Outcomes:

1. Items to be changed are identified and recorded.
2. Impact analysis of changing items is performed.
3. The cost and schedule of changing items are estimated.
4. Changes to the items under requirements change are approved.
5. Changes to the items under requirements change are implemented.
6. Changes to the items under requirements change are verified and validated.
7. Changed items deliverables are controlled and communicated to concerned parties.

RCMP, Items, and Concerned Parties (CP) are the components identified based on process outcomes. Figure 4.3 shows the CT of the proposed RCM process. The component items are the work product of individual phase, such as requirement specification document, design document, source code, testing document. The “*” sign indicates that the component may have more than one instance.

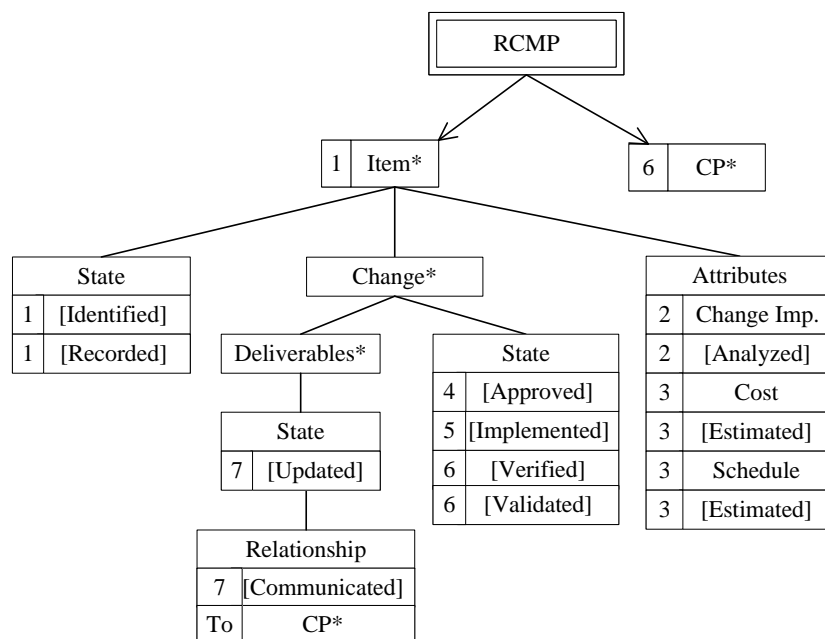


FIGURE 4.3: The CT for RCMP

4.4.3 Comparison between RCM Process and Configuration Management Process

This subsection applies the CT comparison algorithm to identify the differences and similarities between CMP and RCM process. This comparison is performed based on a label matching tree algorithm, which is used to compare different versions of behaviour trees and composition trees [91].

The primary task of the two-step merging algorithm is to find the matching node in CT node names, i.e. components, states, to form the same nodes. Firstly, the components or states are identified that serve the same purpose but which may be called by different names, and then a mapping between these terms needs to be defined. In this research, no such term exists, and therefore this step will be skipped. In the second step, different versions or different trees will be merged to form a Comparison Composition Tree (CCT). To simplify this step, one tree would be called the ‘old tree’ and the other the ‘new tree’. In CCT, the root is a combination of root names of both trees. CCT helps to understand the complete information and the differences between both trees.

For clarity, a CCT follows an intuitive display style convention. The nodes that are part of both the old and the new trees will be represented with normal solid lines. Dotted lines are used to represent nodes that are only part of the old tree, while bold lines are used to represent the information that only exists in the new tree. This is a brief description of the tree merging algorithm; however, complete tree merging algorithm details can be found in previous research [47].

The CCT of CMP 12207:2017 and RCM process is shown in Figure 4.4, and the root is a combination of both tree roots. Similarities and differences are found in the CCT.

- A component called ‘configuration’ is defined in the CMP ISO 12207: 2017, but no such component exists in the RCM process.
- There are no attributes defined for an ‘item’ or ‘work products’ in the CMP ISO/IEC 12207:2017, whereas a number of attributes for an ‘item’ or ‘work products’ are identified in the RCM process.
- One component called ‘concerned parties’ is listed in the RCMP, but not in the CMP ISO/IEC 12207:2017.

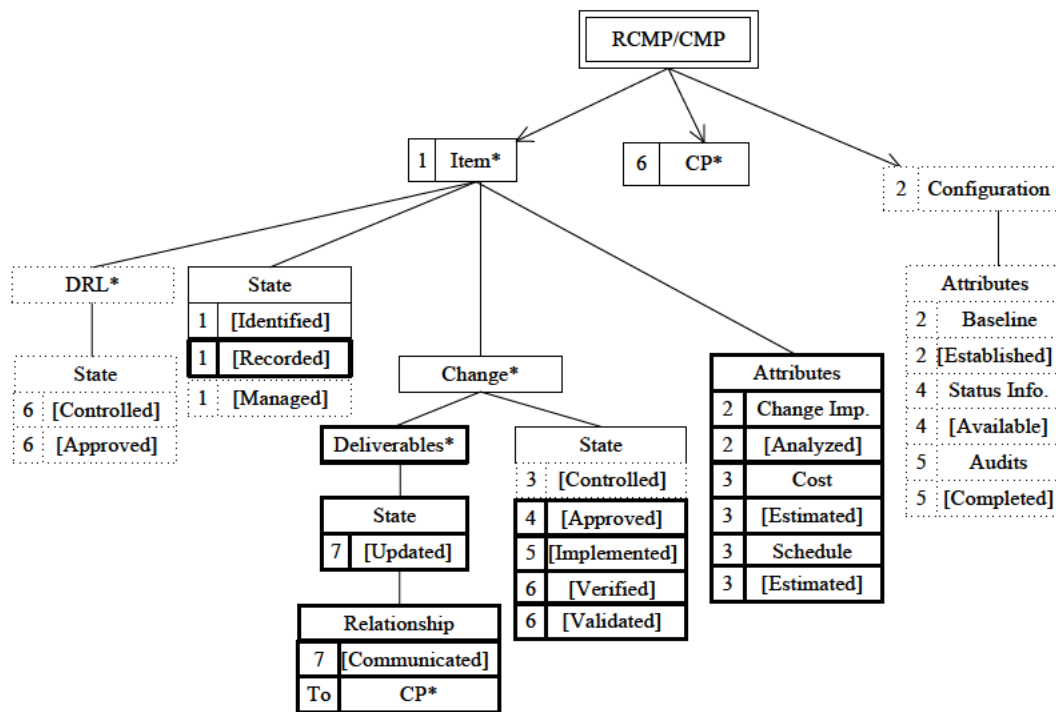


FIGURE 4.4: CCT for CMP of 12207 and proposed RCM process

- ‘Deliveries and releases’ are defined as a component in the CMP ISO/IEC 12207:2017, whereas no such component exists in the RCM process.
- One state of the component ‘item’ or ‘work product’ named ‘identified’ is a part of both trees, while ‘managed’ state exists only in the CMP ISO/IEC 12207:2017 and ‘recorded’ state only exists in the RCM process.
- No attributes are defined for component ‘change’, a sub-component of an item in CMP ISO/IEC 12207:2017, while we have one attribute for this component in the RCM process.
- For the subcomponent ‘change’, only one state called ‘controlled’ is presented in the CMP ISO/IEC 12207:2017, while in RCM process, we have four states, one sub-component with an associated state called ‘updated’.
- Lastly, one relationship of the sub-component ‘deliverables’ is also defined in the RCM process, but no such component is defined in the CMP ISO/IEC 12207:2017.

In summary, deficiencies in the CMP related to change management are thoroughly addressed in the RCM process. Configuration management mainly deals with version control

and system configuration related activities, and the CMP addresses these activities in detail. In contrast, change management is discussed very little in the CMP. The complete change process of an item or work products is defined with only one state called ‘controlled’, whereas the RCM process addresses the change management issues in detail. The RCM process identifies the attributes related to the item or work product that needs to be changed. Different states, such as ‘approved’, ‘implemented’, ‘verified’, and ‘validated’, through which the item or work products need to pass in the change process are also mentioned in the RCM process. Another essential element of the differences is the need to update the deliverables, such as requirements, design documents, source code, and test cases in accordance with required change, which is also included in the RCM process as a subcomponent. Lastly, the communicated relationship is defined, to ensure that all the stakeholders will use the same updated version of all deliverables.

4.5 Mapping between RCM Challenges and RCM Process Outcomes

In this section, we first develop a map between the RCM challenges in both the in-house and Global Software Development (GSD) that we have discussed in Chapter 3 and the RCM process outcomes proposed in this chapter. The mapping will help to understand the challenges might be faced in achieving RCM process outcomes. After that, we conduct a questionnaire survey with the industry practitioners to get their feedback regarding this mapping.

This mapping aims to establish a correlation between the literature findings and RCM process outcomes. As a result, we believe that this mapping will increase the practical significance of RCM challenges by providing an understanding for industry practitioners about what challenges they could face in achieving RCM process outcomes. To help with recall, below is the list of outcomes of the RCM process proposed in this chapter:

1. Items to be changed are identified and recorded.
2. Change impacts are analysed.
3. The cost and schedule of changing items are estimated.

4. Changes to the items under requirement change are approved.
5. Changes to the items under requirement change are implemented.
6. Changes to the items under requirement change are verified and validated.
7. Changed items deliverables are updated and communicated to concerned parties.

In the questionnaire survey, the industry practitioners were asked to give an opinion: to either strongly agree, agree, disagree, or strongly disagree with the defined mapping. The defined mapping and corresponding results are shown in Table 4.1 and Table 4.2.

TABLE 4.1: Mapping between RCM-In-house challenges and RCM process outcomes

Challenge	RCM pro- cess Out- comes	Organisations' Observations (n=69)					
		Positive			Negative		
		SA	A	%	D	SD	%
Impact Analysis	2	28	41	100	0	0	0
Cost/Time Estimation	3	25	41	96	3	0	4
Requirements Traceability	1,7	26	40	96	3	0	4
Artefacts Documents Management	1,4,7	22	43	94	4	0	6
Requirements Dependency	1	15	50	94	4	0	6
Requirements Consistency	4,6	26	38	93	5	0	7
User Involvement	1,3,7	23	39	90	7	0	10
Change Prioritisation	4,5	21	40	88	6	2	12

Note: Strongly Agree (SA); Agree (A); Disagree (D); Strongly Disagree (SD)

The survey results show that more than 90% of the industry practitioners either strongly agreed or agreed with the mapping we formed between RCM-in-house challenges, RCM-GSD challenges, and RCM process outcomes. However, regarding change prioritisation, 88% of the industry practitioners either strongly agreed or agreed with our mapping. A thorough analysis reveals that 100% of the industry practitioners positively responded to the mapping between impact analysis and RCM process outcome number 2. Furthermore, 96% of the industry practitioners gave positive feedback for communication & coordination, knowledge management, and sharing. Similarly, 93% of the industry practitioners either strongly agreed and agreed with the mapping between change control board management, cost/time estimation, and requirement traceability and RCM process outcomes.

TABLE 4.2: Mapping between RCM-GSD challenges and RCM process outcomes

Challenge	RCM Process Outcomes	Organisations' Observation (n=69)					
		Positive			Negative		
		SA	A	%	D	SD	%
Communication and Coordination	1,3,7	19	24	96	2	0	4
Knowledge Management and Sharing	1,4,7	20	22	93	3	0	7
Change Control Board Management	4	21	21	93	2	1	7

Note: Strongly Agree (SA); Agree (A); Disagree (D); Strongly Disagree (SD)

We could not find any suitable mapping between system instability and RCM process outcomes. Therefore, we left this as an open question for industry practitioners, and they gave the following comments regarding system instability.

"I think system instability requires consideration in parallel with other challenging in the complete RCM process, however, more precisely, it can be mapped to outcome 5,6, and 7; as they cover the implementation of the proposed change." Team Lead

One of the other industry professionals revealed his experience as follows;

"It is a big challenge to keep system functionality working and requires considerable planning and effort in the change management process. I think system instability is better correlated with the change implementation phase and will map to outcome numbers 5 and 6." Project Manager

Based on the industry practitioner feedback, the system instability as a core RCM challenge can be mapped to RCM process outcome numbers 5, 6, and 7.

In the survey, practitioners were also asked to mention any best practices that they were following in the RCM process in achieving the RCM process outcomes. Table 4.3 summarises the industry professionals' feedback received as best practices followed while managing changes.

TABLE 4.3: Best practices followed for the RCM process outcomes

RCM Outcomes	Process	Best Practices
Change identification		Use 'how's along with five 'W's
Impact Analysis		<ul style="list-style-type: none"> • Use how's along with five W's • Use requirements analysis matrix • Use traceability matrix
Cost/time estimation		<ul style="list-style-type: none"> • Use a combination of different techniques such as LOC, FPA, use case points, and story points. • Choice of techniques depends on the nature of the project and the impact of the requested change.
Change approval		<ul style="list-style-type: none"> • Change approved after negotiation with the client based on impact analysis and cost/time estimation documents. • Change control boards perform Change approval through a well-defined process.
Change implementation		<ul style="list-style-type: none"> • Agile techniques are more suitable for implementing proposed changes. • Consider all the requirements which were impacted by the requested change directly or indirectly in the implementation plan.
Change verification and validation		<ul style="list-style-type: none"> • Quality assurance verifies and validates Implemented change through informal techniques like inspection and formal techniques like a black box, white box testing. • The selection of appropriate technique depends upon the nature of change such as a change in a graphical user interface, change is system logic etc.
Changed artefacts updating and change communication		<ul style="list-style-type: none"> • Update all the baseline documents including requirements, design, testing and more important schedule and cost document. • Communicate with all concerned stakeholders before releasing the implemented change.

4.6 Conclusion

To fill in the gaps of the existing RCM related processes in ISO/IEC standards, we have proposed a software RCM process in the format of ISO/IEC standards. The process is analysed through a seven-step theoretical model. The seven steps are: identification, analysis, negotiation and approval, implementation, verification and validation, updating deliverables, and communication.

To avoid ambiguity, CTs are used to model the proposed process and the existing CMP

process defined in ISO/IEC 12207:2017. We compare the two processes using their CTs. A CCT is constructed to highlight the differences and similarities between the two processes. The CCT reveals that the proposed RCM process addresses several RCM problems in more detail than the existing CMP.

Furthermore, to help industry practitioners to implement RCM, a mapping has been developed between the RCM process outcomes and RCM challenges. The mapping has been validated through a questionnaire survey to the industry. The survey shows that more than 90% of the industry practitioners agree with the mapping except for one challenge, ‘change prioritisation’. In the survey, we also collected a set of best practices related to RCM process outcomes.

The processed process address the limitations of the existing RCM related process and cover every aspect of RCM domain. However, the proposed process is in the initial stage and not mature enough. We are in contact with the ISO/IEC committee to further improve it. We are also collaborating with our industry partner in china to use this process in a pilot project.

Chapter 5

Change Impact Analysis

After identifying the RCM challenges in [3](#), there is a need to design approaches to address some of the key RCM challenges. By considering it, we choose one of the highly cited RCM challenge identified through SLR and propose an approach to address that issue.

Usually, in real-world systems, the system components are linked with each other to perform collectively to solve any problem. Due to changes in customer demands, and some other factors, the majority of system components are accompanied by frequent change because they are required to keep the system operational. To gain an accurate understanding of the implications of the proposed changes, an effective and efficient Change Impact Analysis (CIA) approach is required.

This chapter proposes an approach to estimate the impact of the proposed requirements changes on other requirements and corresponding design and architecture artefacts. In this work, we use Behaviour Engineering (BE) models such as Behaviour Tree (BT), Composition Tree (CT), Component Interface Diagram (CID), and Requirements Dependency Network (RCDN) a newly introduced model, to perform CIA. Moreover, we proposed a Change Impact Indicator (CII) metric to quantify proposed requirements change impact.

This chapter proposes an approach to estimate change impact, which is one of the highly cited Requirements Change Management (RCM) challenges discussed in [Chapter 3](#). Therefore, this chapter increases the depth of our work related to the RCM problem.

5.1 Introduction

Software requirements keep on changing through the Software Development Life Cycle (SDLC). Frequently, even though the newly introduced requirements only occupy a small portion of system requirements compared to the entire system, the changes could affect some other requirements; they might also impact some critical design artefacts, including the architecture.

The software system's size and complexity make the process of identifying all the change impacts very difficult. According to our previous study [201], understanding the proposed change's impact is one of the major challenges faced during RCM. In another study, it has been reported that 85-90 percent of the software systems budget goes to system operation and maintenance [202]. Therefore, a systematic approach that could be enhanced by automated tools to identify all change impacts is necessary for cost-effective software development [203].

CIA is the term for the kind of systematic approach needed. It is defined as “the activity of identifying the potential consequences, including side effects and ripple effects, of a change, and estimating what needs to be modified to accomplish a change before it has been made” [31]. CIA helps to estimate the extent and cost of implementing the proposed changes. For many decades, various CIA techniques have been introduced and applied at different stages of the SDLC on different artefacts, for example, on requirements [204, 205], architecture [206–209], source code [210–212], and a combination of them [213].

Furthermore, a few studies have been conducted that trace back the changes from one software artefact to another artefact; for example, from source code to software design. Hammad et al. [71, 214] presented an approach that monitors the evolution of a design based on the changes in source code.

In the context of change propagation from requirements to design, Al-Saiyd and Zriqat [215] presented a traceability based approach that uses traces between requirements and design. Similarly, Sudin and Kristensen [70] presented an approach to understand how changes in requirements are carried out during the design process. However, both of these approaches manually identify the potentially impacted design elements based on requirements changes.

In another more relevant work, Goknil et al. [34] proposed a rule-based CIA approach to identify potentially impacted elements in software architecture. They used requirements relations, which they also proposed in other of their work [216]. After that, they used traces between requirements and architecture elements to determine candidate architectural elements for the impact of requirements change in the architecture.

Generally, the above-discussed approaches have the following limitations. Firstly, they will only provide a rough set of candidate elements that might be impacted due to requirements change [34]. And it is laborious to identify the actual impacted elements from the list of candidate impacted elements in large systems because the set often contains too many potentially impacted elements.

Secondly, existing CIA approaches usually define system artefacts at the abstract level without specifying them in more precise modelling languages. The change impacts are described in natural languages rather than in more formal and precise modelling language such as Unified Modelling Language (UML) and Behaviour Tree (BT); therefore, they usually lack the capability to illustrate the propagation from requirements to other software artefacts in an understandable and verifiable way.

Thirdly, most of the approaches do not provide any suitable measure to quantify the change impact. They usually only provide the list of potentially impacted elements without considering their complexities, which helps estimate implementation cost more objectively. Without quantifying the change impact, estimating the implementation cost for the proposed change is very challenging [36].

To address these limitations, we proposed a Behaviour Engineering-based approach for Change Impact Analysis (BECIA). This approach addresses all three limitations very well. To achieve that, we have fully taken advantage of the advanced features of BE and so make the following contributions:

- Our approach uses models from BE such as Integrated Behaviour Trees (IBT) and Integrated Composition Trees (ICT) to analyse requirement changes and affected components. Both IBT and ICT are important models of a software system, and details are discussed in Chapter 2 and section 2.5. We have proposed a new model called Requirements Components Dependency Network (RCDN), which captures relationships between requirements and corresponding components, and it helps to

identify candidate impacted components. RCDN can be used with another BE model to identify impacted components, which helps to address the first limitation.

- Moreover, this approach systematically transforms models of one software artefact into another software artefact instead of creating artefacts from natural languages. In particular, we introduce algorithms to transform IBTs to ICTs, and then to RCDNs. This translation maintains traceability links between these artefacts, which helps to demonstrate change propagation between these artefacts; therefore, it addresses the second limitation.
- To quantify change impacts (third limitation), we introduced a Change Impact Indicator (CII). We use the concept of Kolmogorov Complexity (KC) [217] to estimate a component's complexity and then use it to calculate CII. With an objective measurement of change impact, our approach, aligned with historical data, produces a reliable benchmark to estimate the cost associated with a proposed change. It might also help to identify an optimised change solution among several possible change solutions.
- We show the applicability of the proposed approach using a real-world system named Shaanxi Transportation and Logistics Information Standardisation Management System (STLISMS). Our approach successfully identifies the components that are impacted due to the proposed change.

The rest of this chapter is organised as follows: Section 5.2 introduces all the key elements (contributions) of the BECIA approach. Section 5.3 illustrates the workflow of BECIA with a simple abstract example, and a real-world case study is given in Section 5.4. Section 5.5 presents relevant existing works and compares the proposed approach with relevant existing approaches. Finally, the conclusion is discussed in section 5.6.

5.2 Key Elements of BECIA

This section introduces the four key components of our approach, which are: a translation from IBTs to ICTs, from ICTs to RCDNs, from IBTs to CIDs, and the CII metrics. We introduce each of the components below.

5.2.1 Conversion of an IBT into an ICT

The original BE methodology translates both BTs and CTs directly from the requirements. The CT translation and integration process helps to identify requirement defects and fix ambiguity in the terminologies of the requirements. It is a useful process if the requirements are not well written and the design team is not familiar with the requirements. However, once most requirement defects are fixed and the design team has sufficient knowledge of the project, to stick on this manual process may have the following problems:

Firstly, as the translation and integration process requires manual effort, it could cause inconsistency between the IBT and the ICT due to human errors. Secondly, if the requirements change, it involves more work to update both the IBT and the ICT to keep them consistent with the latest version.

In this research, we have proposed an algorithm to convert an IBT to an ICT. Besides the benefit of preserving links between artefacts, as discussed before, the automated translation has two additional benefits: it will save the human effort required for the updating of IBT and ICT during change management, and guarantee consistency between the two diagrams.

Now we will explain the conversion of IBT into the ICT with a simple abstract example. Supposing a system of four requirements has its corresponding IBT shown in Figure 5.1. It has five components: C1, C2, C3, C4, C5, and eight behaviours from B1 to B8. In this example, all the three-leaf nodes have a reversion sign ‘ \wedge ’, which mean to revert to the closest parent node of the same behaviour.

The process to convert an IBT to an ICT involves applying a tree traversal algorithm; it starts from the root node of the IBT and walks through all the nodes. As an ICT shows static aspects of the system, therefore, it does not matter whether one walks through the IBT in a depth-first order or a breadth-first order, as long as all the behaviour nodes are covered.

Before constructing the ICT, we need to identify the system component which serves as the root node of the ICT. The system component is usually also a component of the IBT’s root node, as it is a recommended practice in developing BTs. Even if it is not the IBT’s

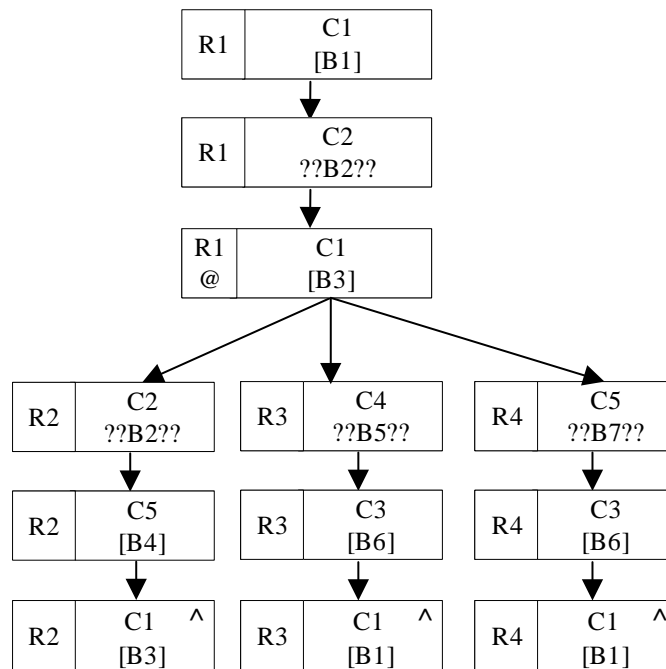


FIGURE 5.1: Example IBT

root node, we can still identify a system component, which is drawn in a double-lined rectangle.

To process each behaviour node in the IBT, the following procedure will be applied.

1. Check the component name; if the component is not in the ICT yet, we will create a component node in the ICT. A simple ICT contains one system component, and then all the other components are direct child components under the system component. A more complex system may have a system of systems, and then it will have more complex composition trees. However, the composition structure of the ICT could be adjusted based on design decisions.
2. Check the node's behaviour and requirement link. It could be in one of the three different situations:
 - (a) If the ICT does not have a node with the same behaviour under the component, then it will create a new node with the behaviour and requirement link under the component, as shown in Figure 5.2 (a).

- (b) If the ICT already has a node with the same behaviour under the component, but the requirement links cell (of the CT node) does not contain the requirement link of the BT node. The new requirement link from the BT node will be added to the CT node's requirement link cell, which contains a list of all requirement links associated with this behaviour. For example, the last node of requirement R3 has component name C1 and behaviour B1, which are already included in ICT; however, requirement link R3 is not included in the requirement links cell. Therefore, we add requirement link R3 in the left cell, as shown in Figure 5.2 (b).
- (c) If the ICT already has a node with the same behaviour under the component, and the requirement link from the BT node is already in the CT's requirement link cell, nothing needs to be done. For example, behaviour B3 appears in two nodes of requirement R3, but we write it only once, as shown in Figure 5.3.

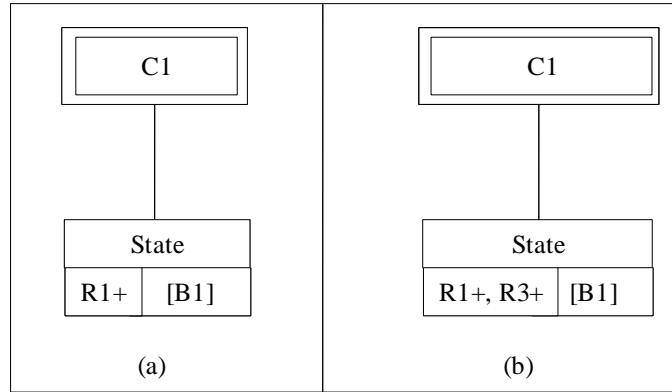


FIGURE 5.2: Steps to model example ICT

An important point regarding step 2 is that if the associated behaviour changes the component state, it means that the component state before and after executing that requirement is different. In other words, if any requirement introduces a new state of a component, then it shows that requirement changes that component state. Then we write plus sign '+' with the requirement ID; otherwise, simply write the requirement ID in ICT.

Lastly, after following similar steps (1 & 2), we will get the ICT for all four requirements, shown in Figure 5.3. An important point related to this process is that the integration node of IBT will be shown with both requirements links in ICT because this node is

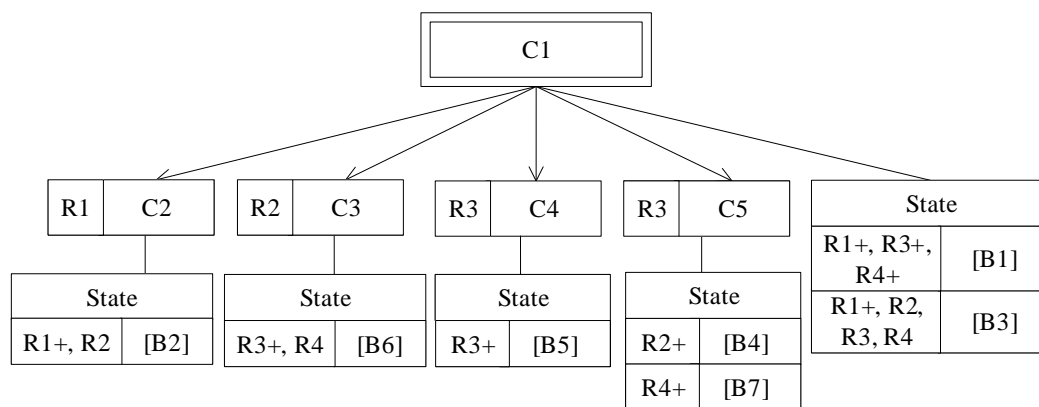


FIGURE 5.3: Example ICT

a postcondition of one requirement and a precondition of the second requirement. For example, in Figure 5.1, an IBT node with @ sign is used to integrate R1 with R2, R3 and R4. So, in ICT, the behaviour B3 of component C1 is shown with requirements links R1, R2, R3, R4 in Figure 5.3.

5.2.2 Conversion of an ICT into RCDN

As the original requirements are individual functional requirements, the implementation dependency relationship among them is not visible. Efficient requirement analysis is necessary to represent the relationships clearly and visibly.

Traditional techniques identify the relationships manually, and they usually reveal the requirements relationship in the problem domain. For example, in a use case diagram, “to create an order” and “to pay an order” are drawn as two use cases closely related in the problem domain. Because those diagrams are created before considering implementation, they usually will not reflect the relationship of two requirements in the solution domain.

As design moves closer to the solution domain, current programming tools increasingly focus on components, which are connected through complex dependencies. Therefore, a hidden dependency between requirements in the solution domain might cause other requirements affected if one requirement is changed. For example, in the Year 2000 Problem (Y2K) situation, the date format change might cause the wrong calculation of retirement pensions. Hence, we aim to develop a new diagram that reflects the dependency relationship in the solution domain.

Our new model captures the dependency relationships among requirements based on their connection with components. Intuitively speaking, if two requirements both involve the state of the same component, those two requirements are closely related; similarly, if the implementation of two requirements does not involve any mutual components, these two requirements are relatively independent. To achieve that, we have introduced a new RE model called RCDN, which connects all the requirements through the components involved in the implementation of those requirements.

An RCDN is a hypergraph $H = (V, E)$, where V is the set of requirements as vertices and E is the set components as hyperedges. Requirements v_1, \dots, v_n are connected by a component e if v_1, \dots, v_n all involve the state of e . The usual graphs are inappropriate in this situation because, in usual graphs, one edge can connect to at most two vertices. However, our approach uses hypergraph because one component (hyperedge) can connect to more than two requirements (vertices). Hypergraph models have shown great effectiveness in capturing higher-order relationships. An RCDN visualises which requirement or component is critical by investigating the number of their connections with other requirements and components. Intuitively, if the change involves a critical requirement, it will most likely require more effort for implementation than for less critical requirements.

Similar to our translation from IBTs to ICTs, our approach translates ICTs into RCDNs. To explain the conversion from an ICT into an RCDN, we used the same example that we have used in the previous subsection and the ICT in Figure 5.3. We will take one component and read all the associated requirement IDs to form hyperedges. We will start with the root component, C1, and will form an RCDN by using the following steps:

1. Check the component name; if the component is not in the RCDN yet, we will create a component as a circle.
2. Check the requirement links in the left cell of the ICT node under the state's section of that component. It could be in one of the three different situations:
 - (a) If the RCDN does not have that requirement, then create a new requirement as a rectangle. If there is a '+' sign next to the requirement, add a strong connection (represented by a solid line) between the requirement and component; otherwise, added a weak connection (represented by a dotted line). For example, in component C1, the first requirement is R1 in the requirement links

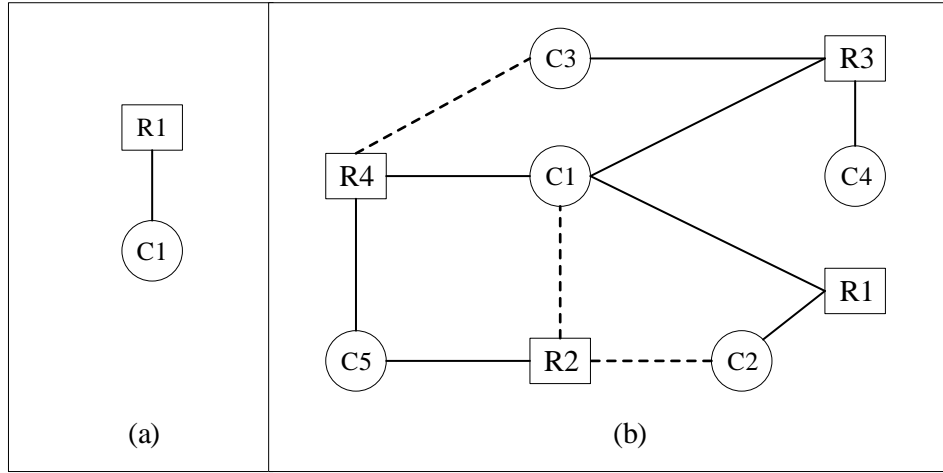


FIGURE 5.4: The RCDN for the example ICT

cell, so we create a new requirement **R1** and connect it with component **C1** as a strong connection, as shown in Figure 5.4 (a). Because requirement **R1** is listed with a plus sign means **R1** changes component **C1**.

- (b) If the RCDN already has a requirement, we only connect with the component based on the connection type.
- (c) If the RCDN already has a requirement and is connected to that component through a weak connection, but the same requirement entry with another behaviour in another node requires a strong connection, then, replace the weak connection with a strong connection.

Lastly, we will get a complete RCDN for all four requirements and five components after following similar steps, as shown in Figure 5.4 (b).

After generating an RCDN, we define different relationship types between requirements (vertices) based on their connections. The three types are: competing, supporting, and sharing. The definition of the three different types and the rules to apply them for checking possibly impacted requirements, are given in Table 5.1.

Among the three different relationships, ‘competing’ has the highest precedence and ‘sharing’ has the lowest precedence. It means if two requirements are both competing and supporting, they are ‘competing’; if both are supporting and sharing, they are ‘supporting’.

TABLE 5.1: The relationship types in RCDN and their interpretation rules for indirect impact analysis

#	Relationship Type	Definition	Rules for indirect impact analysis
1	Competing	Two requirements are called competing if both of them may update the state of one component. In other words, both of them have strong connections to a component. For example, in Figure 5.4 (b), both R1 and R3 have a strong connection to C1; therefore, they are competing requirements.	Requirements Change Propagation Rule 1: If two requirements are competing, and one is changed, then the other requirement needs to be checked if the change may impact it.
2	Supporting	If two requirements are connected to one component, with one strong connection (updating the component's state) and one weak connection (reading the state), the two requirements are supporting. For example, in Figure 5.4 (b), R1 has a strong connection to C2, and R2 has a weak connection to C2; therefore, they are supporting requirements.	<p>Requirements Change Propagation Rule 2: If two requirements are supporting and the changed requirement has a strong connection to the linking component, then we need to check if the other requirement might have been impacted.</p> <p>Requirements Change Propagation Rule 3: If two requirements are supporting and the changed requirement has a weak connection to the linking component, then we do not need to check if the other requirement is impacted.</p>
3	Sharing	If two requirements are connected to a component through weak connections (reading state from the component), the two requirements are sharing.	Requirements Change Propagation Rule 4: If two requirements are sharing and one requirement is changed, then we do not need to check if the other requirement is impacted.

For example, in 5.4 (b), R2 and R4 are competing through C5 and supporting through C1; therefore, the relationship between R2 and R4 is competing.

5.2.3 Transform IBT into CID

This subsection briefly describes another key element of BECIA: transforming an IBT into a CID. This concept is already explained in the existing published research [91]. An IBT is the system's analysis domain view that shows all the states and the flows of control, modelled as component-state interactions. In an IBT representation, a given component may appear in different parts of the tree in different states. Due to this, it becomes hard to understand any specific component details. To get a detailed understanding of individual components, we need to convert the IBT to CID.

A CID shows the interface of a component that includes both what other components are linked to the component and what other components it links to. A CID acts as a blueprint for the implementation of a component. In this research, we use CID to check whether a component is actually changed or not, and if changed, what types of change happen in that component. A CID can be directly projected from the IBT. The first step to project a component's CID is to highlight all the nodes in the IBT of the given component. As a result, we have all the links both coming in and going out of the component. Figure 5.5 shows the CID for component C1 projected from the IBT shown in Figure 5.1.

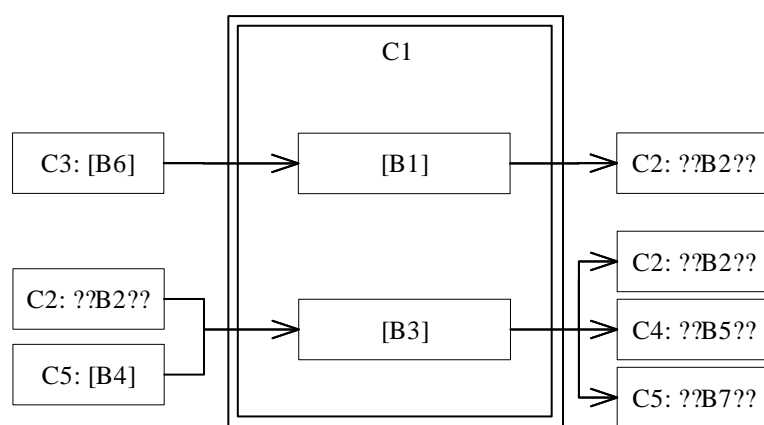


FIGURE 5.5: The CID for component C1

5.2.4 Change Impact Analysis Metric

Existing CIA techniques are usually on the qualifying level, and this research will be on the qualifying level as well as quantifying level. That means we will not only identify which components and requirements will be affected by the proposed change, but also quantify the change impact through a newly introduced Change Impact Indicator (CII). Comparing the change impact indicators might help identify an optimal change solution among many possible change solutions and provide more accurate and objective change effort estimation based on historical change benchmarks.

According to requirements change taxonomies [218–220], changes can be classified into three categories: adding new requirements, deleting an existing requirement, and modifying an existing requirement. Therefore, our proposed metric covers these aspects of requirements change.

In BECIA, we first identify a set of components from RCDN that might be impacted due to the proposed change. After that, we use CID to quantify the actual impact on each component. The set of impacted components is defined as follows:

$$\Theta = \Theta_n \cup \Theta_m \cup \Theta_d \quad (5.1)$$

Here Θ is the set of impacted components, Θ_n is the set of new components, Θ_m is the set of modified components, and Θ_d is the set of deleted components.

To quantify the change impact on a component, we have applied a method similar to the one for estimating the descriptional complexity of the component. After that, the change impact on the entire system is defined as the sum of the change impacts on all the components.

In a CID, each component contains a number of states (or interfaces), and each state has a connection with a number of incoming and outgoing components. As shown in Figure 11, component C1 has two states [B1] and [B3], while [B1] has one incoming component and one outgoing component; [B3] has two incoming components and three outgoing components.

In this work, to objectively measure the impact of the proposed change, we propose a two-fold approach. Firstly, our aim is to identify which component might be changed or affected due to the proposed change. Secondly, we are interested in investigating how difficult it is to change the changed component because each component structure is different from others and different amount of effort is required to implement a change in it [221]. Intuitively, a component with more states and more incoming and outgoing links is more complex. However, manipulating this set of numbers to estimate a convincing measure of complexity is not an easy task. Kolmogorov Complexity (KC) [217] has been studied and accepted as a good complexity measure. KC has been applied in software engineering for many different applications and received positive results [222]. KC will help to estimate the complexity of each changed component based on its structural proprieties.

In our approach, we use the KC to measure the complexity of individual impacted components and use it as a change impact indicator on the component, and then estimate the overall change impact by combining the change impacts on all impacted components. Before introducing how we use the KC concept to measure the complexity of a component, we introduce a general formula to estimate an upper boundary of KC of an integer n .

$$K(n) \leq \log^* n + c \quad (5.2)$$

Where n is an integer, $K(n)$ is the KC of n , and c is a constant, which can be ignored when making the comparison. Where \log is with base 2 and $\log^* n$ is defined as:

$$\log^* n = \log n + \log \log n + \log \log \log n + \dots \text{ until the last positive term.}$$

Considering the practical situation and simplifying the calculation, we use $\log n + 1$ to estimate the complexity of non-negative integer value n .

$$K(n) \leq \log(n + 1) + c \quad (5.3)$$

After defining the fundamental equation to measure an integer's complexity, we now calculate the complexity of a component. In CID, each component is composed of a number of states; therefore, the complexity of a component can be estimated as the sum of all states' complexities. Let S be a state in component C , n_c is the number of states

in C . The complexity of state S can be estimated as:

$$K(S) \leq \log(n_c + 1) \log(n_{s_i} + n_{s_o} + 1) \quad (5.4)$$

Where $K(S)$ is the complexity of a state in a component, n_c is the number of states in that component, n_{s_i} is the number of incoming components of S , and n_{s_o} is the number of outgoing components of S . We have omitted the constant c while calculating the complexity of a state because the value is meaningful only when they are compared to one another; therefore, the constant c could be omitted as it appears in all the compared items.

For example, in Figure 11, the KC of state B1 can be calculated as:

$$K(B1) \leq \log(2+1)\log(1+1+1) = 1.58 \times 1.58 = 2.51$$

After estimating the complexity of individual states, the complexity of a component can be calculated as:

$$K(C) = \sum_{S \in \phi} K(S) \quad (5.5)$$

Where $K(C)$ is the Complexity of component C and ϕ is the set of states of component C . Based on this, we can calculate the complexity of C1 in Figure 11 as:

$$K(C1) = 2.51 + 4.08 = 6.59$$

After measuring the individual component's complexity, we now calculate the impact of the proposed change on system components. In this research, we define a change impact indicator as the complexity of the components involved in the proposed change. As we have discussed earlier in this section, three types of change can be made in software systems: adding a new requirement, modifying, and deleting an existing requirement. Similarly, there are three types of changes at the components level: adding a new component, modifying, and deleting an existing component. To estimate the overall impact, first, we estimate the impact on individual components.

We use δ_C to denote the change impact factor on component C , For a new component, we define the change impact factor as the component's KC, calculated in Equation 5.5

$$\delta_C = K(C) \quad (5.6)$$

For a modified component, we classify its states in three different sets: newly introduced states, modified states (some may have an incoming component number and/or outgoing component number changed), and deleting states as below:

$$\Phi = \Phi_n \cup \Phi_m \cup \Phi_d \quad (5.7)$$

While Φ is the set of all states in component C , which is equal to the set of newly introduced states Φ_n , the set of modified states Φ_m , and the set of deleted states Φ_d .

If any state name is changed, we assume to delete that state and then create it as a new state. We define the change impact indicator on the deleted states as 0; therefore, the change impact indicator of the modified component contains two parts: the first part calculates the change impact on the set of newly introduced state Φ_n , while the second part calculates the change impact on the set of modified states (means incoming or outgoing components changed) Φ_m . The formula is given as:

$$\delta_C = \sum_{S \in \Phi_n} K(S) + \sum_{S \in \Phi_m} \max(K(S') - K(S), 0) \quad (5.8)$$

Where δ_C is the complexity of a modified component, $K(S)$ is the Complexity of the new state and also Complexity before making a change, $K(S')$ is the complexity of a modified state after performing a change. We use 'max' here to avoid negative values, which means if complexity is equal to any negative value, then we take zero.

An important point related to this process is that we will not introduce a separate equation to calculate the impact of deleting a component. The reason is that during the deletion activity, we need to update the interfaces of the components connected to the deleted components. Therefore, a separate equation to calculate the impact of deleting a component will be covered in the modified and new components.

Finally, the cumulative impact of the proposed change in the context of complexity can be estimated as follows:

$$\Delta = \sum_{C \in \Theta} \delta_C \quad (5.9)$$

Where Δ is the overall change impact factor, Θ is the set of impacted components defined in Equation 5.1.

In the next section, we will use a simple example to demonstrate the detailed procedure to calculate the change impact indicator.

5.3 BECIA Workflow with an Example

This section first explains the BECIA workflow by combining all the key elements and then uses an example to demonstrate the process.

5.3.1 BECIA Workflow

The workflow of BECIA is given in Figure 5.6, which is composed of seven steps. Step 1 is in the problem domain, steps 2-4 in the analysis domain, and steps 5-7 in the solution domain of the software development life cycle.

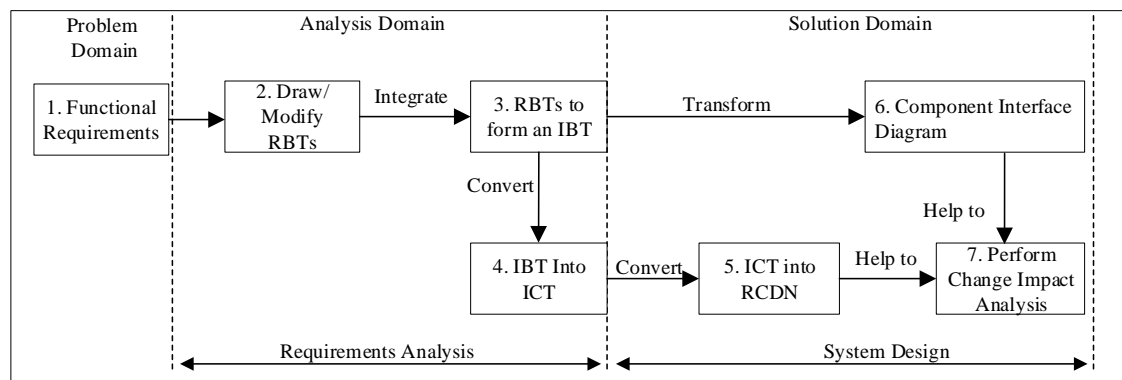


FIGURE 5.6: BECIA workflow

- In step 1, System requirements described in natural languages are gathered. In this research, we only consider functional requirements.

- In step 2, new RBTs are designed, and/or existing RBTs are modified based on the changed requirements by using BT drawing tools such as iRE [104].
- In step 3, new and/or modified RBTs are integrated into the existing IBT.
- In step 4, the IBT is converted into the ICT.
- In step 5, the ICT is converted into the RCDN.
- In step 6, the CIDs of all impacted components are projected out. This step can be performed parallel to step 5 because both these paths originate from step 3.
- In step 7, calculate the change impact indicator from the RCDN and CIDs.

The first step is related to the software requirements gathering and will be performed according to the guidelines of the requirements engineering phase of the software development life cycle; therefore, we will not describe it in this paper. Steps 2 and 3 are standard practices in the BE approach that were introduced in the background section and has been published in previous papers [91, 104], so we will only provide the results of those steps without explaining the detailed process.

5.3.2 Impact Analysis Example

This section uses a simple example to explain the BECIA workflow, particularly step 7, which is to calculate the change impact indicator.

We reuse the IBT example in Figure 5.1 and then create a new RBT based on an assumed new requirement. Then we perform steps 3-7 of BECIA based on the IBT and RBT.

After that, as it is possible to have multiple designs based on the same requirements, we slightly adjust the new RBT to represent a new design for the change request. And then repeat the BECIA process to calculate the change impact indicator of the second design. By comparing the change impact indicators associated with different designs, the designer might be able to identify an optimised design.

5.3.2.1 Steps 2 and 3- Draw/modify RBTs and Integrate them with the IBT

Based on the assumed new requirement, in step 2, we have created a new RBT(R5) shown in Figure 5.7. After that, in step 3, we integrate this RBT with the IBT in Figure 5.1, and the result is shown in Figure 5.8. The two trees are merged in the root node, which is drawn with a thick border. To highlight the change, we use a grey background colour to illustrate new nodes, and a light grey colour with a pattern fill to illustrate modified nodes.

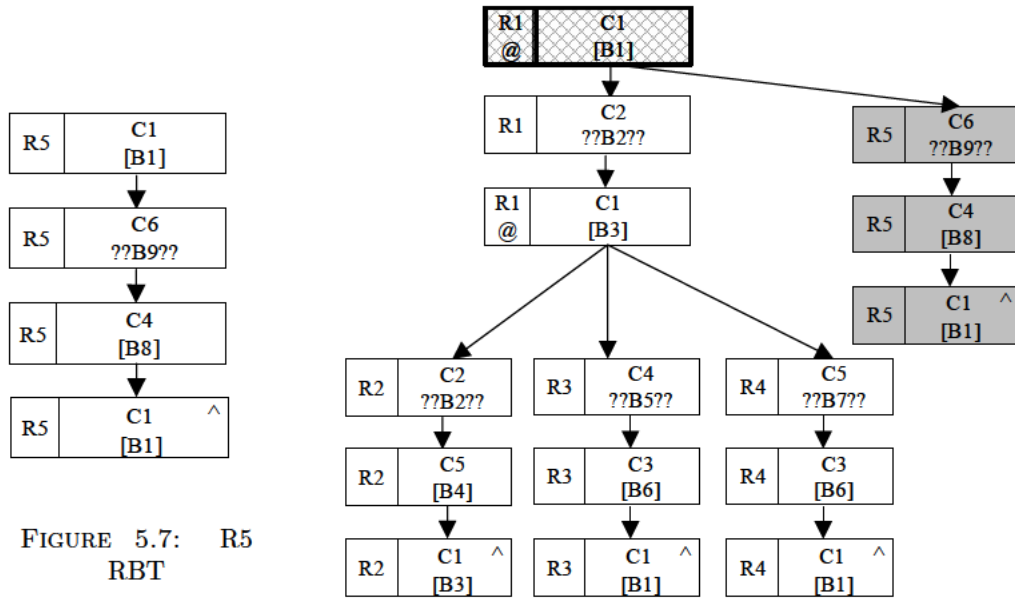


FIGURE 5.7: R5 RBT

FIGURE 5.8: Updated IBT

5.3.2.2 Step 4- Convert an IBT into an ICT

In step 4, we show the process to convert the updated IBT into an ICT. Because the original IBT in Figure 5.1 already has an associated ICT in Figure 5.3, we only need to process the updated section in the IBT to modify the associated ICT accordingly.

We read each node one by one from the updated IBT and incorporate it into the existing ICT based on the two steps defined in subsection 5.2.1. For example, first, we read the root node of R5, an integration node with R1, so we update the state section of the C1. The node with behaviour B1 already exists, so we simply include requirement ID in the left cell and make this node boundary bold with a pattern fill to show the change. On the

other hand, if no node exists for changed behaviour, we model a new node in a grey colour background, as shown for one node of component C4 in Figure 5.9. If any new component appears in the changed requirement, then we create a node for the new component and connect it to the tree's root, as done for component C6.

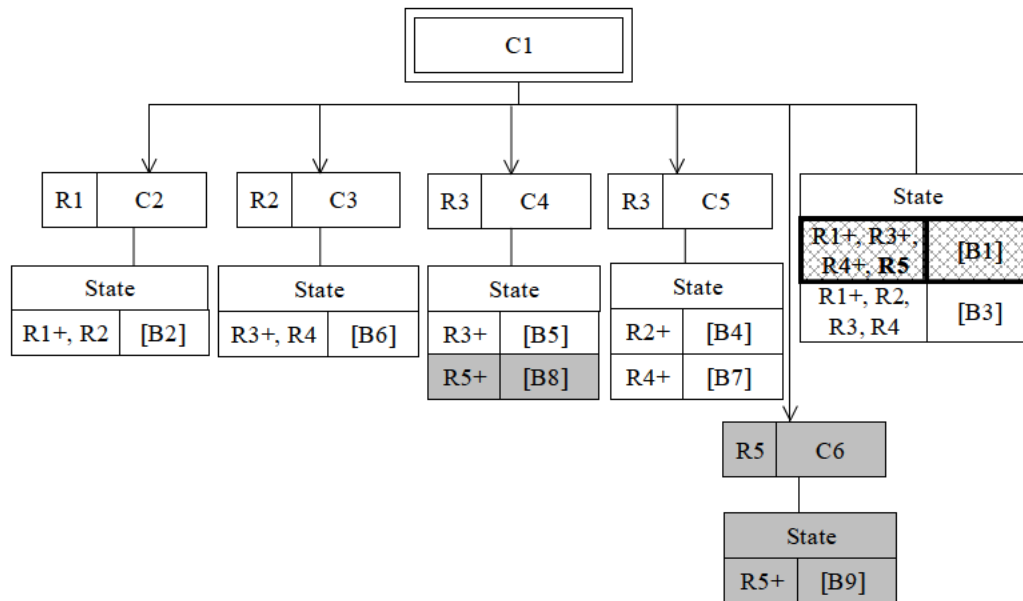


FIGURE 5.9: Updated ICT

5.3.2.3 Step 5- Convert an ICT into an RCDN

In step 5, we update the RCDN based on the changes in ICT. We will follow the same process discussed in subsection 5.2.2. As this requirements change happens as a result of adding a new requirement, we first model a new requirement, R5, as shown in Figure 5.9 in the grey coloured background rectangle. After that, we see if any new component is added to the system or not. One new component, C6, is added, so we create component C6 as the grey coloured background circle and connect it with R5.

After that, we searched for updated components, and we found one new node in component C4, and this node changed the component state, so a strong connection was formed between R5 and C4. We show the changed component/requirements in the bold boundary with the pattern fill, as in Figure 5.10. Lastly, we also find one new requirement ID in the existing state of component C1; however, that requirement does not change the

state of component C1, so we connect requirement R5 to component C1 through a weak connection, as shown in Figure 5.10.

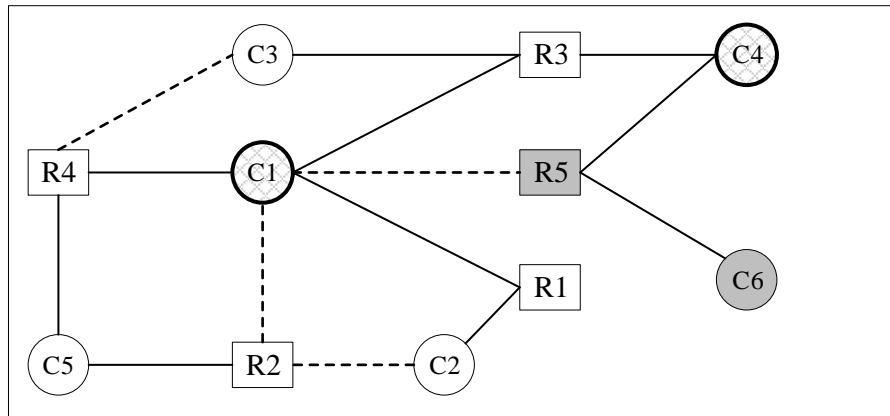


FIGURE 5.10: Updated RCDN

5.3.2.4 Steps 6 and 7- Project out CIDs from an IBT and Calculate Change Impact

Step 6 is to project out CIDs from the updated IBT. However, we don't need to project out CIDs for all components; we only need to get the CIDs that are impacted by the proposed change. To identify the possibly impacted components, we only need to check the RCDN, which shows all the newly introduced components as well as modified components. We will apply four rules for requirements change propagation defined in subsection 5.2.2 to RCDN to identify the potentially impacted components and then use CID to find the actually impacted components.

Let's investigate the CIDs of potentially impacted components. Checking Figure 16, we find one new component C6 and two modified components C1 and C4. We project out the CIDs of the three components from the IBT in Figure 5.8. The CID for C1 is shown in Figure 5.11, and it shows that a couple of interfaces get updated due to this change. The CID of C4 is shown in Figure 5.12, and it reveals that one new state is added in component C4 and one outgoing interface is also changed.

Now we investigate the possible indirect impacts of the proposed change based on the relationships between a changed requirement and other requirements. As R3 and R5 are competing requirements, based on requirements change propagation rule 1, we need to

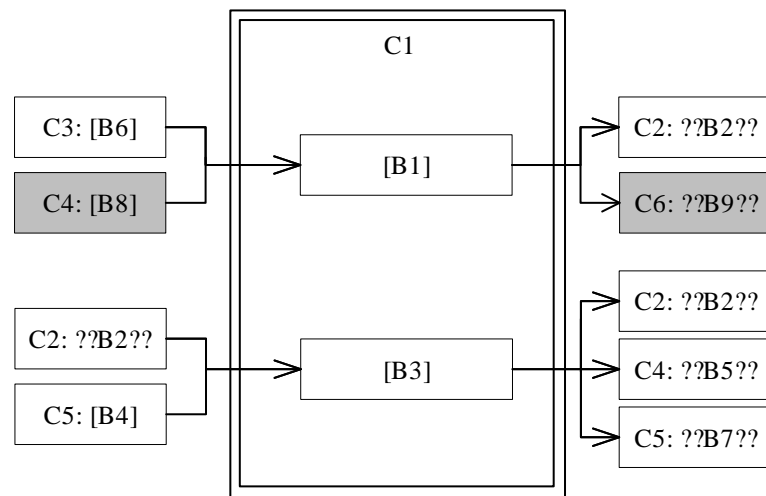


FIGURE 5.11: The CID for component C1

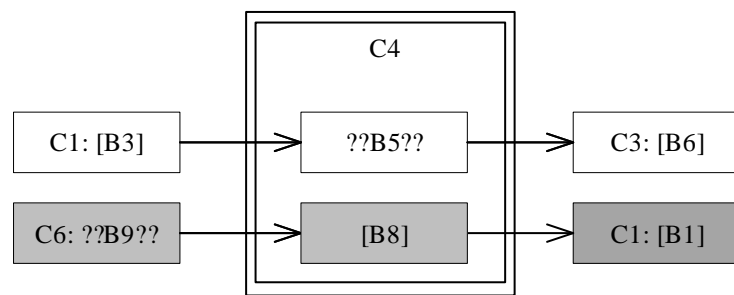


FIGURE 5.12: The CID for component C4

check components connected to R3 as well. Requirement R3 is connected to components C1 and C3; C1 is already checked, so now we only check C3. The CID of C3 in Figure 5.13 shows that no change happens in component C3.

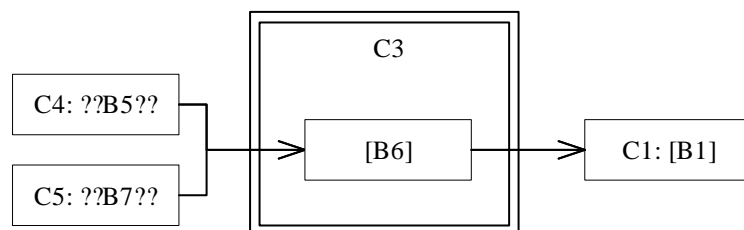


FIGURE 5.13: The CID for component C3

Furthermore, The RCDN in Figure 5.10 shows that the new requirement R5 is connected

to R1, R2, R3, and R4 by supporting and sharing relationships through component C1. However, R5 is connected to C1 through a weak connection; therefore, according to requirements change propagation rule 3, change cannot propagate through component C1. As no new component is identified that needs to be checked, so we terminate this process.

After identifying the actually impacted components, now we measure the complexity of these components. We use the CID of these components and Equation 5.5 to calculate the complexity of each state in the new component. The CID of C6 is shown in Figure 5.14, and the complexity of its state 'B9', would be calculated as follows:

$$K(B9) \leq \log(1+1)\log(1+1+1) = 1 \times 1.58 = 1.58$$

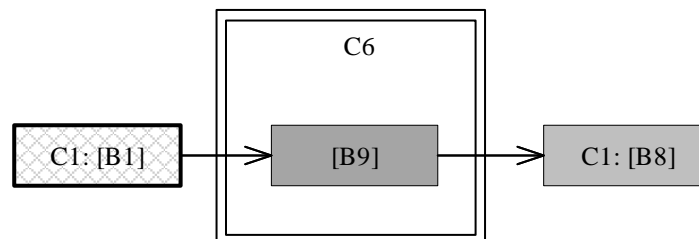


FIGURE 5.14: The CID for component C6

There is only one state in the new component, so this complexity is also a complexity of a new component, C6. After calculating the complexity of a new component, based on Equation 5.6, the change impact of the new component will be:

$$\delta_{C6} = 1.58$$

Now we calculate the complexity of the modified component. First, we show the calculation process for component C4 (CID in Figure 5.12), which has two states, one state (B5) is unchanged, and one is a new state (B8). In the case of an unchanged state, the complexity would be zero, and for the new state, complexity would be:

$$K(B8) \leq \log(2+1)\log(1+1+1) = 1.58 \times 1.58 = 2.51$$

After measuring the complexity of all the states of C4, the change impact for modified component (C4) can be calculated by using Equation 5.8 as follows:

$$\delta_{C4} = (0 + 2.51) = 2.51$$

Similarly, for component C1 (CID in Figure 5.11), one state (B3) is unchanged, its complexity is zero, and for the modified state (B1), complexity will be:

$$K(B1) \leq \log(2+1)\log(2+2+1) = 1.58 \times 2.32 = 3.68$$

Similarly, the change impact for C1 can be calculated as follows:

$$\delta_{C1} = (0 + 2.51) = 2.51$$

After measuring the change impact of new and modified components, based on Equation 5.9, the overall change impact will be:

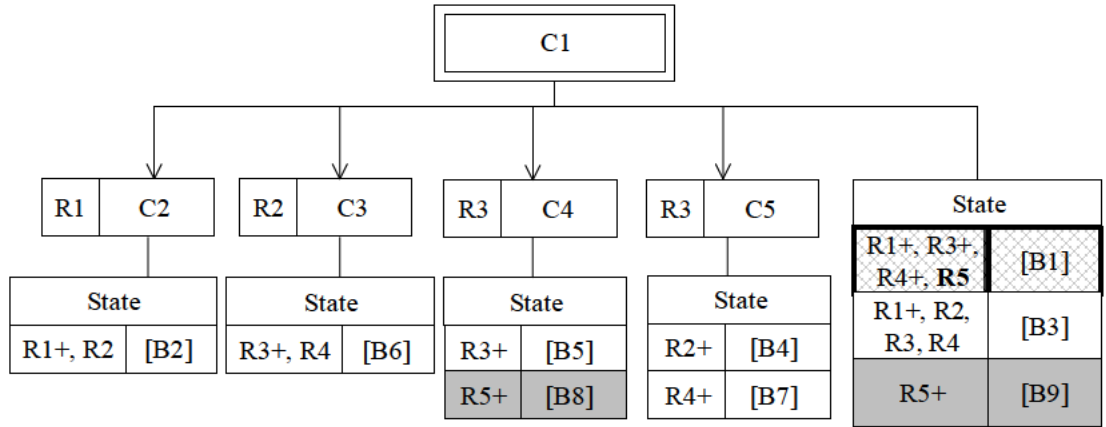
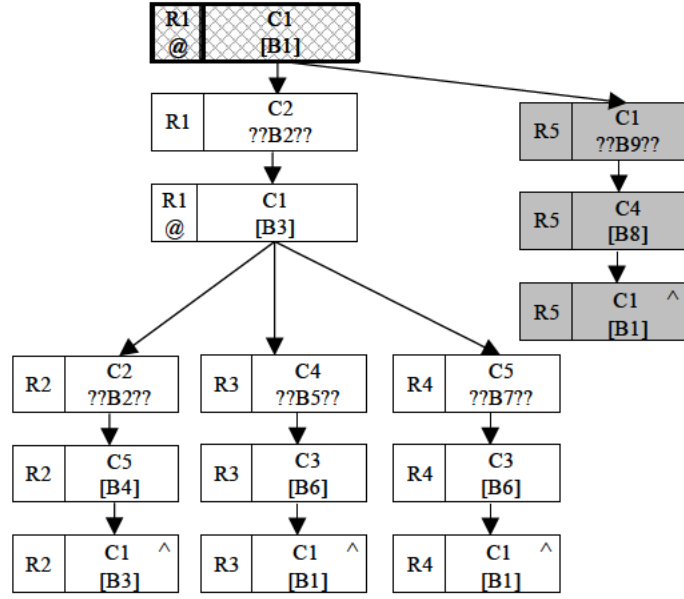
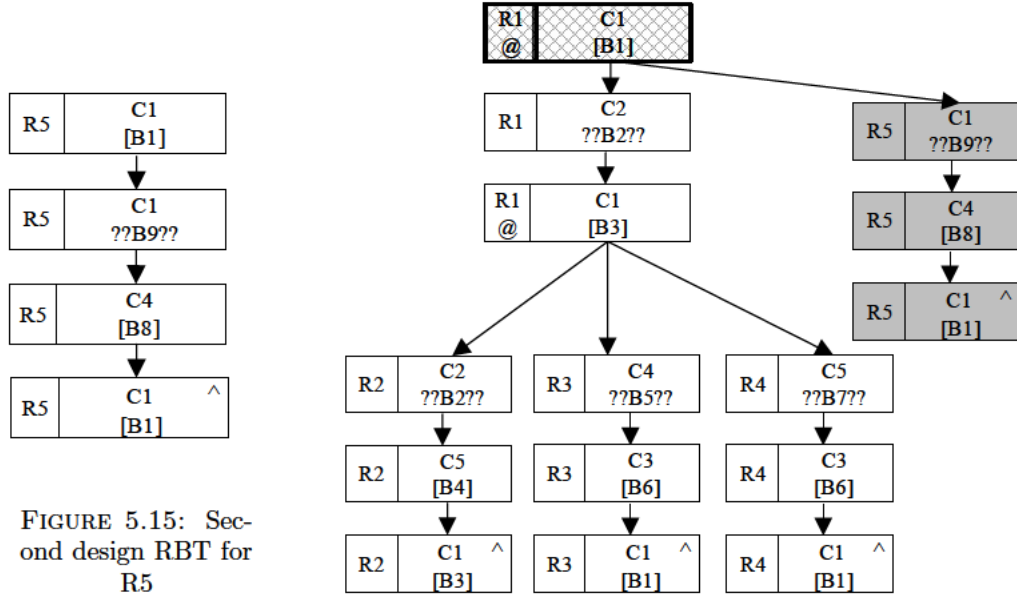
$$\Delta = 1.58 + 2.51 + 3.68 = 7.77$$

This complexity value is on a logarithmic scale in the bits unit. The change impact in terms of complexity will help to calculate more accurate and objective change effort estimation based on historical change benchmarks.

5.3.3 Alternative Design Approach

In software systems, it is possible to have multiple designs based on the same requirements. Therefore, in this subsection, we slightly adjust the new RBT to represent a new design for the change request. In the new RBT, instead of introducing a new component C6 to implement the proposed change, we include the requested functionality in the existing component, which is C4.

And then, we illustrate change impact calculation based on the second design. Due to the space limitations, we only show the diagrams of each step without any explanation and then calculate the final complexity. The RBT for R5, updated IBT, updated ICT and updated RCDN based on the second design are shown in Figure 5.15, Figure 5.16, Figure 5.17, Figure 5.18, respectively.



Now we calculate the change impact for the impacted components. RCDN in Figure 5.18 shows that no new component is introduced, and two existing components are modified due to the proposed change. Based on Equation 5.8, the change impact for component C1 (CID in Figure 5.19) will be:

$$\delta_{C1} = 3.17 + 4.64 = 7.81$$

Similarly, for component C4 (CID in Figure 5.20), the change impact will be:

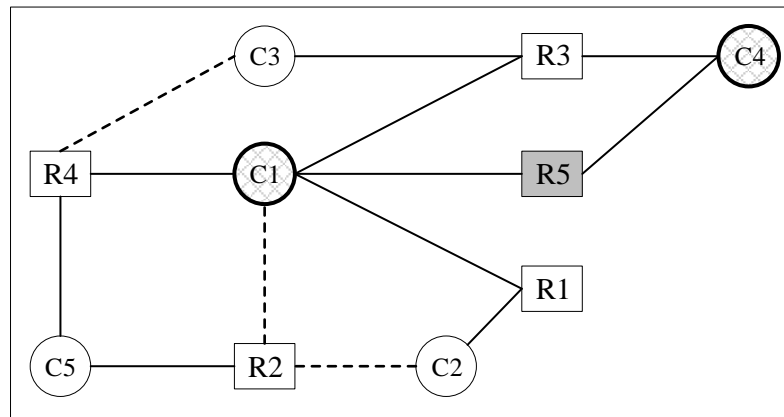


FIGURE 5.18: Updated RCDN for the second design

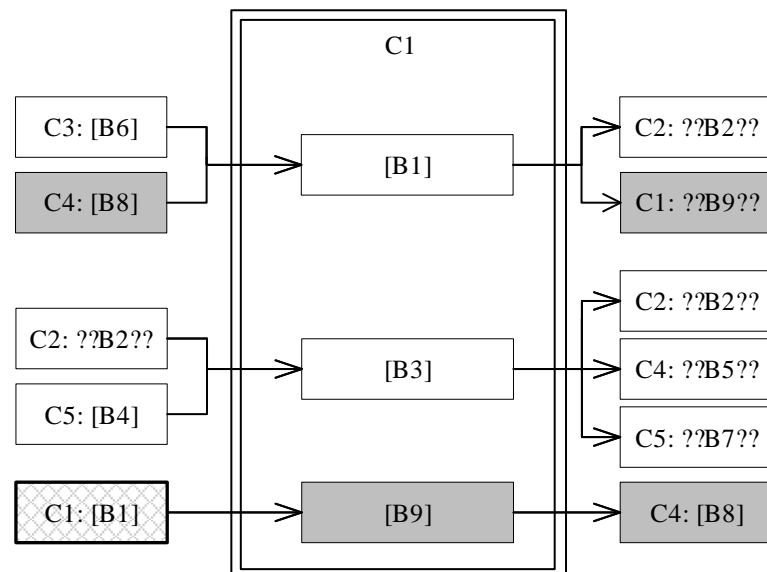


FIGURE 5.19: The CID for component C1 based on the second design

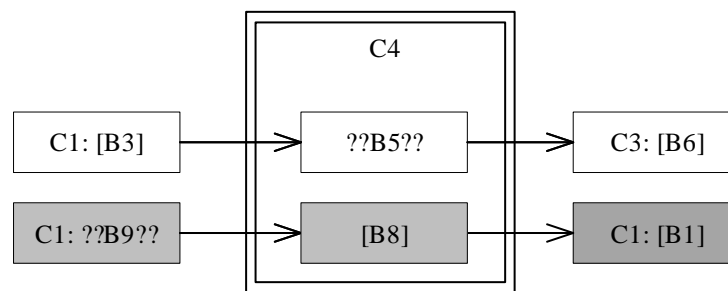


FIGURE 5.20: The CID for component C4 based on the second design

$$\delta_{C4} = 2.51$$

Finally, based on Equation 5.9, the overall impact of this change will be:

$$\Delta = 2.51 + 7.81 = 10.32$$

Regarding the indirect impact, we access the requirements connected to modified components (C1 and C4) based on our four rules. A thorough analysis reveals that we need to check the remaining three components (C2, C3, C5) to investigate the indirect impact. The CID for C2 and C5 is shown in Figures 5.21 and 5.22. The CID for component C3 is the same as in design approach one, which is shown in Figure 5.13. However, no change happens in these three components.

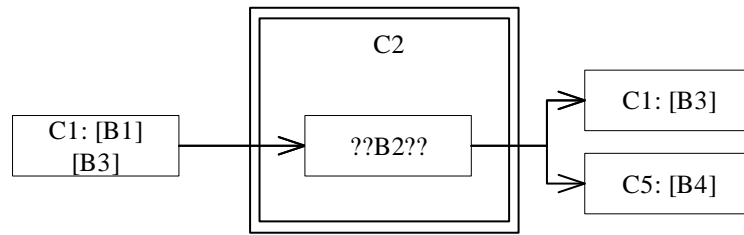


FIGURE 5.21: The CID for component C2 based on the second design

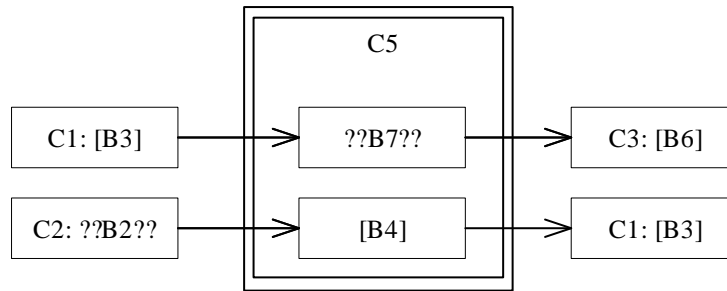


FIGURE 5.22: The CID for component C5 based on the second design

In summary, the overall change impact factor associated with the first design is 7.71, which is less than the change impact factor associated with the second design (10.32). Intuitively, the CII value shows that the proposed change implementation requires less effort in the first design approach as compared to the second design approach. Therefore, regarding the change impact indicator, the first design is a better solution.

5.4 Case Study

This section demonstrates our proposed approach's effectiveness by using a real-world system named Shaanxi Transportation and Logistics Information Standardization Management System (STLISMS). Currently, this system is in the development phase in South Facing Alliance company [223], which is developing a Transport and Logistic Information Service Platform (TLISP) for Shaanxi Province, China. The project is expected to serve more than ten thousand transport companies that provide logistic services in Shaanxi Province, China. STLISMS is an information management system related to transportation and logistics, and it is designed to promote the standards and enforce the implementation of the standards in the relevant projects. STLISMS provides the following major functions:

1. It provides an official portal for customers to search and download all relevant standards.
2. It broadcasts new standards and provides educational materials.
3. All new projects related to the Shaanxi transportation and logistics management system must be registered in STLISMS, and the project plan must be uploaded for review and approval.
4. Each new project must be approved through STLISMS before it can be developed and used.
5. Each project must regularly upload a progress report to ensure the project complies with the relevant standards.

In compliance with the above listed major functions, we collected 18 functional requirements from the users. We take 16 of them as initial requirements and two of them as new requirements emerges due to the proposed change in the system functionality. The initial set of requirements and new requirements is shown in Table 5.2. The new requirements are listed at the end in shaded cells.

TABLE 5.2: STLISMS functional requirements

#	Requirements Description
R1	When the system is idle, a customer can create an account; to create an account, the customer needs to provide relevant information and documents. After that, the account is in the state of new and the system is still in the state of idle.
R2	When the system is idle, a valid customer can log in to the system. After the customer logs in, the system will be in the state of ready.
R3	When the system is in the state of ready, the customer can log out and then the system is in the state of idle.
R4	When the system is in the state of ready, the customer can create a new project. After that, the state of the project is new. The system will be back to ready.
R5	When the system is ready, and a project is new, the customer can provide supporting material; if all supporting materials have been provided, the customer can submit the project. Then the project will be in the state of submitted and the system back to ready.
R6	When the system is ready, the customer can enter search criteria and click search standards; the system will display a list of standards that match the searching criteria.
R7	When the list of standards is displayed, the customer can click on a specific standard, then the content of the standard will be displayed on the screen.
R8	When the system is in the state of ready, and if there is a notification to the customer, the customer can click 'show notification'. Then the notification will be shown to the customer.
R9	When the system is in the state of idle, an admin can input the admin id and password to login. Once an admin is login, the system is in the state of management.
R10	When the system is in the state of management, the admin can log out and then the system is in the state of idle.
R11	When the system is in the state of management, if there is a new customer created, the system will show a customer notification.
R12	When the notification is shown to the customer, the admin clicks the check button, the new customer information will be displayed, and the system will display the new customer.
R13	When the customer information is displayed, the admin can check the customer's detail and if the admin clicks 'approves', the customer's state will be valid, if click rejects, the customer's state will be invalid. Either way, the system will revert to the state of management.
R14	When the system is in the state of management, if there is a new project created, the system will show a project notification.

R15	When the created project notification is shown, if the admin clicks the check button, the new Project information will be displayed.
R16	When the project information is displayed, the admin can check the project's detail, and if the admin click approves, the project's state will be approved; if click pushes back, the project's state will be pushed; if click rejects, the project's state will be rejected. Either way, the system will be back to the state of management.
R17	When the standards' contents are displayed, the customer can select 'download the standard'; then, the standard will be in the state of downloaded. The system will back to a ready state.
R18	When the notification is displayed to the customer, the customer can click 'acknowledge', so the state of notification will be in the state of acknowledged. The system will back to a ready state.

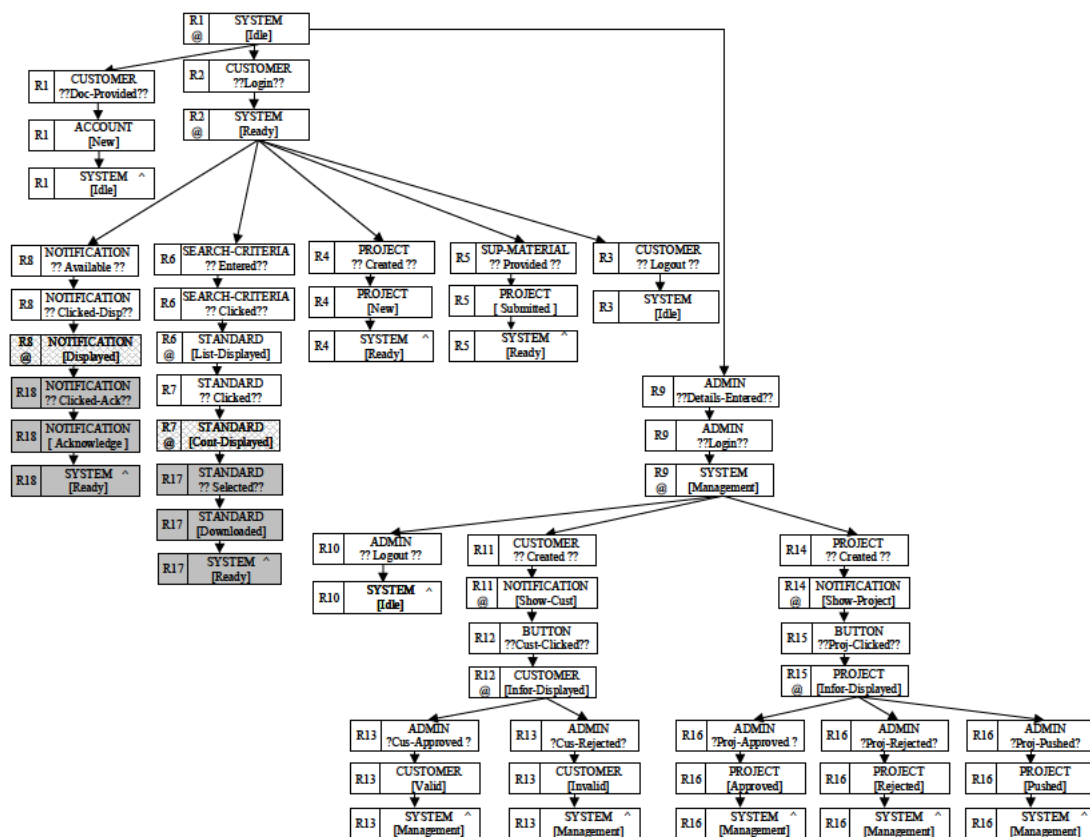


FIGURE 5.23: STLISMS updated IBT

Now, we use our proposed approach to investigate the proposed change impact on other requirements and system artefacts. In steps 2 and 3 of the BECIA workflow, we draw and integrate the new RBTs with the existing IBT but we only show the updated IBT in Figure 5.23. After that, in step 3, we follow a similar approach to the one discussed in

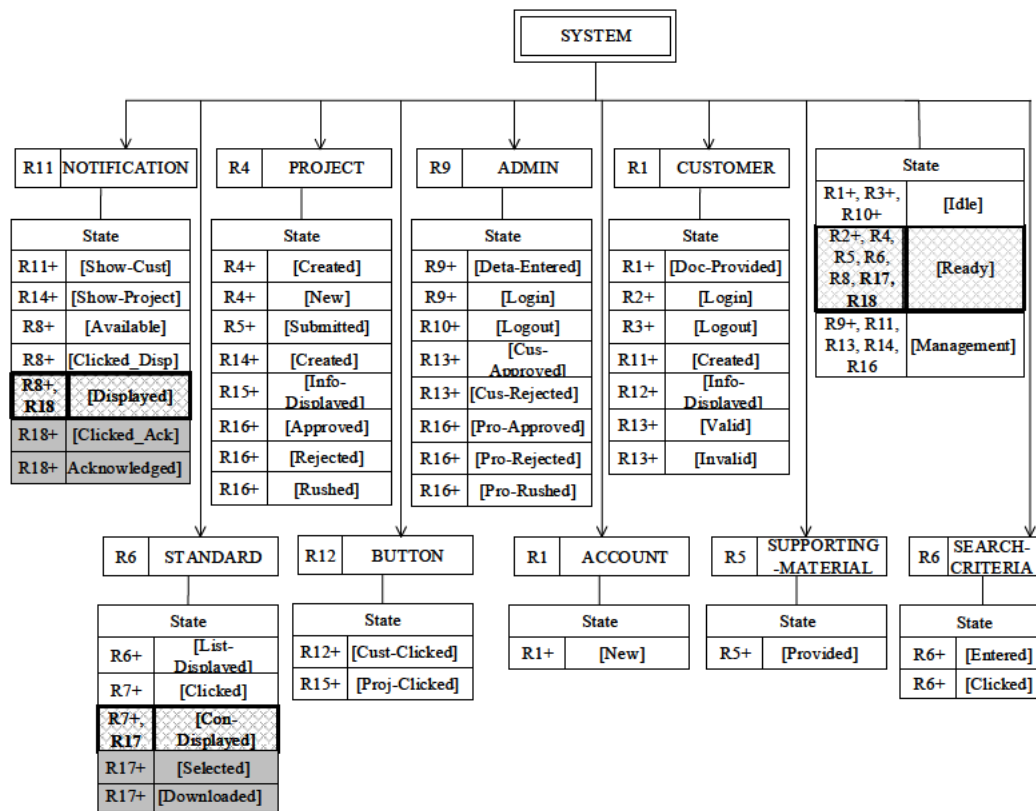


FIGURE 5.24: STLISMS updated ICT

subsection 5.2.1. to update the existing ICT based on the updated IBT, and the updated ICT is shown in Figure 5.24.

After that, in step 5, a similar procedure to the one discussed in subsection 5.2.2 is used to update the RCDN based on the updated ICT. An updated RCDN is shown in Figure 5.25, and the changed requirements and corresponding components are shown in circles and rectangles with different backgrounds. In RCDN, we define an index for each component, such as C1 for SYSTEM, and we will use these indexes in the remaining section.

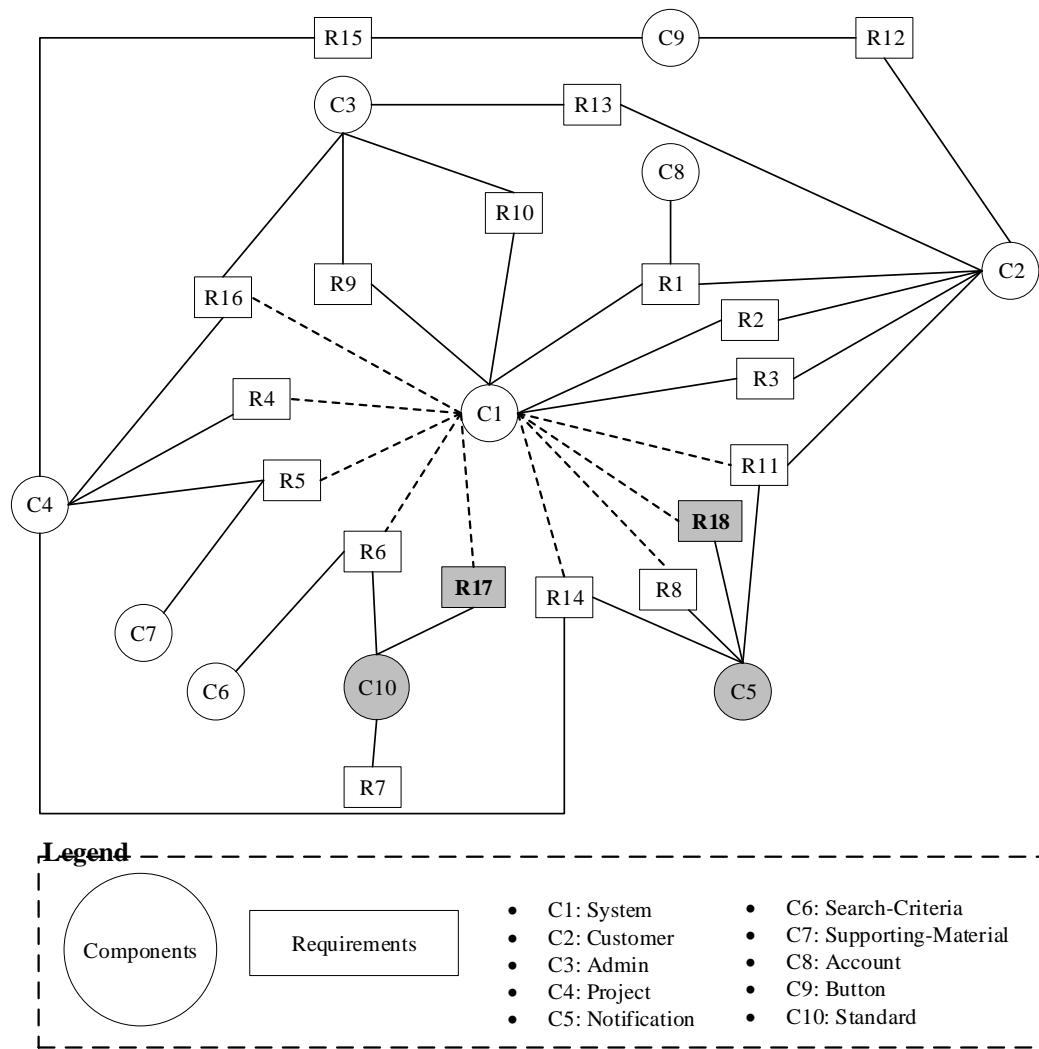


FIGURE 5.25: STLISMS updated RCDN

Now we calculate the impact of the proposed requirements change. RCDN in Figure 5.25 shows that no new component is introduced, and three existing components (C1, C5 and C10) are modified due to the proposed requirements change. Based on Equation 5.4, the complexity of new and modified states for component C1 (SYSTEM) will be:

$$K(C1.B2) \leq \log(3+1)\log(5+4+1) = 6.64$$

Based on Equation 5.8, the change impact for component C1 (CID in Figure 5.26 will be:

$$\delta_{C1} = 6.64$$

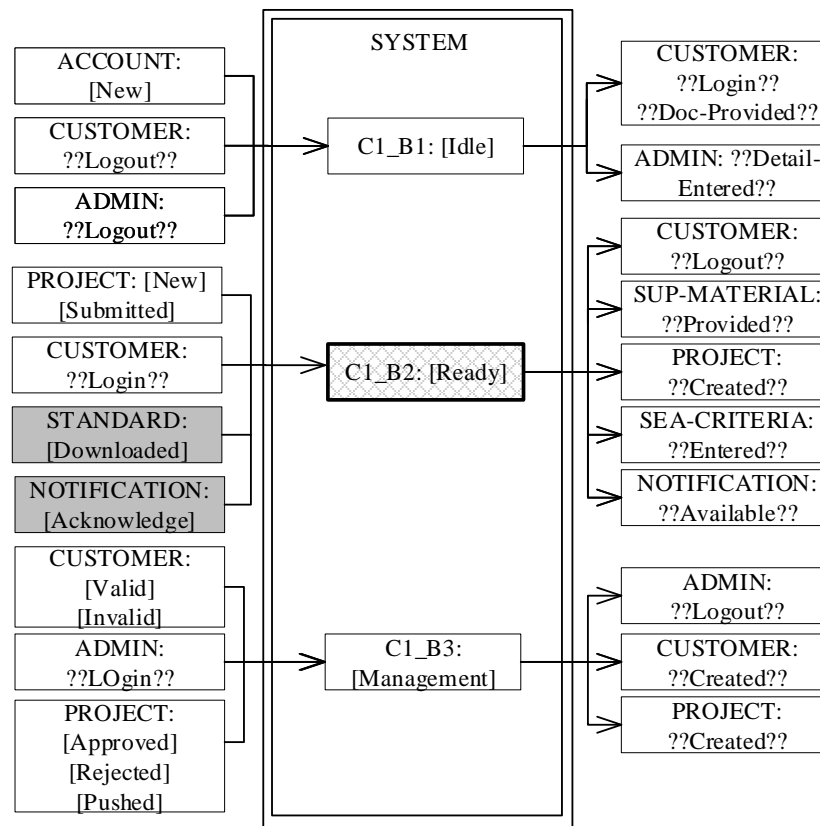


FIGURE 5.26: STLISMS-The CID for C1

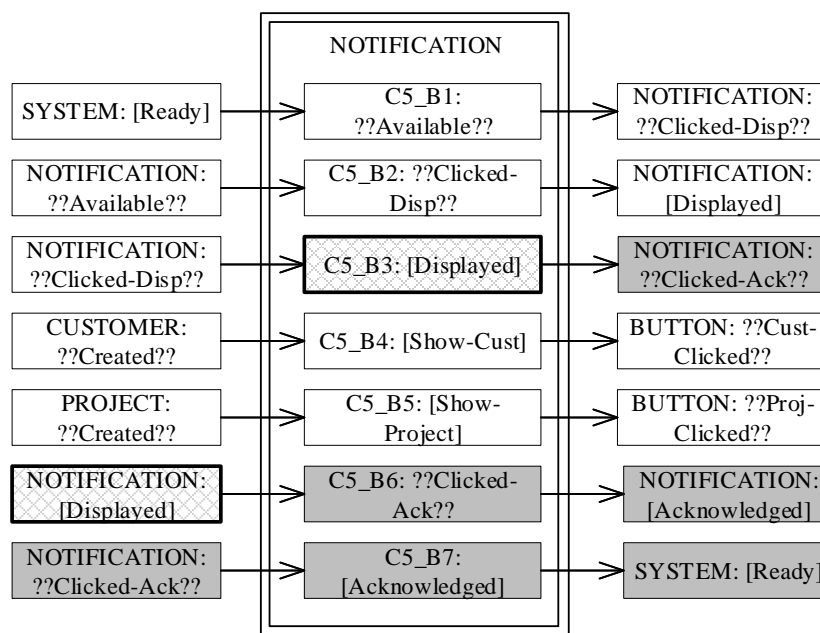


FIGURE 5.27: STLISMS-The CID for C5

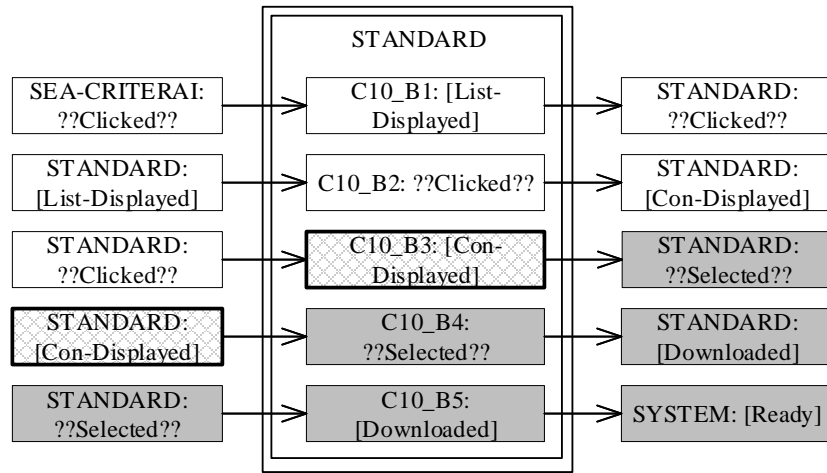


FIGURE 5.28: STLISMS-The CID for C10

An important point related to this calculation is that we calculate complexity after modifying a state such as for C5-B3 because Complexity before modification is less and will be ignored in Equation 5.8.

Similarly, based on Equation 5.4, the complexity of new and modified states for component C5 (NOTIFICATION) will be:

$$K(C5.B3) \leq \log(7+1)\log(1+1+1) = 4.75$$

$$K(C5.B6) \leq \log(7+1)\log(1+1+1) = 4.75$$

$$K(C5.B7) \leq \log(7+1)\log(1+1+1) = 4.75$$

Now, based on Equation 5.8, the change impact for component C5 (CID in Figure 5.27) will be:

$$\delta_{C1} = 4.75 + 4.75 + 4.75 = 14.25$$

The complexity of new and modified states for component C10 (STANDARD) will be:

$$K(C10.B3) \leq \log(5+1)\log(1+1+1) = 4.1$$

$$K(C10.B4) \leq \log(5+1)\log(1+1+1) = 4.1$$

$$K(C10.B5) \leq \log(5+1)\log(1+1+1) = 4.1$$

Based on Equation 5.8, the change impact for component C10 (CID in Figure 5.28 will be:

$$\delta_{C1} = 4.1 + 4.1 + 4.1 = 12.3$$

After calculating the change impact for modified components, now we analyse RCDN based on our defined rules to investigate the indirect impact of the proposed change. RCDN in Figure 5.25 shows that the new requirement (R17) is connected to R6 and R7 by a competing relationship. Therefore, based on rule 1 of requirement change propagation, we will check components connected to R6 and R7, which are C6 and C1. C1 is already checked, and C6 CID (shown in Figure 5.29) investigation reveals that no change happens in C6 due to new requirements R17. Moreover, R17 is connected to R1-R6, R8-R11, R14, R16 and R18 by a supporting and sharing relationship through C1. However, R17 is connected to C1 by a weak connection; therefore, based on rule 3 of requirement change propagation, we will not check the connected requirements and their components in the impact analysis process.

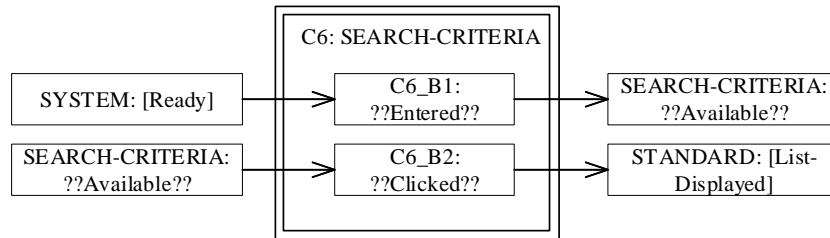


FIGURE 5.29: STLISMS-The CID for C6

Similarly, for the new requirement R18, RCDN in Figure 5.25 shows R18 is connected to R8, R11, and R14 by a competing relationship. Therefore, based on rule 1 of requirement change propagation, we will check components connected to these three requirements, which are C1, C2 and C4. C1 is already checked, C2 and C4 CIDs (Shown in Figure 5.30 and Figure 5.31) reveals that no change happens in C2 and C4 due to new requirement (R18). Moreover, R17 is connected to R1-R6, R8-R11, R14, R16 and R17 by a supporting and sharing relationship through C1. However, R18 is connected to C1 by a weak connection; therefore, based on rule 3 of requirement change propagation, we will not check the connected requirements and their components in the impact analysis process.

No new component is identified for investigation, so we terminate the impact analysis process here.

Finally, based on Equation 5.9, the overall impact of this change will be:

$$\Delta = 6.64 + 14.25 + 12.3 = 33.19$$

This change impact is in terms of KC complexity, and it estimates the descriptive complexity of the proposed change. Along with the comparison of historical data, it could be used to estimate the implementation cost for the proposed change.

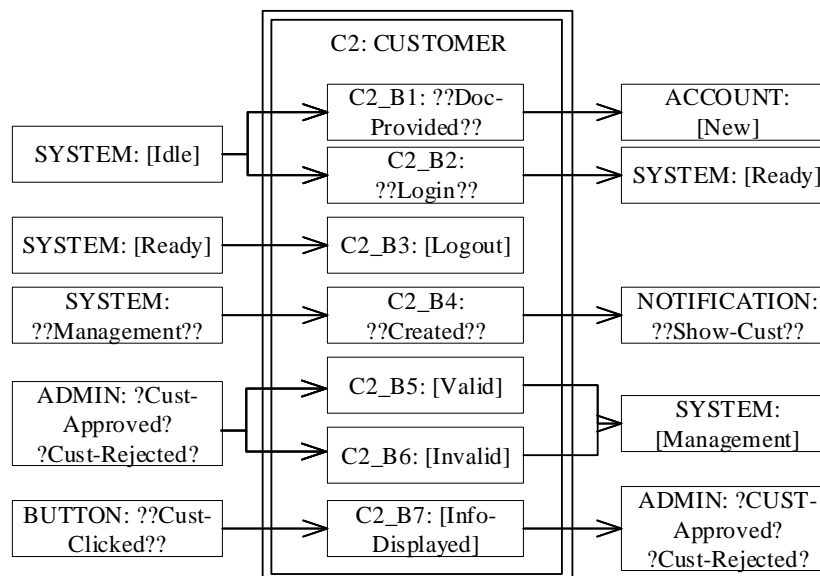


FIGURE 5.30: STLISMS-The CID for C2

5.5 Related Work

Most of the existing research focuses on performing the CIA in the source code compared to other SDLC phases. According to Kretsou [224], 62% of the existing research focuses on performing the CIA in the source code, followed by 22% in the design phase, 14% in architecture, and 2% in requirements. In the context of CIA at various development phases, our approach presents a technique for understanding change impact starting from requirements and moving through to the design & architecture. Therefore, it covers the phases of SDLC that have not been thoroughly explored before.

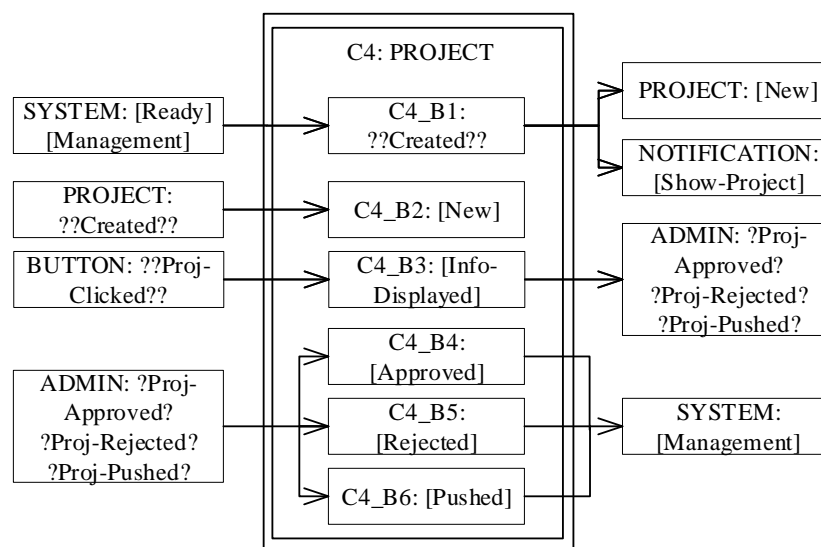


FIGURE 5.31: STLISMS-The CID for C4

Regarding the techniques used to perform CIA, according to Kilpinen [225], there are two main categories of CIA related research, i.e. traceability based and non-traceability method or dependency matrix-based. Also, some studies use a combination of these approaches.

5.5.1 Change Impact Analysis in System Requirements

In traceability approaches, Ibrahim et al. [161] and Li et al. [226] proposed a traceability-based approach to analyse the impact of requested change at the requirements level. They used a traceability matrix and dependency graph to calculate the impact of requirements change on the other requirements.

In another approach, Spilkerman [67] and Goknil et al. [216] proposed different requirements relations to understand the impact of the proposed change. The main contribution of their work is to refine requirements relations, which are captured by a set of impact rules and traceability links. They used formal semantics of requirements relations types that are also used in another study [227].

Some of the existing studies [228, 229] used goal-oriented requirement language (GRL) based approaches to perform impact analysis. Alkaf et al. [230] proposed an automated approach to perform impact analysis by using use case maps and GRL. Similarly, Lee et al.

[231] proposed a goal-driven traceability technique for analysing requirements; they used use cases and the concept of GRL to perform impact analysis at the system requirements level.

Regarding non-traceability techniques, Hassine et al. [232] proposed a CIA approach based on the categorisation of dependencies between use case scenarios. In another study, Ali and Lai [233] proposed an impact analysis approach for global software development projects. They used a dependency matrix to understand the impact of the proposed change on the set of other software requirements. Similarly, Jayatilleke et al. [210] used dependency between functions to analyse the impact of changes on the existing system.

5.5.2 Change Impact Analysis in System Design and Architecture

Briend et al. [234, 235] proposed a CIA technique for an architectural model, which operates on UML models. They designed 97 OCL rules to search impacted elements between different UML models. They used a distance measure to understand the propagation of changes to indirectly related software entities. Similarly, Xing and Stroulia [236] proposed an approach to investigate change impact by analysing the difference between two UML models (class diagrams).

Kchaou et al. [237] proposed a technique to understand the change impact between different UML models. They used a graph technique to model the structural dependencies and used information retrieval to control the semantic traceability between use case documentation and sequence diagrams.

In another study, Feng and Maletic [238] proposed a taxonomy of changes to analyse change impact in component-based architectures. Their approach drives component interaction traces from class and sequence diagrams, which are sliced by impact rules to get the set of impacted entities.

5.5.3 Change Impact Analysis in Source Code

Existing research that used non-traceability based approaches to estimate CIA at source code level can be divided into three major classes: dependency graphs, mining software repositories, and structural information such as coupling.

Regarding the traceability based approaches, Kama and Azli [239] presented a traceability based approach to perform CIA at the source code level. They used horizontal traceability to understand the change in the same version of code and vertical traceability between different source code versions. Similarly, Rahimi and Cleland-Huang [240] presented a traceability approach to establish links between requirements and source code and then used these links to measure the impact of the proposed change at both levels.

Regarding the dependency graph category, Malhotra and Chhabra [241] proposed a CIA approach based on the source code dependencies. Their approach uses seven types of dependencies, including symbol dependency, temporal, include, data, control, semantics, feature, and environment dependencies to calculate change impact. Similarly, Lutillier et al. [242] used include and symbol dependencies and Cafeo et al. [243] used feature dependencies to measure the change impact. In another study, Angerer et al. [36] used control flow and data flow information of the source code and proposed a dependency matrix to understand the change impact.

In mining repositories, Behnamghader et al. [244] proposed a CIA technique based on the commit information of every release to understand the impact of implemented change on other code segments or system parts. Similarly, Dyer et al. [245] developed a query system specifically for mining repositories and performed code analysis to understand the impact of the proposed change. In another study, Ahsan and Wotawa [246] proposed a machine learning-based approach to predict and calculate the impact of requested change on other code segments.

Regarding CIA measurement based on structural information, Beszedes et al. [247] proposed a dynamic coupling function based CIA approach. Their approach investigates the execution sequence by using a dynamic coupling function and then measure the CIA based on this execution information. Similarly, Poshyvank et al. [248] proposed a new coupling matrix named conceptual coupling and measured the impact of the implemented change in the set of other system elements.

5.5.4 Comparison with Existing Work

Analysing the above-discussed approaches for CIA, the following conclusions can be made:

- Regarding change propagation across different software artefacts, most of the existing research uses traceability techniques to establish links among different software artefacts and then uses these links to understand the propagation of the proposed change through different software artefacts. They use different techniques such as dependency graph [226, 237], dynamic slicing [249, 250], distance measure [234, 235], and reverse engineering [251]. Although the traceability based approaches are close to our approach, there are some inherent issues with them.
 - Firstly, it requires time and cost to establish traceability links between different software artefacts [252].
 - Secondly, excessive use of traceability produces more links between artefacts, which are not easy to manage.
 - Thirdly, there is limited support for automatically generating a traceability matrix, which is very important for the effective use of traceability analysis for CIA [253].

On the other hand, our approach offers traceability benefits with the inherent properties of the modelling language, BT, we used to model system requirements [93]. We exploit these properties and propose two algorithms to systematically transform system requirements from one software artefact to another software artefact, such as from IBT to ICT and then ICT to RCDN. This translation maintains traceability links between these artefacts, which helps to demonstrate change propagation between them.

- Regarding identifying the impacted architectural elements, the existing research mostly provides a rough set of candidate elements that might be impacted instead of pinpointing the actually impacted elements such as [34]. On the other hand, our approach uses one newly introduced model, RCDN, to identify a set of candidate elements and then uses CID to identify actually impacted elements. Furthermore, our approach not only identifies the impacted components but, by using CID, also highlights the segments of the models that need to change due to the proposed change.
- Regarding quantifying change impact, most of the existing studies perform CIA without quantifying the change impact. Only a few studies propose some metrics to quantify change impact but they are mostly performed in source code such

as [72, 210]. In contrast, our approach estimates change impact in architecture, identifies the impacted components, and estimates the complexity of the impacted components. The advantage of this approach is that the estimation of individual components complexity will help to assess an accurate and objective measure of the development cost for the proposed change.

- Our approach is fully automatic except for the first step of translating functional requirements into RBTs. All the other steps are based on well-defined rules and processes. Our approach also provides visible and easily verifiable traceability across all different design artefacts.
- Lastly, in a software system, it is possible to have more than one design solution for the same requirements change request. After computing and comparing the CII of each different design solution, it is possible to help the designer find the best design solution.

5.6 Conclusion

In this chapter, we have proposed a novel behaviour engineering-based approach to perform change impact analysis, BECIA. In BECIA, we use IBT to model system requirements and design algorithms to convert an IBT into an ICT, and then to convert an ICT into an RCDN. The RCDN helps to capture relationships between requirements and associated components. It shows which components are connected to which requirements and how requirements are connected through components. Moreover, an RCDN helps to identify a set of potentially impacted components.

After that, we use CIDs retrieved from the IBT and the RCDN to investigate which components are actually impacted due to the proposed changes and the details of the impacts. Lastly, we propose a change impact indicator to quantify the change impact, which will help to identify optimal change solutions among many possible options. It also helps to estimate the development cost for a proposed change based on historical change benchmarks.

Chapter 6

A Formal Model for Behaviour Trees based on Context-Free Grammar

We propose an approach to address one of highly cited RCM challenges (Change impact analysis) identified in Chapter 5. To move on, we choose another important RCM challenge, requirements defects, and propose an approach to address them in this chapter and in Chapter 7.

Modelling languages helps to express natural language system requirements in a structure that is defined by a consistent set of rules and they help to improve the quality of software development. However, the lack of systematic approaches to faithfully translating natural languages into modelling languages limits the applicability of modelling languages. The formalisation of this process is useful for addressing this limitation.

In this chapter, we firstly define normal formed Behaviour Trees (BTs). This normal form works as a template that significantly simplifies the process of translating a requirement from a natural language into its BT representation. We then use Context-Free Grammar (CFG) to verify the formalised structure. This formalisation helps to expose some common requirements defects, such as the incompleteness of individual requirements.

This work further increase the depth of work related to problem 1 by addressing another key RCM challenge i.e. requirements defects, identified in this research. It is important to

note that requirements defects can also arise during requirements elicitation and analysis, so in this context, this is another problem (problem 2) related to the RE phase of SDLC. Therefore, from this perspective, this approach also expands the breadth of our work related to the RE phase of SDLC.

The work introduced in this chapter constitutes a foundation for requirements defects detection and the work is extended in the next chapter.

6.1 Introduction

key activity in requirements engineering is to use different semi-formal modelling languages such as Unified Modelling Language (UML) and BT to model requirements [254]. The visual models capture the precise requirements and are usually easy to understand for both customers and developers [255]. Using semi-formal languages as a bridge to connect software requirements from natural languages to their formal representations offers many advantages. For example, without a bridge, the customers who provide the requirements may find it difficult to understand formal languages, while the engineers may not have the domain knowledge to interpret the customers' requirements correctly [75]. Therefore, it is not easy to verify whether the formal representation is correct.

UML and Behaviour Engineering (BE) are the most widely used modelling languages in the requirements engineering domain. Conventional software engineering approaches, such as UML, construct system designs that will *satisfy* the requirements, while BE approaches such as BT tends to construct a system design *out of* the requirements. This innovative approach allows stakeholders to cross-refer the design elements with the original requirements and guarantees that the design conforms to the minimum criterion to satisfy the requirements [92].

As a powerful notation, BT demonstrates many advantages to capture functional requirements. However, the existing research related to BT formalisation mainly focuses on mapping from BTs to other formal languages but misses the formalisation from natural languages to BTs. The translation from natural languages to BTs is challenging, as the original requirements may contain many defects [256]. The original BT approach tries to translate any arbitrary sentences into RBTs. However, due to human languages' flexibility, natural language requirements could be ambiguous and incomplete and then

produce wrong BTs. If a Requirements Behaviour Tree (RBT) is incomplete and contains mistakes, the Integrated Behaviour Tree (IBT) will be wrong as well [257]. As a result, it will be much more difficult to identify the mistakes and fix them if the defects can't be identified and fixed from the original requirements. The original BTs are semi-formal because they might inherit the incompleteness from the requirements described in natural language, but formalised BTs have enforced completeness in the normal form for RBTs and serve as a formal specification.

Given this need to formalise the translation from natural languages to BTs, we propose a concept of normal formed BTs. The proposed formal model consists of two key elements. Firstly, we define a normal form for RBT based on a new axiom called Inertia Axiom, and then define a valid BT as a normal formed RBT or an IBT integrated through a finite set of normal formed RBTs. Secondly, we develop a CFG that can verify or generate all valid BTs. The developed CFG form a theoretical foundation to verify BT structure and helps to detect and eliminate requirements defects from the beginning. A simple system is used to demonstrate the applicability of our approach. The proposed formal model has successfully identified potential defects present in natural language requirements of the given system.

Moreover, to evaluate the proposed formal model and provide an end-to-end support for requirements analyst, we develop a tool named Behaviour Tree Compiler (BTC) based on the developed CFG. BTC uses iRE which has already been published [104], as a BT editor and take the output from the iRE as input to process the IBT and then verify it by using our proposed CFG. The results show that the proposed grammar has successfully identified all syntactic errors, which were present in the given system and fails to abide normal formed BT rules.

The rest of this chapter is organised as follows: The existing works related to BT formalisation and CFG applications in the software engineering domain are discussed in section 6.2. Section 6.3 introduces the formal model proposed to verify valid BTs, following which, an example to demonstrate the applicability of the proposed model is presented in section 6.4. The developed tool is introduced in section 6.5. Finally, the conclusion is discussed in section 6.6.

6.2 Related Work

This section summarises the existing research related to BT formalisation and CFG applications in the software engineering domain for verification and validation.

Formalisation of BT: In the past, a number of studies have been carried out to formalise BT semantics. For example, Colvin and Hayes [99], and Ahmad et al. [258] proposed the semantics of BT by using CSP and also BTs integration rules. Moreover, BT also has been used for model-checking; for example, Grunske et al. [259, 260] used failure modes and effects analysis to perform model checking and assess the safety and security of software systems. Likewise, Lindsay et al. [261] generated test cases by using BT to verify the completeness and correctness of a BT model. Moreover, a hierarchical component model has been proposed and demonstrated through a BT simulator called BECIE [262]. In another study, Saad et al. [263] translated BT directly to datalog, a formal language and then performed requirements validation.

CFG Applications in Software Engineering: Besova et al. [264] proposed a model transformation approach to improve overall model quality by using CFG graphs. In another study, Damasceno et al. [265] used CFG and Mealy machine to design family models for software product lines. Javed et al. [266] and Chanda et al. [38] used CFG to verify the syntax of two widely used UML diagrams (class and sequence diagram). Similarly, Ruiz-Rude et al. [267], and Manda et al. [268] used grammar techniques to perform static code analysis of domain-specific languages.

In summary, even though some studies have been carried out to formalise BT semantics. However, existing research mostly focuses on formalising requirements models into other formal languages and misses the translation of natural languages requirements into BTs. The translation from natural language into BTs is challenging and may introduce many requirements defects [256]. To address this research gap, we first propose a normal form for RBT to define a valid BT and then develop a CFG that can generate all valid BTs. The valid BTs helps to detect and eliminate requirements defects such as incompleteness from the beginning, which usually arise due to natural language issues such as incompleteness and ambiguity.

6.3 Formal Model

In this section, we define a formal model to validate a valid BT. The model contains two steps. In the first step, we define a normal form for RBT based on the Inertia Axiom, which is also introduced in this research. A valid BT is defined as a normal formed RBT or an IBT integrated through a finite number of normal formed RBTs. In the second step, we propose a set of CFG rules to verify or generate a valid BT (RBT & IBT).

6.3.1 Normal Form of Requirement Behaviour Tree

In this subsection, we define a normal form for RBT based on the Inertia Axiom. The important point is that our normal formed RBT complies with BT taxonomy[96] and preserves the syntax and semantics of BTs. The normal formed BTs are relatively more “formal” than original BTs, and they will help to improve the quality of requirements modelling and help to identify some common requirements defects such as the incompleteness of individual requirements.

Before defining the normal form for RBT, we would like to explain the rationale behind doing that with a simple example. Consider two requirements (R1 and R4) of a microwave oven system, a well-known software engineering example and that has been used in the existing published research [91].

R1: Originally, the oven is in an Idle state, and the Door is closed, and when the button is pushed, the Power-tube will be energised and the oven will start cooking.

R4: Whenever the oven is cooking, or the door is open, the light in the oven will be on.

Figure 6.1 shows the IBT constructed based on these two requirements. A thorough analysis reveals that there is no new action or event described in R4, and it contains only some additional information for R1; therefore, it is not a complete independent requirement.

Sometimes, the end-users tend to describe the result (postcondition) while describing the system requirements without mentioning the event that triggers a system to reach that result. In other situations, a user might also skip preconditions that need to be met so that the later events can be accepted. As a result, a requirement analyst has to add in these missing parts based on their domain knowledge, but it may lead to incorrect

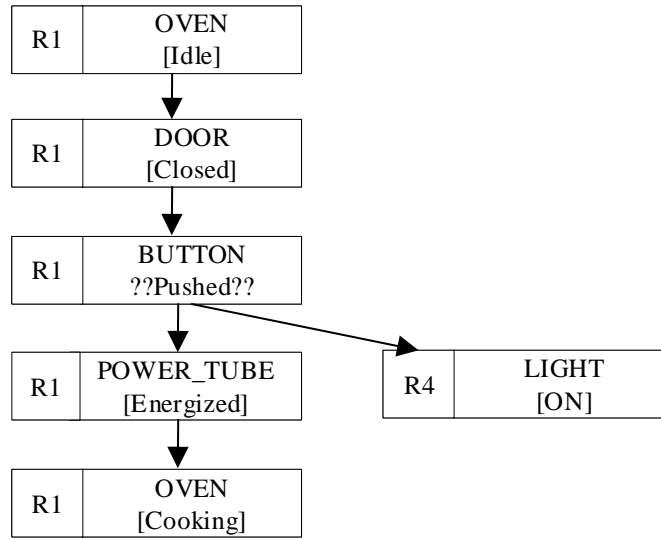


FIGURE 6.1: Oven IBT for (R1 & R4)

interpretation of system requirements. The proposed grammar helps to identify these missing parts by verifying the syntactic correctness of an individual requirement.

Now, we define the normal form for RBTs. The original BT approach introduces two axioms, *Precondition Axiom* and *Integration Axiom* [91]. Here we introduce a third axiom called *Inertia Axiom*,

Definition (Inertia Axiom): *A system will be in a state unless an external or internal event happens to trigger it into a new state, while the new state could be the same as the original state.*

Our proposed axiom is well aligned with software engineering principles. According to this principle, each software requirement should usually be analysed in the context of three key elements: 1, precondition, 2, event, and 3, postcondition. The precondition enforces the conditions that need to be met, so the expected event can be accepted; the consequence of the event is called the postcondition. Based on the above-defined axiom and intuition of an individual software requirement, we define a normal form of RBT.

Normal Formed RBT: *A RBT is called normal formed if and only if it consists of three sequential parts: precondition, event, and postcondition.*

The precondition and postcondition of an RBT must consist of at least one state realisation node. An RBT event is usually a node with the behaviour type as an event, but it could be a node with selection or guard behaviour types.

We take one requirement from the same example to explain the normal formed RBT.

R5: *Whenever the oven is cooking, then opening the door stops the cooking.*

Figure 6.2 (a) shows the RBT for R5. The first node is the precondition. It shows the “OVEN” component is in the state of “Cooking”. The square brackets around “Cooking” indicates the behaviour type as “state realisation”. After that, the second node is the event. In this node, the “DOOR” component is in the state of “Opened”, and the double question marks around “Opened” indicates its behaviour type as an event. Due to the event of “DOOR” ??Opened??. the system gets to the postcondition represented in the third node with the “OVEN” component in the state of “Cooking-Stopped”. The behaviour type is also a “state realisation”. As discussed above, the precondition and postcondition of an RBT must consist of at least one state realisation node. The RBT for requirement (R1) with more than one node in precondition and postcondition is shown in Figure 6.2 (b).

R1: *Originally, the oven is in an Idle state, and the Door is closed, and when the button is pushed, the Power-tube will be energised and the oven will start cooking.*

Inertia axiom aims to formalise an RBT and formalised RBTs can be used to form an formalised IBT. Because an IBT is formed by integrating a set of individual RBTs, as discussed in chapter 2 (2.5.1). If an IBT is formed by integrating a finite set of normal formed RBTs, it is called a normal formed IBT. The concept of normal formed IBT sets up a solid foundation to improve requirements modelling quality and helps to formalise requirements defects. The important point is that our normal formed RBT is a subset of the original RBT; therefore, it complies with BT taxonomy [96] and preserves the syntax and semantics of BT modelling notation.

6.3.2 Context-Free Grammar for Normal Formed BT

In this subsection, we introduce a CFG that can generate all valid BTs. This grammar helps to verify if a BT is valid. Here a BT could be an RBT or an IBT.

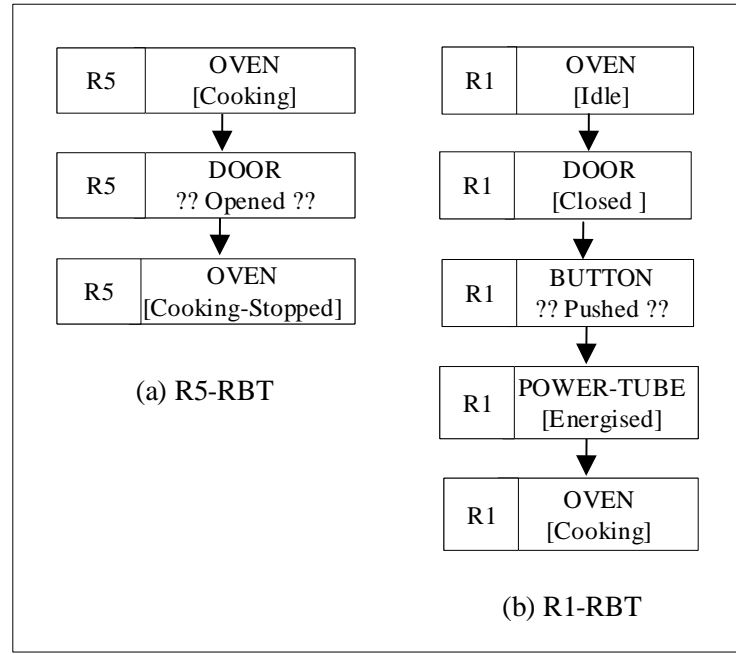


FIGURE 6.2: Normal formed RBT

Let the grammar be in the form of (S, N, T, P) where S, N, T, P represent start symbol, non-terminals, terminals, and production rules, respectively. The non-terminals $N = \{\text{COND}, \text{FL}, \text{EVT}, \text{COND}^\wedge, \text{SR_NODE}, \text{SR_NODE}^\wedge, \text{WH_NODE}, \text{SEL_NODE}, \text{GRD_NODE}, \text{BN}, \text{CN}, \text{RT}\}$, and terminals $T = \{\text{CNS}, \text{BNS}, \text{RTS}, \text{KEYS}\}$, where CNS is the set of all component names, BNS is the set of all behaviour names, RTS is the set of all requirement tags, and KEYS are some special key characters including “.”, “(”, “)”, “{”, “}”, “[”, “]”, “?”, and “^”. We used “.” to show connection between two nodes. The following two CFG rules are used to generate a valid IBT structure with a minimum number of nodes. The symbol of “ ϵ ” means termination with an empty string.

$$S \longrightarrow \text{COND} \cdot \text{FL} (\text{EVT} \cdot \text{COND} \cdot \text{FL}) \mid \text{COND} \cdot \text{FL} (\text{EVT} \cdot \text{COND}^\wedge) \quad (6.1)$$

$$\text{FL} \longrightarrow \text{FL} (\text{EVT} \cdot \text{COND} \cdot \text{FL}) \mid \text{FL} (\text{EVT} \cdot \text{COND}^\wedge) \mid \epsilon \quad (6.2)$$

Where S, COND, EVT, and FL are non-terminals and indicate start symbol, condition, event, and flow type, respectively. We use the same non-terminal COND to show precondition and postcondition of a requirement because during integration of RBTs to form an IBT, the postcondition of one RBT may be the precondition of another RBT. The EVT

indicates an event that can belong to one of the three event types discussed in subsection 2.5.1, and corresponding rules will be defined later in this section.

The parenthesis in production rules is used to combine nodes that belong to the same requirement. The FL indicates the sequence of nodes that can be organised either in sequential or branching flow. In production rule (6.1), the FL after the first instance of COND is used to organise nodes in a way that both branches will share the same precondition. The FL after the second instance of COND is used to organise nodes in sequential form, and postcondition of an existing branch will become a precondition of a new branch. The simple sequence with two nodes (event, condition) either belongs to an individual requirement or forms a part of a requirement. The COND^\wedge exhibits a reversion node.

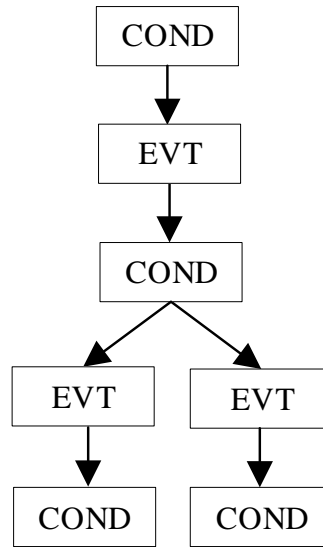


FIGURE 6.3: An abstract IBT example

We use one abstract BT example to demonstrate how these two rules will help to verify or generate a valid BT structure. Figure 6.3 shows an abstract BT structure, and Figure 6.4 displays the corresponding parse tree to generate that BT. The COND and EVT non-terminals show abstract condition, and an event, respectively and the production rules to define these non-terminals will be discussed later in this section.

We concatenate \cdot with the preceding non-terminal to make a simple diagram. The parse tree starts with a start symbol S, which is replaced with the production rule (6.1). After that, the two instances of non-terminal FL are replaced with different options of the

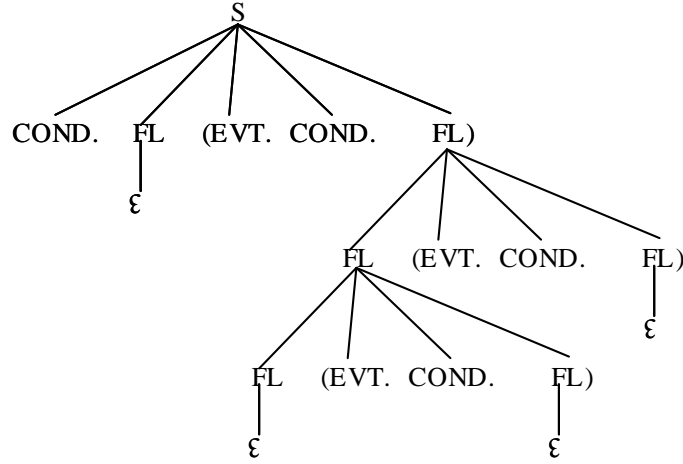


FIGURE 6.4: Parse tree for an abstract IBT

production rule (6.2). By following LR parser, we first read **COND**, then start parenthesis, which shows the start of flow followed by an **EVT**, and a **COND**. After that, we read the start parenthesis again to start another flow, followed by another **EVT**, **COND**, and an end parenthesis, which terminates the closest started flow. After that, we read another flow with the same pattern. An important point here is that both flows become two separate branches with the same precondition, as shown in Figure 6.3. Lastly, we read the end parenthesis to end the overall tree. The resulting sentence will generate an IBT structure equivalent to one which is shown in Figure 6.3.

Now we introduce the remaining production rules in our CFG as follows:

$$\text{COND} \longrightarrow \text{SR_NODE} \cdot \text{COND}' \quad (6.3)$$

$$\text{COND}' \longrightarrow \text{SR_NODE} \cdot \text{COND}' \mid \epsilon \quad (6.4)$$

$$\text{COND}^\wedge \longrightarrow \text{COND}' \cdot \text{SR_NODE}^\wedge \quad (6.5)$$

$$\text{SR_NODE} \longrightarrow \{\text{CN}, [\text{BN}], \text{RT}\} \quad (6.6)$$

$$\text{SR_NODE}^\wedge \longrightarrow \{\text{CN}, [\text{BN}], \text{RT}, ^\wedge\} \quad (6.7)$$

$$\text{EVT} \longrightarrow \text{WH_NODE} \mid \text{SEL_NODE} \mid \text{GRD_NODE} \quad (6.8)$$

$$\text{WH_NODE} \longrightarrow \{\text{CN}, ??\text{BN}??, \text{RT}\} \quad (6.9)$$

$$\text{SEL_NODE} \longrightarrow \{\text{CN}, ?\text{BN}?, \text{RT}\} \quad (6.10)$$

$$\text{GRD_NODE} \longrightarrow \{\text{CN}, ???\text{BN}???, \text{RT}\} \quad (6.11)$$

$$\text{CN} \longrightarrow \text{cn}; \text{ where } \text{cn} \in \text{CNS} \quad (6.12)$$

$$\text{BN} \longrightarrow \text{bn}; \text{ where } \text{bn} \in \text{BNS} \quad (6.13)$$

$$\text{RT} \longrightarrow \text{rt}; \text{ where } \text{rt} \in \text{RTS} \quad (6.14)$$

Where a COND can be both a precondition and a postcondition, an SR_NODE indicates a state realisation node. COND' is used to get more than one node in a precondition and a postcondition. The BT node consists of three elements, behaviour name, behaviour type, and requirement tag. We used a pair of curly brackets {} to display one node with all these elements. The CN, BN, and RT are non-terminals and would be substituted with cn, bn, and rt, which are elements of the set of component names, the set of behaviour types, and the set of requirement tags, respectively. The behaviour types are already discussed in subsection 2.5.1, and we mentioned behaviour types with corresponding symbols instead of writing in strings such as [...] for state realisation.

Now we will demonstrate the working of all these rules with a simple example, a Security Alarm System (SAS), with only two requirements. Both requirements of the SAS are listed below, and the corresponding RBTs are shown in Figure 6.5(a) and 6.5(b).

R1: The SAS is activated by pressing the SET-BUTTON.

R2: The SAS is deactivated once the three-digit code is entered.

The two RBTs are integrated to form an IBT, which is shown in Figure 6.5(c). After that, by using our CFG rules, we designed the parse tree, which is shown in Figure 6.6.

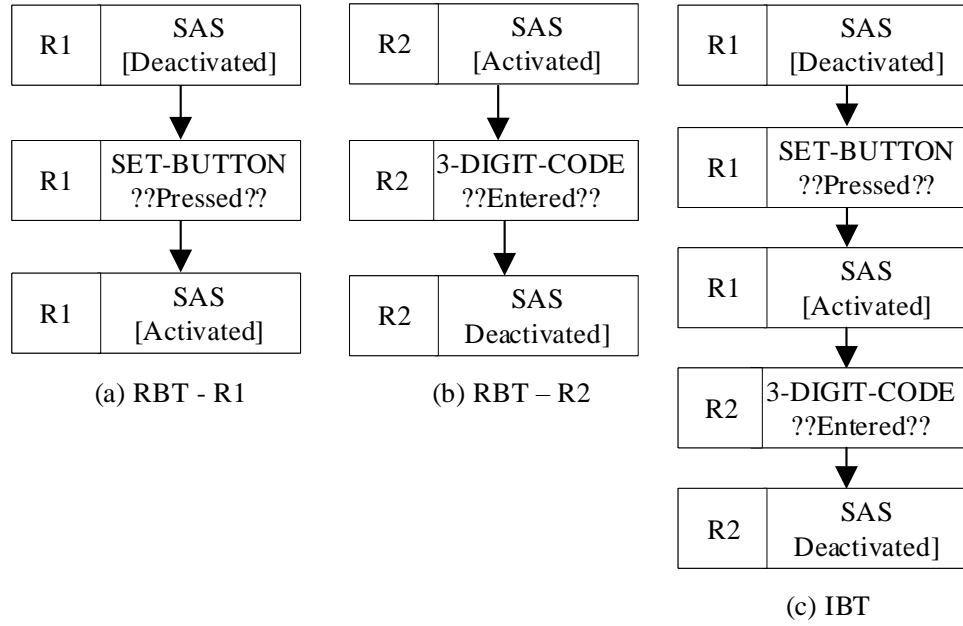


FIGURE 6.5: An IBT for SAS

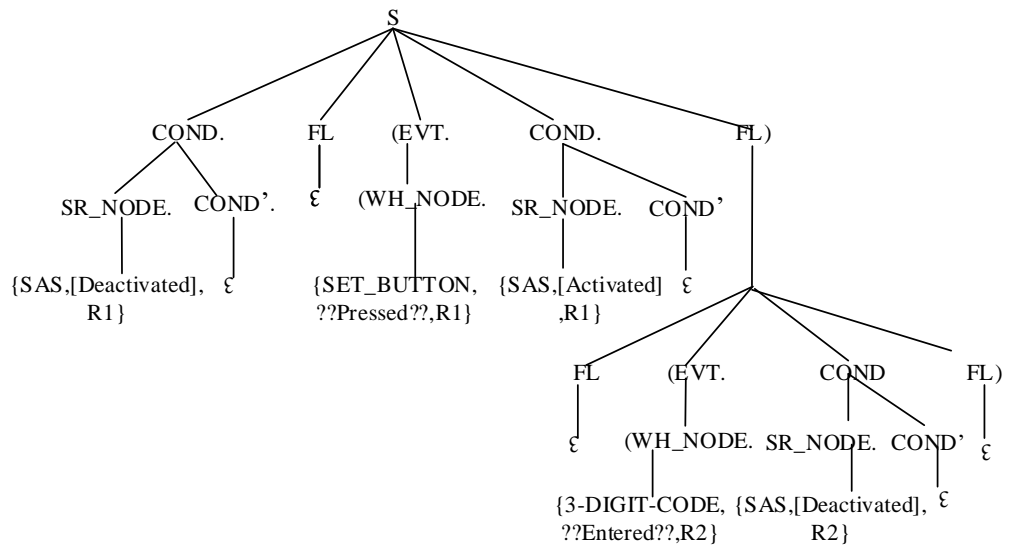


FIGURE 6.6: The parse tree for SAS IBT

By following a similar approach to that taken to read the parse tree, the sentence is given below:

$$\begin{aligned} & \text{SR_NODE} \cdot (\text{WH_NODE} \cdot \text{SR_NODE} \cdot (\text{WH_NODE} \cdot \text{SR_NODE})) \\ & \{ \text{SAS}, [\text{Deactiavted}], \text{R1} \} \cdot (\{ \text{SET} - \text{BUTTON}, ??\text{Pressed}??, \text{R1} \} \cdot \\ & \{ \text{SAS}, [\text{Activated}], \text{R1} \} \cdot (\{ 3 - \text{DIGIT} - \text{CODE}, ??\text{Entered}??, \text{R2} \} \cdot \{ \text{SAS}, [\text{Deactiavted}], \text{R2} \})) \end{aligned}$$

This sequence of nodes will generate an equivalent IBT, as shown in Figure 6.5(c). By analysing the SAS requirements described in natural language, we can find that the system state (precondition) required before an event is missing in both requirements. However, the proposed grammar helps to identify this missing part by enforcing a minimum number of single RBT elements.

6.4 An Example

This section uses an example to demonstrate the working of the proposed formal model. We choose the microwave oven system with 7 requirements listed in Table 6.1 and the corresponding IBT is shown in Figure 6.7. This IBT is reproduced from previously published research [91].

TABLE 6.1: Functional requirements of microwave oven

#	Requirement Description
R1	Originally, the oven is in Idle state and the Door is closed and when the button is pushed, the Power-tube will be energised the oven will start cooking.
R2	If the button is pushed while the oven is cooking it will cause the oven to cook for an extra minute.
R3	Pushing the button when the door is opened has no effect (because it is disabled).
R4	Whenever the oven is cooking or the door is opened the light in the oven will be on.
R5	Whenever the oven is cooking, opening the door will stop the cooking.
R6	Whenever the oven is open, then closing the door turns off the light. This is the normal idle state, prior to cooking when the user has placed food in the oven.
R7	If the oven times-out the light and the power-tube are turned off and then a beeper emits a sound to indicate that the cooking is finished.

A thorough analysis of the given IBT reveals that R3 and R4 do not follow the normal formed RBT structure. Neither requirement is well-defined, and they are incomplete

because they have just provided some extra information on other requirements. The parse tree generated from the microwave oven original requirements is shown in Figure 6.8, and we have only shown R6 and R3 nodes to illustrate the potential defect in R3. A similar problem is associated with R4. The given parse tree shows that we could not achieve a final sentence with all terminals in the leaf nodes due to the defect in R3. One non-terminal WH_NODE, circled in the rectangle, shows that R3 is not a complete, independent requirement.

Although the given example has been used in many existing studies, when it is checked against our CFG, some issues are discovered, and it does not in normal form because these two requirements (R3 and R4) were modelled as individual requirements. In this research, we address this issue to generate quality or a valid IBT.

Now, we will modify the oven requirements based on the normal formed RBTs. According to the defined normal form, a normal formed requirement should contain a precondition, event, and postcondition. The absence of precondition or postcondition shows the incompleteness of an individual requirement and this incompleteness can be derived from an event. However, the absence of an event shows that the given requirement is not well-defined and should be further analysed and merged with an event relevant to the described information. R3 and R4 have the same issue (absence of event); therefore, further analysis is required to make them complete.

After thorough analysis, we merged these two requirements with other relevant requirements, and the modified requirements for the microwave oven system are shown in Table 6.2, and the new corresponding IBT is shown in Figure 6.9. To trace the modified requirements back to the original requirements, the mapping between the new set of requirements and the original set of requirements is shown in Table 6.3.

Now, we will demonstrate how our proposed grammar can generate the sentence equivalent to the IBT in Figure 6.9 by following the same approach to read the parse tree shown in Figure 6.10. The sentence given-below is equivalent to the IBT shown in Figure 6.9.

{R4', OVEN, [Open]}·({R4', DOOR, ??Closed??}·{R4', BUTTON, [Enabled]}·{R4', LIGHT, [Off]}·
 {R4', OVEN, [Idle]}·{R1', DOOR, [Closed]}·({R1', BUTTON, ??Pushed??}·{R1', LIGHT, [On]}·
 {R1', POWER – TUBE, [Energised]}·{R1', OVEN, [Cooking]}·({R2', BUTTON, ??Pushed??}·
 {R2', OVEN, [Extra_Minute]}·{R2', OVEN, [Cooking]}^)({R3', DOOR, ??Opened??}·

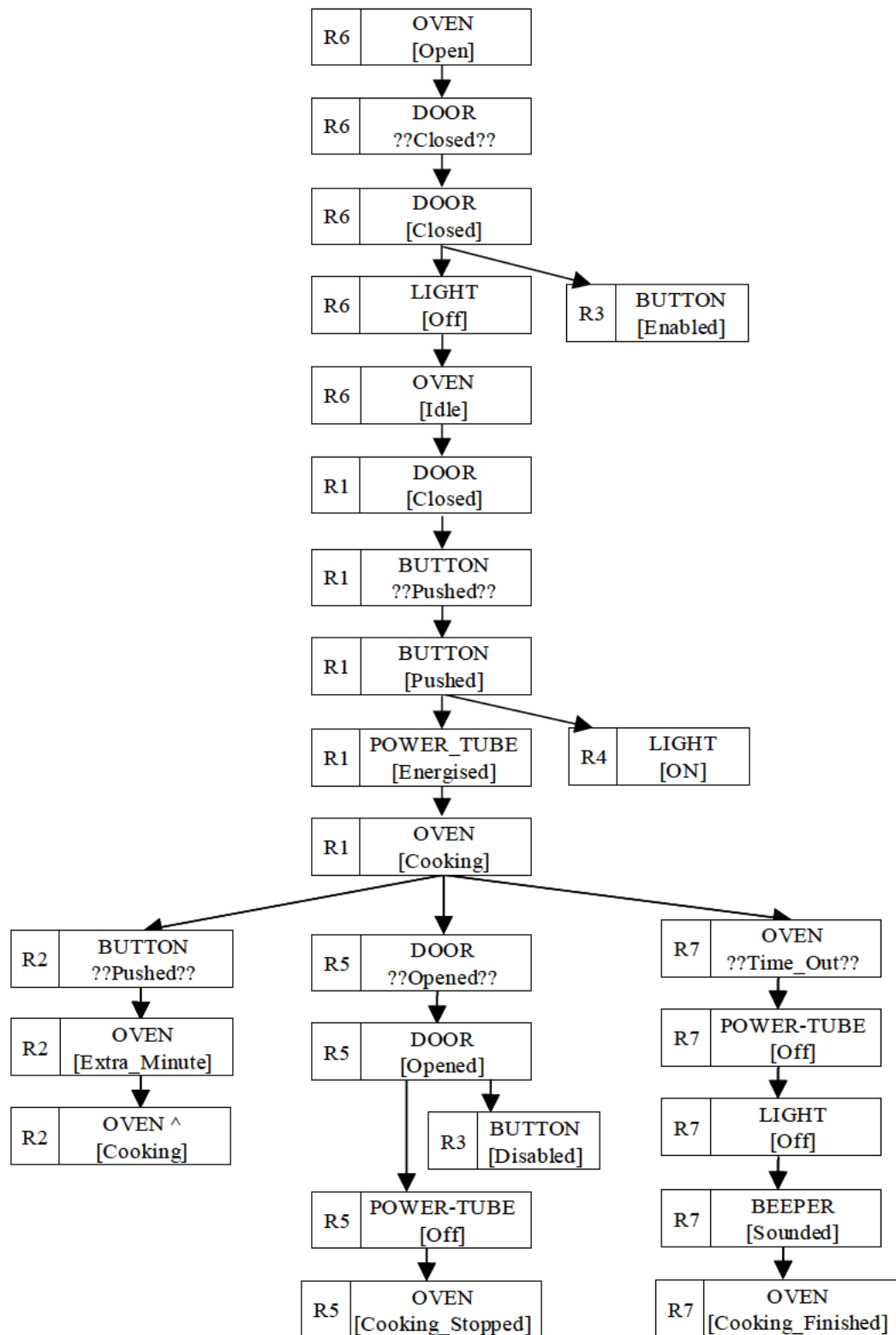


FIGURE 6.7: An IBT for original requirements of microwave oven

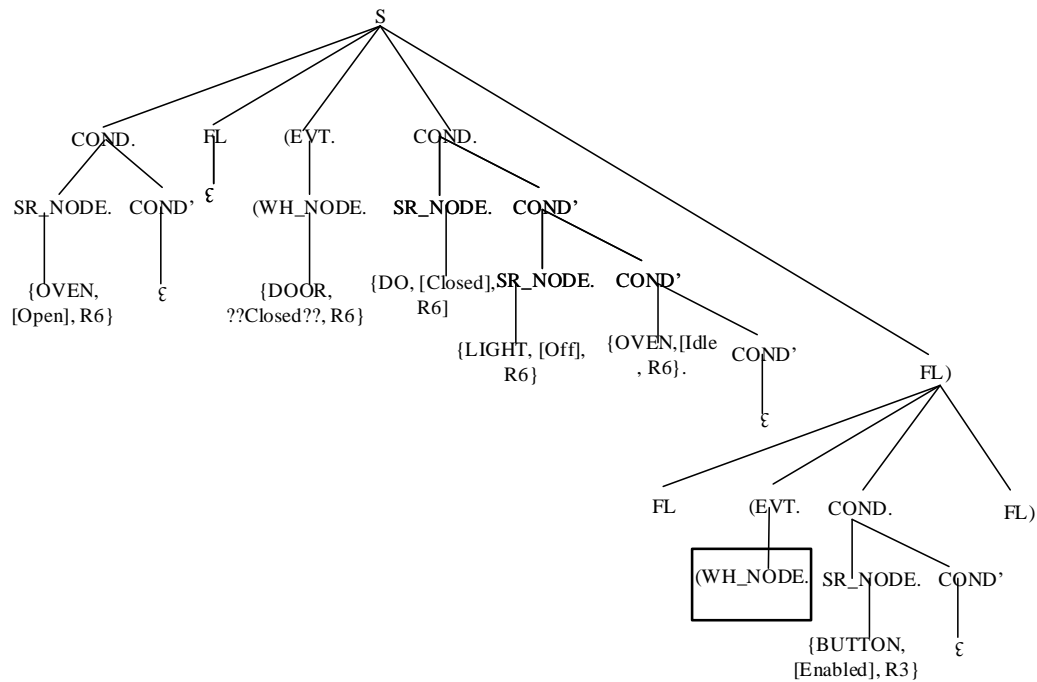


FIGURE 6.8: The parse tree for microwave oven original IBT

TABLE 6.2: Modified functional requirements for microwave oven

#	Requirement Description
R1'	If the oven is idle with the door is closed and you push the button, the light will turn on and the oven will start cooking (that is, energise the power-tube for one minute).
R2'	If the button is pushed while the oven is cooking it will cause the oven to cook for an extra minute.
R3'	Opening the door disables the button and stops the cooking.
R4'	Closing the door enables the button and turns off the light. This is the normal idle state, prior to cooking when the user has placed food in the oven.
R5'	If the oven times-out the light and the power-tube is turned off, and then a beeper emits a sound to indicate that the cooking is finished.

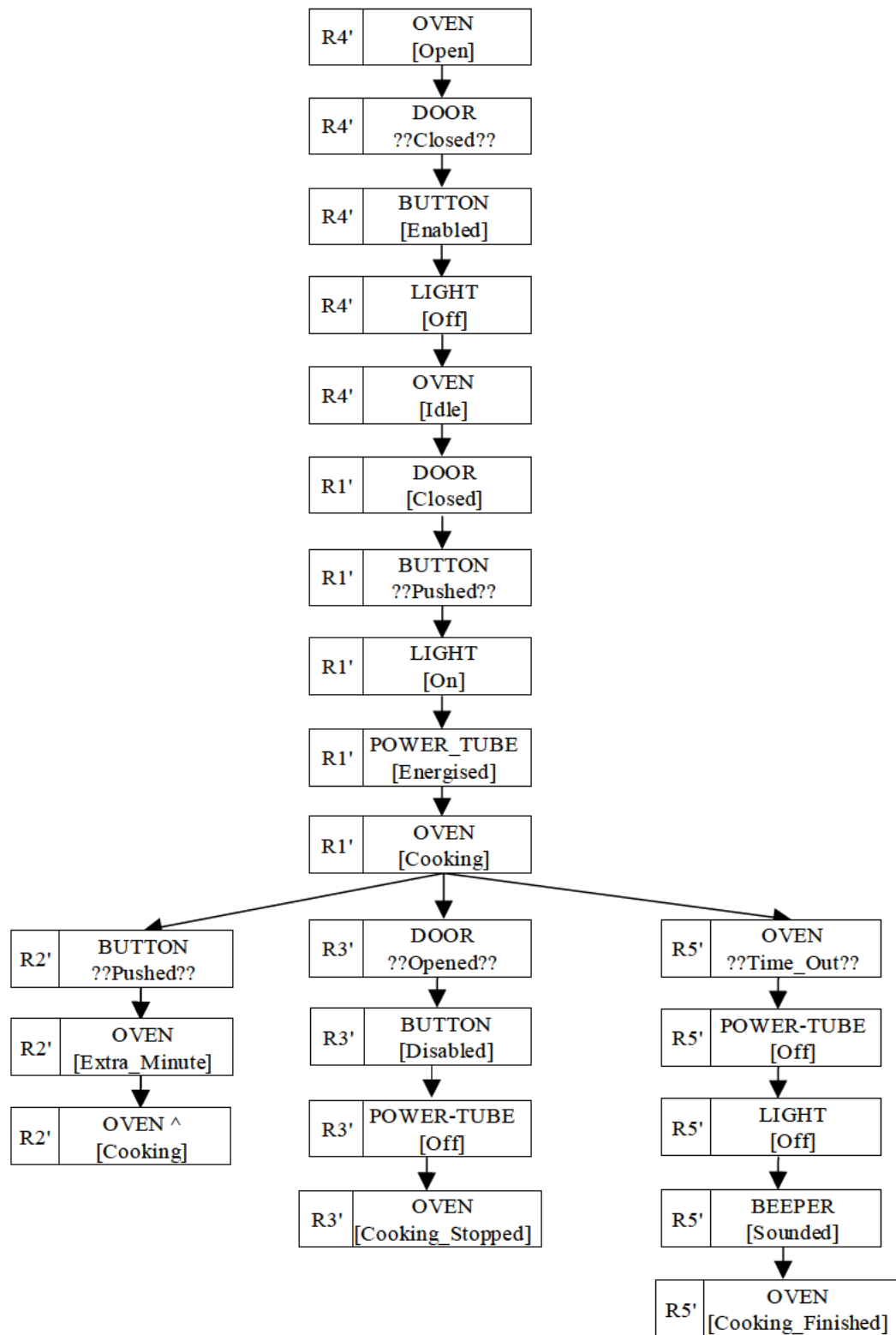


FIGURE 6.9: The modified IBT for microwave oven

TABLE 6.3: Mapping between original and modified functional requirements for microwave oven

New Requirements Tags	Original Requirements Tags
R1'	R1, R4
R2'	R2
R3'	R3, R5
R4'	R4, R6
R5'	R7

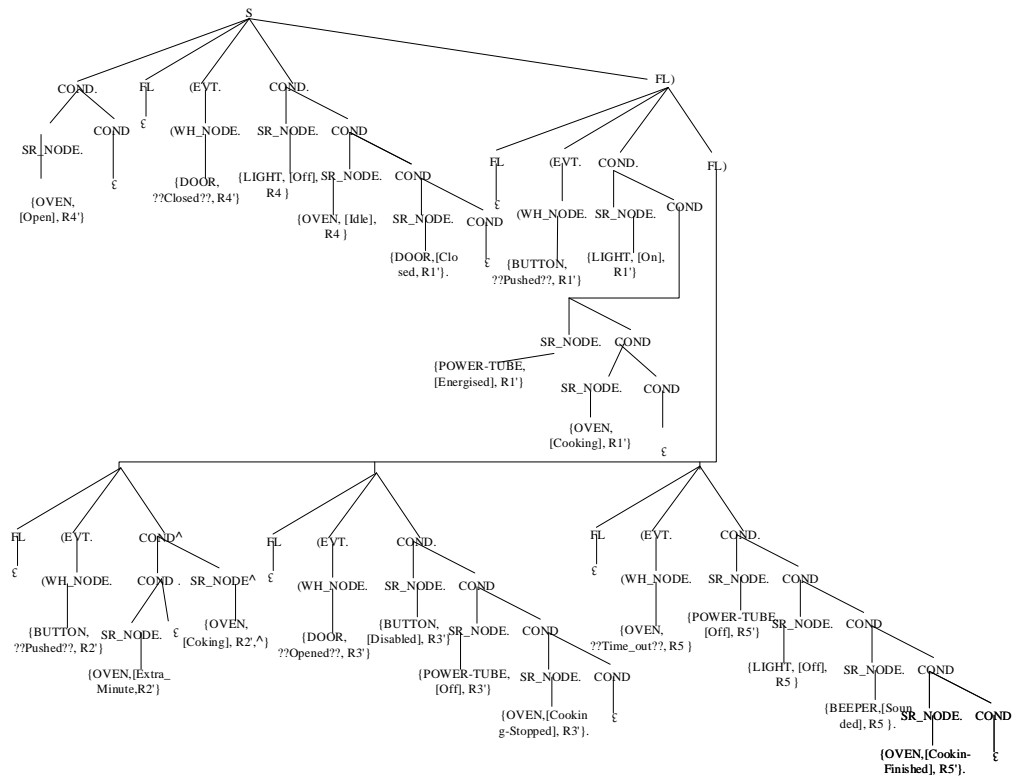


FIGURE 6.10: The parse tree for microwave oven modified IBT

$\{R3', \text{BUTTON}, [\text{Disabled}]\} \cdot \{R3', \text{POWER} - \text{TUBE}, [\text{Off}]\} \cdot \{R3', \text{OVEN}, [\text{Cooking_Stopped}]\})$
 $(\{R5', \text{OVEN}, ??\text{Time_Out}??\} \cdot \{R5', \text{POWER} - \text{TUBE}, [\text{Off}]\} \cdot \{R5', \text{LIGHT}, [\text{Off}]\} \cdot$
 $\{R5', \text{BEEPER}, [\text{Sounded}]\} \cdot \{R5', \text{OVEN}, [\text{Cooking_Finished}]\})))$

The proposed formal model has a number of applications related to requirements engineering and software requirements modelling. Firstly, valid BTs helps to identify the most common requirement defects, such as incompleteness, by ensuring individual requirements' syntactic correctness. Secondly, the proposed formal model can be applied in both the initial requirement's analysis phase and during requirements change management.

6.5 Tool-BT Compiler

In this section, we discuss the tool named BT Compiler (BTC), which is developed to implement the proposed CFG and verify the IBT structure generated, based on the CFG. Generally, a CFG can be used in two different ways: firstly, to generate sentences through the CFG rules, and secondly, to verify if a sentence complies with the CFG rules. BTC is based on the second way and will help to verify whether a given string corresponds to a valid BT.

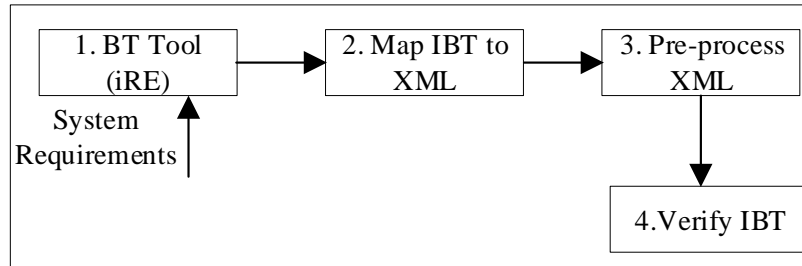


FIGURE 6.11: BTC workflow

Figure 6.11 shows the BTC workflow in four key steps. The first two steps are already implemented in the published research, and the last two steps are our main contribution in this tool. In the first step, a BE tool called iRE, which has already been published, is used to design IBT based on the system requirements. In next the step, the IBT is mapped to an XML representation by using the same tool. In step 3, the XML representation is pre-processed for converting into a format that can be used as the input for the next step. In the last step, the IBT will be verified by using the CFG rules.

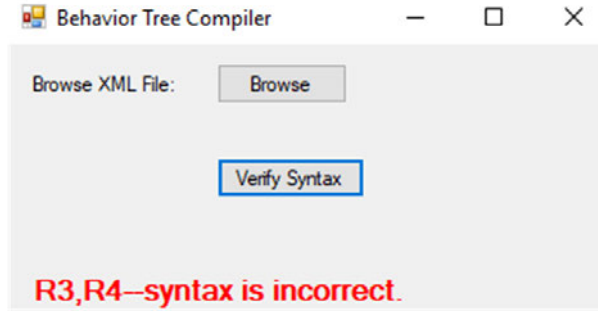


FIGURE 6.12: BTC output for original requirements of microwave oven

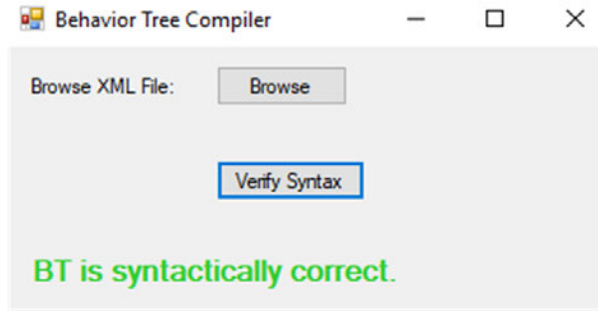


FIGURE 6.13: BTC output for modified requirements of microwave oven

The BTC takes an input of an IBT in an XML format, which is the output of iRE and outputs either successful verification of the given IBT or information about all syntactically incorrect RBTs. Figure 6.12 shows the BTC output corresponding to the original IBT shown in Figure 6.7. Figure 6.13 shows the BTC output for the modified IBT, which is shown Figure 6.9.

6.6 Conclusion

BT is one of the modelling languages and has proved useful through numerous studies and industry cases. There have been several attempts to formalise the translation from BTs to other formal languages. However, there is a need to formalise the translation from natural languages to BTs to overcome some drawbacks in natural languages.

To address this research gap, we proposed a two-step formal model to generate valid BTs. Firstly, we define normal formed RBTs based on a newly introduced Inertia Axiom and then define valid BT as a normal formed RBT or an IBT integrated from a finite set of normal formed RBTs. Secondly, we use CFG as a formal grammar to generate and verify valid BTs. This chapter uses microwave oven, a commonly used example in the software

engineering domain to demonstrate the applicability of the proposed method. Moreover, a tool has been developed to support and validate the proposed formal model. The results show that the proposed grammar has successfully identified all syntactic errors in the original requirements of the example and those requirements defects that have not been reported in previous studies.

Chapter 7

Formalisation of Requirements Defects Detection

After defining the normal form for Requirements Behaviour Trees (RBTs) in the previous chapter, this chapter further extends this work. It formalises the four most common requirements defects in the Behaviour Tree (BT) context.

Based on the formalisation of the requirements defects, we propose a Behaviour Engineering-based Requirement Defects Detection (BERDD) framework. In (BERDD), we translate user requirements into BTs and then develop an algorithm to convert BTs into a formal language called Web Ontology Language (OWL). To further support BERDD, we develop a prototyping tool that uses SPARQL query language to retrieve information related to requirements defects from the OWL knowledge base. We believe that this approach exceeds all other requirements defects detection tools regarding coverage.

7.1 Introduction

Software projects often begin with unclear, ambiguous, and incomplete requirements from the initial requirements elicitation that may introduce many requirements defects, which could increase projects risks and even cause project failure [5, 20]. Software requirements defects are not only introduced during the initial requirement elicitation process but may also be introduced during requirements changes. Requirements defects detection is always a challenging task in the Software Development Life Cycle (SDLC). To detect requirements

defects during requirements engineering phases is less expensive and easy to fix than in later phases of SDLC [40, 41, 269].

The RCM process may also introduce many requirements defects such as incompleteness, inconsistency, etc., which are also common and faced during the requirements elicitation and analysis phase. According to Anwer et al. [201], requirements inconsistency is one of the major challenges faced during the RCM process in both in-house and global software development.

In the past, several studies have been carried out to address defects detection during the requirements elicitation and analysis phase [170, 270] and during the RCM process [72, 233]. However, traditional approaches that use review and inspection to detect requirements defects are time-consuming, error-prone, and make the system maintenance process cumbersome. Moreover, few studies have been conducted to detect requirement defects through logic reasoning and supported by tools. Goknil et al. [271] proposed a first-order logic-based approach to perform change impact analysis and detect inconsistency defects faced in requirement analysis and in the RCM process. Similarly, Reder and Egyed [272] developed an automated tool based on the design rules. The developed tool helps to detect inconsistency defects arising during the RCM process by validating the design rules. The existing approaches produce good results; however, some limitations need attention:

Limitation 1: Some existing approaches translate natural language requirements directly to some formal languages such as first-order logic or propositional logic by using natural language processing techniques then apply reasoning techniques to detect requirements defects [82, 273]. However, the customers who provide the requirements may find it difficult to understand formal languages, while the engineers may not have the domain knowledge to interpret the customers' requirements correctly. Therefore, it is difficult to verify if the formal representation is correct.

Limitation 2: Most of the existing approaches based on formal logic do not provide appropriate tool support. The reason is that they define requirements defects through natural language descriptions, which might have ambiguous interpretations. Due to this, it seems difficult to formalise defects, and as a result, it becomes challenging to develop an automated tool.

Limitation 3: Most of the existing approaches cover only inconsistency defects but miss some other common defects such as incompleteness, redundancy, and ambiguity.

Given this need to formalise requirements defects (limitation 1) and limited coverage of the existing approaches (limitation 3), we use BT as a modelling notation and introduce Inertia Axiom as a foundation to define normal formed RBT. The normal formed RBT is used as a foundation to define requirements defects. Based on this concept, we have successfully defined four major requirements defects such as incompleteness, ambiguity, redundancy, and inconsistency in the BT context and develop algorithms to detect them. Simple examples are given to explain the algorithms to detect these four requirements defects. This work provides significant coverage of requirements defects, and according to Hayes [274], these four requirements defects types covers almost 83% of the total requirements defects. Similarly, according to another study [275], our new work covers 73% of the total defects in the requirements engineering phase and according to our literature research whose details are given in Section 3, none other researches have such high coverage.

Based on BT notation, we also propose a Behaviour Engineering-based Requirement Defects Detection (BERDD) framework. In BERDD, we translate software requirements into Behaviour Trees BTs, and then we develop an algorithm to translate BTs into a formal logic language called OWL. One of the advantages of this approach is that behaviour trees, which is a semi-formal modelling notation introduced in behaviour engineering, have proved to be an ideal bridge to connect software requirements in natural languages to their formal representations [104, 257, 263]. Therefore, BTs help the customers to verify the accuracy of the formal representation.

To support BERDD and address limitation 2, we have developed a prototyping tool named Requirements Defects Identifier (RDI). Although there are many commercial requirements management tools available such as DOORS [276], and Requisite Pro IBM [277, 278] that provide good coverage of requirements management, they are limited in analysis and validation support for requirements defects detection [279]. RDI provides an end-to-end support for requirements analysts to identify requirements defects. Good tool support also reduces dependency on expert-level understanding of formal languages [45]. Our tool uses SPARQL, a semantic query language, to query the OWL knowledge base and retrieve data about requirements defects, but it does not require the user to know SPARQL.

To prove the feasibility of our approach and demonstrate the capability of RDI, a real-world system has been explored in this chapter. The results show that defects of all four different types have been successfully identified.

The rest of the chapter is organised as follows. Section 7.2 presents relevant existing works, and section 7.3 discusses the proposed framework BERDD. Section 7.4 is the most important section in this chapter; it uses the Inertia Axiom as a foundation to define requirements defects formally. It also introduces algorithms to detect those defects. The developed tool is presented in Section 7.5, following this, an example to demonstrate the applicability of the proposed approach is given in section 7.6. Implications and limitations of this chapter are discussed in section 7.7. Finally, the conclusion is discussed in section 7.8.

7.2 Related Work

Existing research related to requirements defects detection can be classified into two main categories: the first uses formal languages that allow automated analysis of Requirements Engineering (RE) defects. The second uses Natural Language Processing (NLP) techniques to detect RE defects. We also explore existing work that uses SPARQL to perform different types of analysis on software requirements.

7.2.1 Requirements Defects Detection using Formal Languages

In this category, some studies have translated user requirements from natural languages directly to formal languages, but some approaches first translated user requirements to some semi-formal languages such as UML and then converted them into formal languages and performed defects detection.

7.2.2 Requirements Translation directly to Formal Languages

Many different logics, including first-order logic, propositional logic, and description logic, have been used as formal languages to represent software requirements. These logics also provide the capability to perform analysis and reasoning.

For example, Zowghi et al. [43] proposed a technique to detect inconsistencies in requirements formally. They developed a prototype tool called CARL to evaluate the proposed technique. They translated the user requirements to propositional logic and then applied reasoning on it to check inconsistencies. This approach's limitation is that propositional logic is not powerful enough to model complex system behaviour [116].

Similarly, Nguyen et al. [170] developed an automated knowledge-based engineering tool called REInDetector to detect requirements inconsistencies and redundancies. Their approach translates user requirements directly to DL based ontologies and then performs requirements defects analysis on them. However, this approach cannot detect conflicts associated with requirements that cannot be translated directly into DL, such as temporal operators [280].

In another study, Chanda et al. [38] presented an approach to check consistency between different UML models (class, use case, activity diagram) using regular grammars. They developed context-free grammars for each diagram using regular expressions to check syntactic correctness and consistency between different models. This approach performs well but only checks problems from a syntactic perspective without considering semantics perspectives.

7.2.3 Requirements Translation using Semi-Formal Languages as a Bridge

In the past, many studies have been carried out to check requirements defects by using semi-formal languages. These approaches have been conducted in two steps; first, they translate natural language described requirements into semi-formal languages and then into formal specifications.

For example, Kamalrudin et al. [281] developed a tool named MaramaAIC to check inconsistencies in natural language requirements. They extracted abstract interactions, also called essential use case patterns, from natural language requirements using an interaction library and then highlighted the inconsistencies and incompleteness problems. However, this approach depends on the developed interaction library, limiting the applicability of this study.

In another study, Liu et al. [282] developed a technique to check requirements defects in use case diagrams. They translated use case descriptions to an activity diagram through

dependency parsing techniques and then defined formal rules to check defects in the activity diagram. They designed eight rules to parse use case descriptions to activity diagrams covering a limited set of constructs from use case descriptions.

Kroha et al. [283] implemented an approach to check the consistency of requirements specifications by using OWL. They built UML models from user requirements for better understanding and then transformed UML models to textual form by using off-the-shelf tools; after that, they built an ontology from textual requirements. Finally, they used automatic reasoners on OWL to perform consistency checks.

7.2.4 Requirements Defects Analysis using Natural Language Processing

In the past, a few studies have been conducted by using NLP techniques to detect requirements defects. These studies used many techniques, for example, Mavin et al. [284] used constrained natural language, Arora et al. [285] used pre-defined templates, and Tjong and Berry [286] used a rule-based approach to detect RE defects.

Hasso et al. [287] proposed a rule-based approach to detect requirements defects expressed in the German language. Similarly, Femmer et al. [288] proposed a rule-based RE defects detection approach and achieved 59% precision.

7.2.5 Application of SPARQL for Requirements Analysis

In the past, some work has also been undertaken to analyse software requirements by using SPARQL. Wei et al. [289] proposed an approach to perform analysis of different UML diagrams using SPARQL queries. They converted class, sequence, and state machine diagrams into OWL, and they used SPARQL queries to perform various types of analysis. But their work does not cover requirements defect detection. Similarly, Sadowska and Huzar [290] used SPARQL to perform different types of analysis on class diagrams.

In another study, Siegemund et al. [291] proposed an approach to detect inconsistency and incompleteness defects in software requirements using SPARQL queries. Based on the goals-oriented software engineering approach, they described natural language requirements as goals and developed rules for checking inconsistency and incompleteness based

on the defined goals. Furthermore, Verma and Kass [292] proposed a similar approach to detect requirements defects. They defined a controlled syntax to write software requirements and then developed rules for checking inconsistency and incompleteness based on the defined syntax. SPARQL query is used in their approach.

7.2.6 Limitations of Current Research related to Requirements Defects Detection

The existing research related to requirements defects detection has produced some good results; however, there are a few limitations as follows:

- The above-mentioned formal approaches that translate natural language requirement directly into formal languages are useful to detect requirements defects; however, the success of using formal languages requires an expert-level understanding of these languages [45]. In the absence of such knowledge, the customers who provide the requirements may find it difficult to understand formal languages, while the engineers may not have the domain knowledge to interpret the customers' requirements correctly. Therefore, it is difficult to verify if a formal representation is correct.
- The above-mentioned approaches that use UML to bridge requirements from natural languages to formal languages address the problem of semantic ambiguity in natural languages; however, inconsistency detection is difficult as there are so many different types of UML diagrams [293]. UML offers 13 different types of diagrams to represent the structure and behaviour of software systems [294, 295]. In contrast, BT was developed to use a minimum set of coherent modelling notations throughout the modelling process [91, 96, 97]. As a result, it is easier to detect inconsistency in a minimum set of diagrams.
- The above-listed approaches that use NLP produce good results; however, there is some inherent limitation with NLP based approaches. For example, Zhao et al. [296] conducted a mapping study to investigate NLP based approaches for requirements engineering activities. They reported that current research had been used on a small scale, and that only 7.18% of the studies used some industrial case studies for evaluation. Another limitation of this research is that they mostly focused on

syntactic analysis rather than semantic analysis [296]. Similarly, Dalpiaz et al. [297] discussed the future of NLP in the context of requirements engineering and concluded that current NLP research still lacks in some aspects, including in the unavailability of correct performance metrics and in the lack of understanding of context-dependency involved in natural languages requirements.

- SPARQL has been used frequently to perform different types of analysis for software requirements. However, to the best of our knowledge, only one study used it to detect requirements defects, and it detects only two defects types. Moreover, the existing approaches lack proper tool support, and they need the requirement analysts to understand SPARQL to do the work, which limits their applicability.

7.3 Requirements Defects Detection Framework

This section introduces the novel framework, BERDD (Behaviour Engineering based Requirements Defects Detection), which detects the requirements defects that arise during requirements elicitation and analysis and RCM. First, we discuss the framework's overall structure, and after that, we discuss its workflow.

7.3.1 BERDD Structure

The overall structure of BERDD (shown in Figure 7.1) contains four domains; they are problem domain, techniques domain, tools domain, and solution domain.

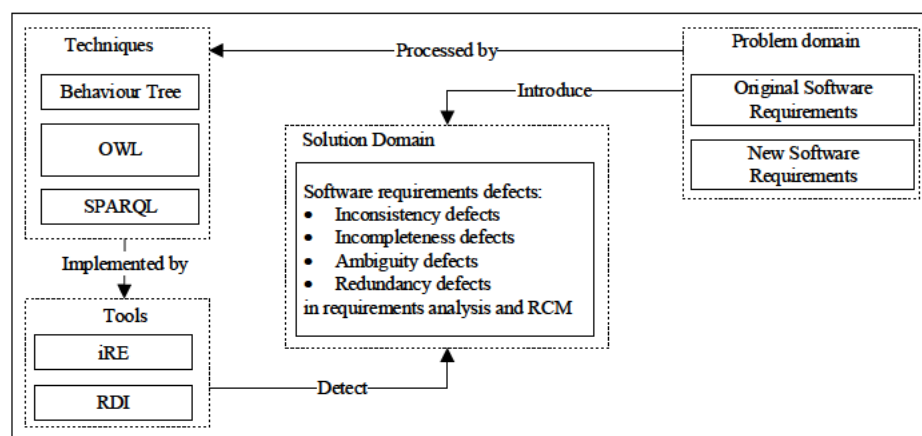


FIGURE 7.1: Key elements of BERDD

The problem domain is composed of two elements, the original software requirements and the new software requirements. Even though the proposed framework is designed to detect requirements defects during the requirement analysis phase, it can also be applied to detect requirements defects raised during the RCM process. The modified requirements in the RCM process are also considered as new software requirements. The problem domain introduces the problems that need to be detected in the solution domain. The solution domain comprises the four most common software requirements defects: requirements incompleteness, inconsistency, redundancy, and ambiguity.

The techniques domain is composed of three techniques. Firstly, we use behaviour trees to model software requirements. Secondly, OWL is used to represent the requirements formally. Lastly, SPARQL query language is used to query the knowledge base and detect defects from software requirements represented as OWL.

In the tool's domain, two tools are used to implement the previously discussed techniques. Firstly, the iRE, which has already been published [104], is used to design and modify BTs and to map BTs into XML format. Secondly, the newly developed Requirements Defects Identifier (RDI) is used to translate a BT from XML format to its OWL representation. RDI can also execute SPARQL queries through a SPARQL engine on the knowledge base and identify requirements defects. The combination of the two tools will provide end-to-end support for the requirements analyst to identify requirements defects introduced during the requirements analysis and RCM.

The solution domain contains the list of identified requirements defects. This paper gives the algorithms to detect four of the most common categories of defects, but other types of defects could also be detected if corresponding algorithms were given.

7.3.2 BERDD Workflow

The workflow of BERDD is given in Figure 7.2, and it is composed of three steps.

- In step 1, a new RBT is designed by using drawing tools such as iRE [104]. In case of requirements change, an existing RBT is modified based on the change type. An important consideration related to BERDD is that, before applying this framework during RCM, the existing system should have all RBTs and the IBT

ready based on the previous requirements. The integration of new RBTs with an existing IBT is also performed in this step. An important consideration is that some requirements defects such as requirements incompleteness are identified during the translation of natural language requirements to RBT or RBT integration to form an IBT (discussed in detail in the next section). However, other defects such as inconsistency, redundancy, and ambiguity are exposed after integrating new RBTs with an existing IBT. Lastly, the final IBT is mapped to an XML file using the same tool (iRE).

- In step 2, the requirement defects are formalised in the BT context. The defects formalisation step is constant and independent of the given IBT and has no prerequisite, and can be performed in parallel to step 1.
- In step 3, the prototyping tool named RDI (introduced in section 6) will be used to performing functions, including translating an IBT into an OWL, using a query language to access the OWL knowledge base and detect formalised requirements defects. In this research, we use SPARQL queries to query the OWL knowledge base; however, other query languages such as description logic query can also be used to query the knowledge base.

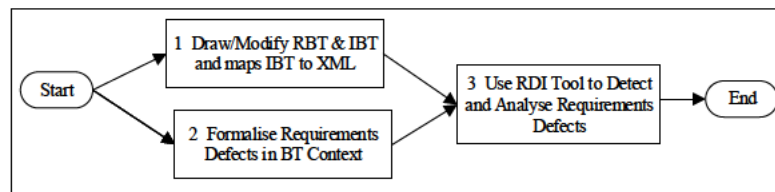


FIGURE 7.2: BERDD Workflow

The first step is identical to the normal BE approaches that were introduced in subsection 2.5.1 and published in previous papers [91, 104]. Step 2 will be introduced in the next section, and step 3, which is related to our developed tool, will be discussed in section 7.5.

7.4 Requirements Defects

In this section, first, we discuss requirements defects classification. After that, we discuss step 2 of the BERDD framework, which is to give a formal definition of the four most

common requirements defects in the BT context, and the algorithms to detect them. Before providing the definitions, we recall Inertia Axiom and the normal form of RBT, which was introduced in the previous chapter, Chapter 6. After that, we introduce the four most common requirements defects and formalise them. It is usually much easier to detect requirements defects in semi-formal or formal notation than natural language requirements description; that is why we use modelling notation to formalise requirements defects. In this subsection, we discuss these defects theoretically with simple examples; in section 7.6, we will use a more complex example and developed tool to demonstrate the defects detection process. The process, how SPARQL queries are used to detect these defects, is also discussed in that section.

7.4.1 Requirements Defects Classification

In this subsection, we discuss the classification of requirements defects and their corresponding definitions presented in the literature. Several requirements defects taxonomies have been proposed in the existing research with many different definitions of each type of defects. We choose the four most common requirement defects such as incompleteness, ambiguity, redundancy, and inconsistency. The four most common requirements defects and their corresponding definitions from different studies are listed in Table 7.1.

Even with the same name, the different definitions may give different coverage for each type of requirement defect. This work will only cover some aspects of these requirements defects. Regarding incompleteness, three listed definitions cover two different aspects of incompleteness: the incompleteness of an individual requirement (1a, 1b) and incompleteness of overall system requirements (1c). In this research, we will only address the incompleteness defect of an individual requirement.

Regarding ambiguity, the first two definitions are similar and will be addressed during the BT process's translation phase. This research will address the third definition, which states that two different terms refer to the same thing.

Regarding inconsistency, the first definition (3a) is more related to ambiguity, and we cover it in the ambiguity defect part. This research covers the second and third definitions, indicating that two or more stakeholders might have conflicting requirements.

TABLE 7.1: Requirements defects classification

No.	Defect Type	Definitions
1	Incompleteness	1a. Information relevant to the requirement is missing [298]. 1b. A requirement must have all relevant components [169]. 1c. Required behaviour and output for all possible states under all possible constraints [274, 299].
2	Ambiguity	2a. The requirement contains information or vocabulary that can have more than one interpretation [298]. 2b. The information in the requirement is subjective [274, 298]. 2c. Two different terms are used to refer to the same thing [300].
3	Inconsistency	3a. Two or more requirements in a specification contradict each other due to inconsistent use of words and terms [168, 274]. 3b. Two or more stakeholders have different, conflicting requirements [301]. 3c. The requirement or the information contained in the requirement is inconsistent with the overall document [298].
4	Redundancy	4b. Two different executions are redundant if they produce indistinguishable functional results from an external viewpoint [298, 302].

Regarding the coverage of this approach, as we have discussed above, these four requirements defects types cover almost 83% [274] and 73% [275] of the total requirements defects. Regarding each category, it might be difficult to quantify the extent to which our approach covers each category because no such study has been conducted in the past. However, we believe that we cover a significant part of each defect types that can be detected automatically with minimal human intervention and domain knowledge. For example, domain knowledge is required to check the incompleteness of overall system requirements; however, individual requirements' incompleteness can be checked automatically without domain knowledge, and we have checked it in our approach. Similarly, based on the consistency and redundancy definition we have discussed above, we believe that our approach covers the significant part of these two defects types.

7.4.2 Normal Form of Requirements Behaviour Tree

In this section, we simply recall the Inertia Axiom and normal form of RBT. The original BT approach introduced two axioms, Precondition Axiom and Integration Axiom [91].

Here we introduce a third axiom called Inertia Axiom.

Definition (Inertia Axiom): *A system will be in a state unless an external or internal event happens to trigger it into a new state, while the new state could be the same as the original state.*

Our proposed axiom is well aligned with software engineering principles. According to this principle, each software requirement should usually be analysed in the context of three key elements: 1, precondition, 2, event, and 3, postcondition. The precondition enforces the conditions that need to be met so that the expected event can be accepted; the event's consequence is called the postcondition. Based on the above-defined axiom and intuition of an individual software requirement, now we define a normal form of RBT.

Normal Formed RBT: *A RBT is called normal formed if and only if it consists of three sequential parts, precondition, event, and postcondition.*

The precondition and postcondition of an RBT must consist of at least one state realisation node. An RBT event is usually a node with the behaviour type as an event, but it could be a node with selection or guard behaviour types.

To explain the normal formed RBT, we take one requirement from a simple example, a microwave oven, which has been used in existing published research [106]. We simplify this requirement to explain the normal formed RBT concept effectively.

R5: *Whenever the oven is cooking, then opening the door stops the cooking.*

Figure 7.3(a) shows the RBT for R5. The first node is the precondition. It shows “OVEN” component is in the state of “Cooking”. The square brackets around “Cooking” indicates the behaviour type as “state realisation”. After that, the second node is the event. In this node, the “DOOR” component is “Opened”, and the double question marks around “Opened” indicates its behaviour type as an event. Due to the event of “DOOR” ??Opened??. the system gets to the postcondition represented in the third node with the “OVEN” component in the state of “Cooking-Stopped”. The behaviour type is also a “state realisation”. As discussed above, the precondition and postcondition of an RBT must consist of at least one state realisation node. The RBT for requirement (R1) with more than one node in precondition and postcondition is shown in Figure 7.3(b).

R1: *Originally, the oven is in Idle state and the Door is closed and when the button is pushed, the Power-tube will be energised and the oven will start cooking.*

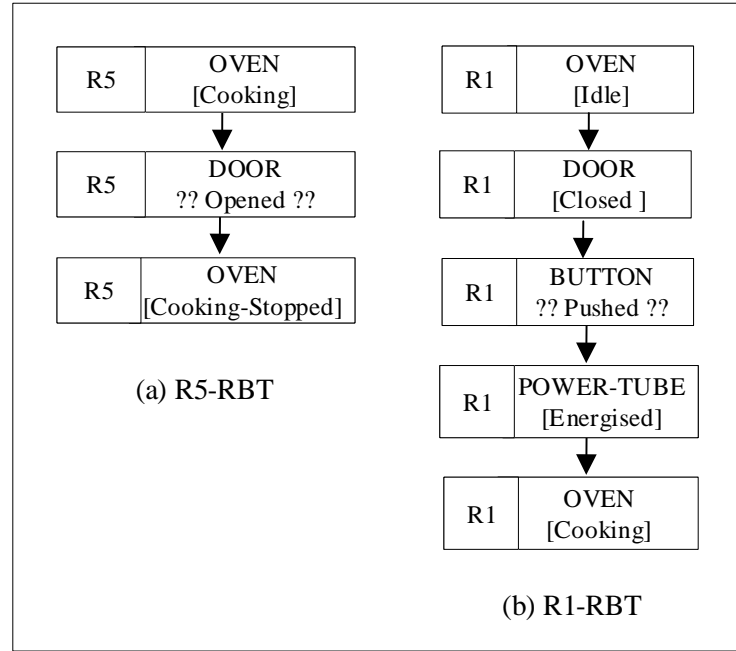


FIGURE 7.3: Normal formed RBT

Based on the normal formed RBT, we can define a normal formed IBT. An IBT is formed by integrating a set of individual RBTs, as discussed in chapter 2 (2.5.1). If an IBT is formed by integrating a finite set of normal formed RBTs, it is called a normal formed IBT. The concept of normal formed IBT sets up a solid foundation to improve requirements modelling quality and helps to formalise requirements defects. The important point is that our normal formed RBT is a subset of the original RBT; therefore, it is aligned with BT taxonomy [96] and preserves the BT modelling notations.

A normal formed RBT helps to define the structure of an individual requirement, and in this research, we formalise the four most common requirements defects based on this defined structure. This research only defines the four most common requirements defects. However, other types of defects, if they could be defined based on the normal formed IBT, could also be formalised and might be automatically detected.

7.4.3 Incompleteness Defect and its Detection

7.4.3.1 Definition

Requirements incompleteness can be discussed from two perspectives: the incompleteness of overall system requirements and secondly, the incompleteness of individual requirements and both types are discussed in detail in subsection 7.4.1. In this research, we are only focusing on the incompleteness of individual requirements. According to normal formed RBTs, each requirement must contain a precondition as the context, an event that triggers the transition of state and a postcondition to consider a complete requirement. Based on this definition, now we define incompleteness defect in the context of BT:

Definition: *A requirement is incomplete if the precondition or event or postcondition is missing.*

Although missing any of these parts makes the requirements incomplete, the incompleteness defect due to missing precondition and postcondition is more common than missing event. Based on the definition of the incompleteness defect, for each requirement, the analyst should ask the questions: what is the precondition? What is the event, and what is the postcondition? If for an individual requirement, any of the above questions cannot be answered, and then the requirement has an incompleteness defect.

In the context of the BT approach, the incompleteness problem as a missing precondition can be discussed from two perspectives. Firstly, the behaviour type of the first node of the new RBT is not state realisation, and secondly, the root node of any RBT, including the new tree is not matched with any node of an IBT. If the new RBT's root node has a state realisation behaviour type but cannot be matched with any of the IBT nodes, the requirements analyst may consider the new RBT as a root requirement of an IBT. Otherwise, it would be considered an incomplete requirement. The missing event means individual requirement does not have an event node after precondition, and missing postcondition means no node with state realisation behaviour type after an event node.

7.4.3.2 Example

We will demonstrate requirements incompleteness through a simple system microwave oven [106]. As BERDD can be applied in both requirements analysis and RCM, so, in

TABLE 7.2: The requirements of microwave oven-Incompleteness

#	Requirement
R1	Originally, the oven is in Idle state and the Door is closed and when the button is pushed, the Power-tube will be energised the oven will start cooking.
R2	If the button is pushed while the oven is cooking, it will cause the oven to cook for an extra minute.

this example, we consider BERDD in the context of requirements change. For example, initially, we have two functional requirements shown in Table 7.2, and the corresponding IBT is shown in Figure 7.4. Here we use requirements number in the sequence, and it might differ from numbering in the original paper. During the requirements change process, the following new requirement comes from one of the system stakeholders, and the corresponding RBT is shown in Figure 7.5.

R3: *Opening the door stops the cooking.*

First, we will design an RBT of the new requirement (R3), and after that, we evaluate the new RBT based on normal formed RBT definition to check for an incompleteness defect. Checking the new RBT (R3) shows that the root node does not have a state realisation behaviour type, and the conclusion is that the precondition of the new requirement (R3) is missing. Now it is the requirements analyst's responsibility to analyse the new requirement based on domain knowledge and either introduce the missing node as a precondition or contact the corresponding stakeholder for further discussion.

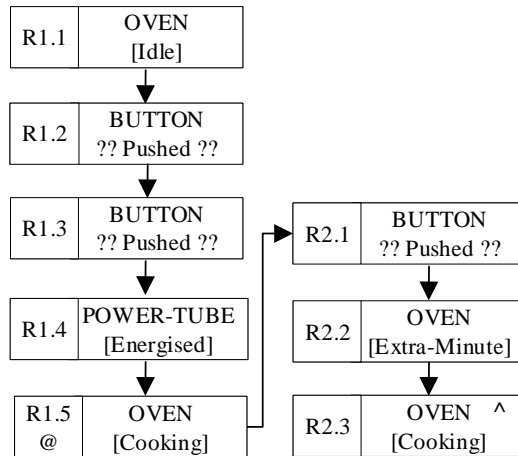


FIGURE 7.4: Oven IBT for incompleteness defects

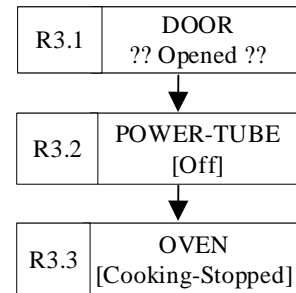


FIGURE 7.5: R3-RBT

7.4.4 Ambiguity Defect and its Detection

7.4.4.1 Definition

The software requirements written in natural languages can be ambiguous and ultimately cause a system failure. Requirements ambiguity can appear in many ways: the first is semantic ambiguity, in which a sentence has correct grammar but could have different interpretations. For example, a man saw a woman on a hill with a telescope. This sentence may have three different interpretations depending on what exactly “with a telescope” describes. This kind of ambiguity can be usually resolved during the process while natural language described requirements are translated into RBTs. The second type of ambiguity happens when two different terms are used to refer to the same thing [300]. For example, in a software system, people may use a vehicle and a car to refer to the same component. Contrary to the second type of ambiguity, the third type of ambiguity happens when the same term refers to two or more different objects. In this research, we only consider the last two types of ambiguities, as the first type of ambiguity can be identified and resolved during the requirements translation stage.

In BT notation, behaviour types are predefined in BT taxonomy and independent of the problem description. However, component names and behaviour names are defined based on the problem description, so the last two types of ambiguity can arise in both component names and behaviour names. The ambiguity is defined as follows:

Definition: *The requirements are ambiguous if two different terms refer to the same component name or behaviour name or the same term refers to two different component names or behaviour names.*

For example, a node in a new RBT with a component name of CONTROL-BUTTON and another node in an IBT has its component name as BUTTON; both component names refer to the same component. An important consideration is that it is impossible to decide about ambiguity separately with a component name or a behaviour name, so it is better to analyse the component name and behaviour name together in the same node.

7.4.4.2 Example

We will demonstrate requirements ambiguity with the simple system, Security Alarm System (SAS). In this example, we will use BERDD in the context of the requirements analysis phase. For example, we have four functional requirements shown in Table 7.3, and the corresponding IBT is shown in Figure 7.6.

TABLE 7.3: The SAS requirements

#	Requirement
R1	Whenever the SAS is deactivated, it will be activated by pressing the set button.
R2	If a trip signal occurs while the SAS is set, a high-pitched tone (alarm) is emitted.
R3	A three-digit code must be entered to turn off the alarm.
R4	Whenever the SAS is activated, correct entry of the code deactivates the SAS.

A potential ambiguity defect arises with the component name “3-DIGIT-CODE” and “CODE”, whether both refer to the same thing or different things. Although both terms are closely related and to some extent requirements analysts can understand that, both terms point to the same component. However, an in-depth analysis can be performed when we check the component name together with the behaviour name. We can analyse that one node with the same behaviour name ‘Entered’ and behaviour type ‘Event’ exists in IBT, which means that CODE and 3-DIGIT-CODE point to the same component but with two different terms. After identifying potential ambiguity, the requirements analyst will analyse the scenario and decide how to handle the potential ambiguity.

7.4.5 5.4 Redundancy Defect and its Detection

7.4.5.1 Definition

Software redundancy is also called functional redundancy, and is one of the important mechanisms used throughout the software development life cycle. Redundancy has been broadly used in many software engineering practices to improve fault tolerance, reliability, and self-healing, in a self-adaptive system [303–306]. Software Redundancy itself is not an error; however, ineffective identification and utilisation of redundant elements in software systems limits their applicability [307].

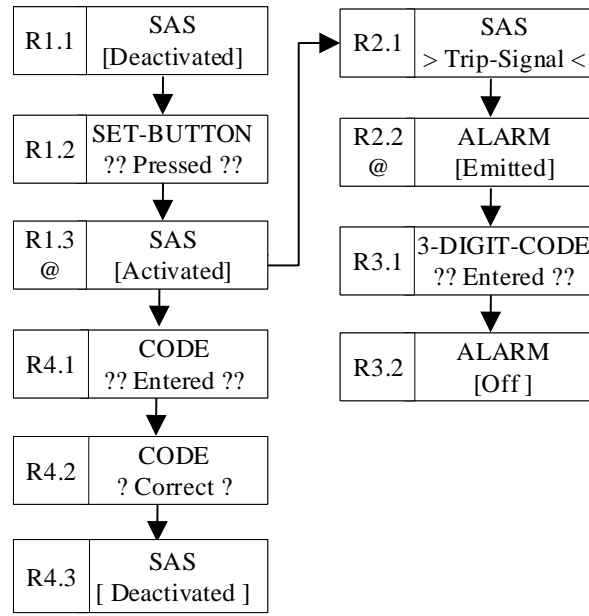


FIGURE 7.6: SAS IBT for ambiguity defects

In this research, we only focus on requirements redundancy defect, as we have discussed in subsection 7.4.1, which is different from software redundancy. A requirement redundancy defect means that the same requirement has been stated more than once; therefore, it could be mistakenly considered as two different requirements. Requirements redundancy usually appears during the requirements elicitation phase and should be detected during the requirements analysis phase. In this research, based on the normal formed RBT structure, the formal definition of redundancy is:

Definition: *Two requirements are redundant if they have the same precondition, event, and postcondition.*

Although most of the time, individual requirements express one event, individual requirements might express more than one event on occasion. Hence, one requirement may have the same precondition, event, and postcondition, which are part of another requirement.

7.4.5.2 Example

We will demonstrate the redundancy defects by using the simple system, microwave oven. We have three functional requirements shown in Table 7.4, and the corresponding IBT

is shown in Figure 7.7. Here we use requirements number in the sequence, and it might differ from numbering in the original paper.

A thorough analysis of the IBT reveals one potential redundancy defect. The nodes R1.2, R1.3, R1.4, and R1.5 are like nodes R3.5, R3.6, R3.7, and R3.8. The important point is that both trees have all the required elements of a normal formed RBT such as precondition, event, and postcondition.

TABLE 7.4: Oven requirements for redundancy defects

#	Requirement
R1	Originally, the oven is in Idle state and the Door is closed and when the button is pushed, the Power-tube will be energised the oven will start cooking.
R2	If the button is pushed while the oven is cooking, it will cause the oven to cook for an extra minute.
R3	While the oven is cooking, if the user opens the door, the cooking will be paused, and it resumes if the door is closed and the user pushes the button.

7.4.6 Inconsistency Defect and its Detection

7.4.6.1 Definition

Detecting requirements inconsistencies is a key component of any requirements engineering approach and has been topic of interest for several decades. Inconsistency in software requirements can be defined in many ways as we have discussed in subsection 7.4.1. In this research, we will cover the inconsistency type (3c), which states that a potential inconsistency arises when two or more stakeholders have different, conflicting requirements [301]. The formal definition of inconsistency in the context of normalised BT would be:

Definition: *Two requirements are inconsistent if they have the same precondition, event, but different postcondition or same event and postcondition but a different precondition.*

7.4.6.2 Example

We will demonstrate requirements inconsistency detection by using a simple system microwave oven. We have three functional requirements shown in Table 7.5, and the corresponding IBT is shown in Figure 7.8. Here we use requirements number in the sequence,

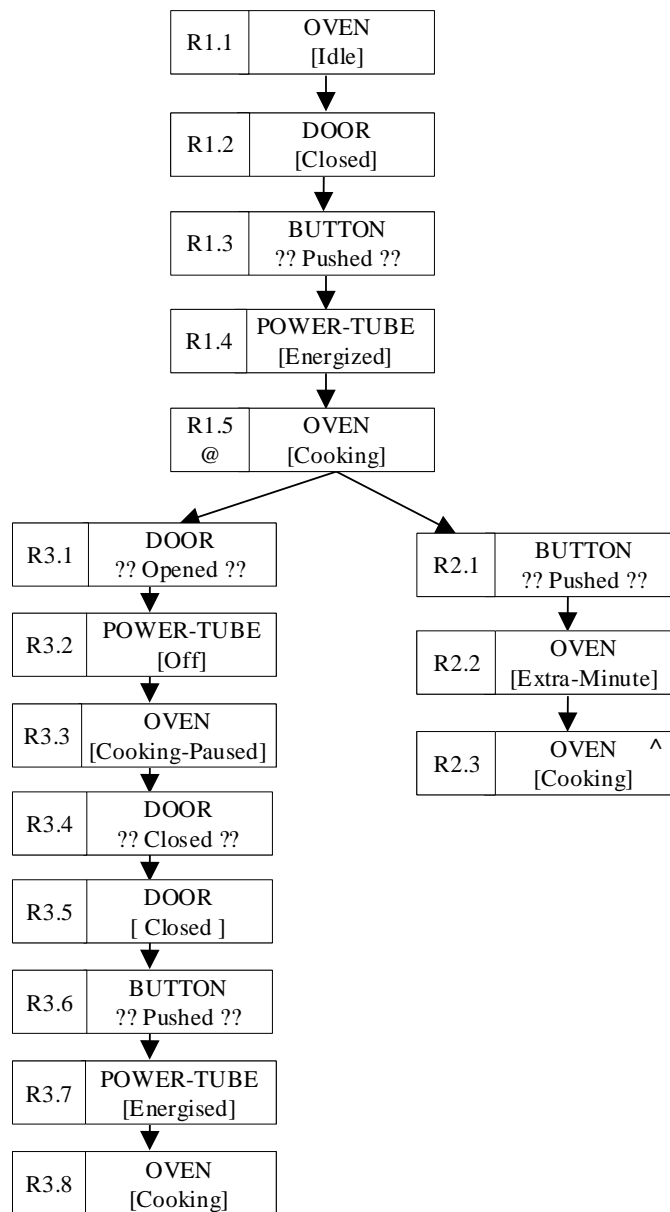


FIGURE 7.7: Oven IBT for redundancy defects

and it might differ from numbering in the original paper.

A thorough analysis of the IBT reveals one potential inconsistency defect. The nodes R1.5, R2.1, R2.2, are like nodes R1.5 R3.1, R3.2; however, the following nodes (R2.3) and (R3.3) of both sequences are different. It means that the same precondition and event produces a different postcondition, which shows a potential inconsistency defect. After successfully detecting potential inconsistency defects, the requirements analyst will decide how to resolve them.

TABLE 7.5: Oven requirements for inconsistency defects

#	Requirement
R1	Originally, the oven is in Idle state and the Door is closed and when the button is pushed, the Power-tube will be energised the oven will start cooking.
R2	While the oven is cooking, if the user opens the door, the oven pauses the cooking and resumes if the door is closed and the user pushes the button.
R3	Whenever the oven is cooking, opening the door will stop the cooking.

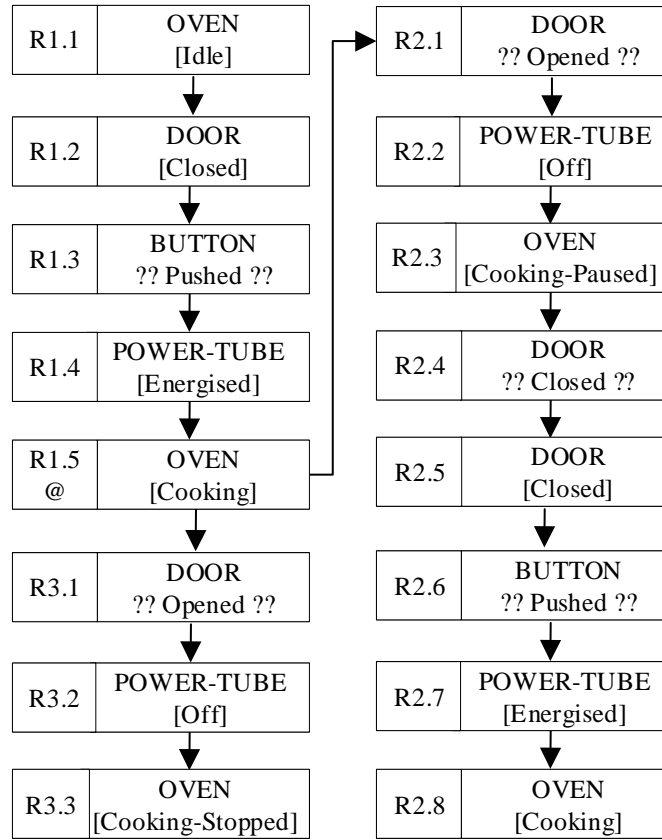


FIGURE 7.8: Oven IBT for inconsistency defects

7.5 Tool (RDI) Environment

This section introduces the newly developed tool (RDI) environment, which helps to execute step 3 of the BERDD workflow (See Figure 7.2. Firstly, we discuss the workflow of RDI, and then we discuss step 3 (translation from IBT-XML to OWL) RDI workflow.

7.5.1 RDI Workflow

This subsection introduces the key elements of RDI (see Figure 7.9) and its workflow. The RDI workflow is composed of five steps.

- In step 1, either original software requirements are gathered, or a requirement change request is captured from different stakeholders. In the case of requirements change, further analysis to understand the type of change request is also performed in this step. The requirement change request can be categorised into two different groups, either addition of new requirements or modification of existing requirements.
- In step 2 (a, b), either a new RBT is designed, or an existing RBT is modified using some BT drawing tools such as iRE [104]. The integration of the new/modified RBT with the IBT and saving it to an XML file is also performed in this step. The RDI tool collaborates with the BE tool iRE to provide end-to-end support for requirements analysts to identify requirements defects during requirements analysis or the requirements change phase.
- Step 3 translates the XML file into OWL based on the translation rules, which will be discussed in the next subsection.
- In step 4, SPARQL queries are constructed based on defect types and other related input parameters. An important consideration is that the users don't have to master SPARQL; they only need to choose a defect type and enter requirements defect-related parameters by using the RDI interface. Then a SPARQL query will be built automatically based on the input. The SPARQL queries will be executed by a SPARQL query engine (dotNetRDF), and the query results will be displayed to the users.
- In step 5, the requirements analyst will analyse the query results and decide how to handle the detected defects based on domain knowledge.

Step 2 will be executed by using the iRE tool, and the output of this step will be used as an input of step 3, which will be discussed in detail in the following subsection. Steps 4 and 5 will be discussed in the next section with the help of an example. We built RDI by using visual studio 2019 Windows form application in C#. We included the

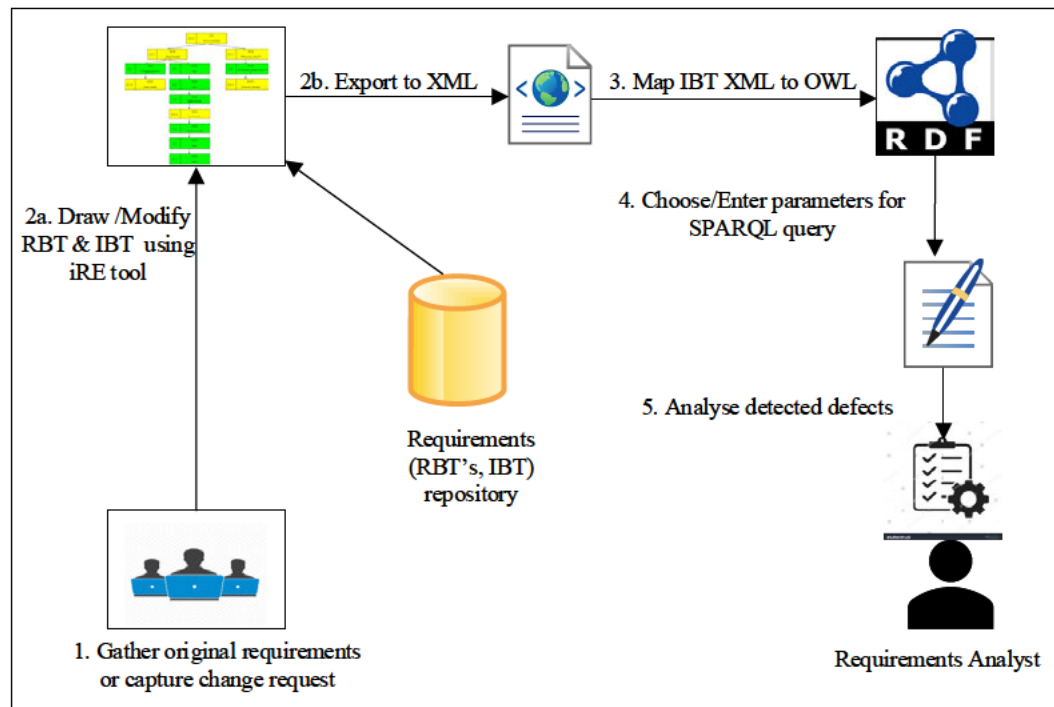


FIGURE 7.9: RDI Workflow

dotNetRDF library [308] (as a .dll file) in the visual studio environment; this library is used to execute SPARQL queries. This subsection only briefly introduces the key elements and the workflow of RDI, as more details of the tool and some screenshots are given in the next section.

7.5.2 Behaviour Tree Representation in OWL

This subsection provides a detailed description of step 3 of the RDI workflow, which is to translate an IBT from its XML form into its OWL representation. Before explaining an algorithm to convert IBT to OWL through an example, we provide a graphical representation of the process.

Figure 7.10 illustrates the process to convert an XML representation of an IBT into its OWL representation. This process will take the XML representation of IBT as an input and generate OWL representation of an IBT as an output. In step 1, we define some predefined fields of the OWL file related to IBT representation, including OWL classes and object properties. In step 2, we define IBT nodes as OWL individuals. In step 3,

we associate IBT node properties such as behaviour name, behaviour type, component name through OWL object properties defined in the first step. Following this, in step 4, we establish links between IBT nodes using OWL object property "nextNode". In the following part of this section, this process will be explained by using an example.

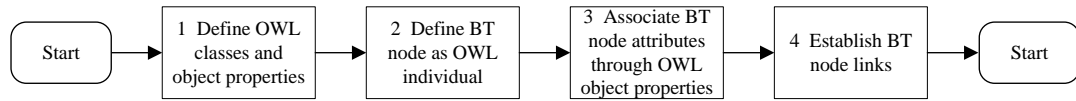


FIGURE 7.10: Steps to convert IBT-XML to OWL

To define some predefined fields of an OWL file related to IBT representation, we extract BT taxonomy based on a previously published metamodel [98], and through this taxonomy, four (4) meta-classes are identified as below:

- Component Name
- Behaviour Name
- Behaviour Type
- BT Nodes

An IBT is composed of many nodes, and each node has a few attributes such as behaviour name, component name, behaviour type, and traceability link, which must belong to one of the above listed four meta-classes. In OWL representation, class instances are described as individuals. An object property is used to establish a relationship between two individuals. Based on the BT taxonomy, we define four (4) different types of object properties: `hasComponentName`, `hasBehaviourName`, `hasBehaviourType`, and `nextNode` to establish a relation between different OWL individuals. The translation proposed in this paper follows similar guidelines as performed for UML to OWL translation [270, 290], and we developed translation rules based on these guidelines.

1. Input: XML file of IBT
2. Output: OWL file of IBT

```
//Declare four meta-classes
3. For each of the meta classes, insert "Declaration (Class (:meta-class_name))"
//e.g., Declaration (Class (:ComponentName))

//Declare object properties
4. For each of the object property, insert "Declaration (ObjectProperty (:objectprop-
erty_name))"
// e.g., Declaration (ObjectProperty (:hasComponentName))

//initialise empty node attributes list
5. L_attributes:= ∅

//read xml file
6. While (!=EOF)
7.     {
           //read node properties tag from XML file
8.         Node = XML_Node Tag

//check BT node attributes such as behaviour name, component name, and behaviour
type in corresponding lists, if not exist then declare it, associate it with related meta- class.

9. For each of the node attribute, If ((Node (attribute) Not IN L_attributes)
10.     {
11.         Declaration (Individual (:Node (attribute)))
12.         Declaration (:Node (attribute) :meta-class_name)
13.         Add (L_attributes, Node (attribute))
14.     }

//declare node ID as an individual and associate it with BTNodes class
15. Declaration (Individual (:Node (NodeID)))
16. Declaration (:Node (NodeID) :BTNodes))

//associate node attributes with the node ID
17. For each of the node attribute, insert "ObjectProperty (:Node (NodeID) :Node (at-
tribute))"
e.g., hasComponentName (R1.1, SAS)
```



```
//use nextNode object property to establish relationship between node
18. nextNode (Node(:SourceID) :Node(DestinationID))
19. }
```

Program 1: The pseudocode to translate an IBT into OWL

The process of translating an IBT into OWL is described in pseudocode in program 1. This program will take an IBT expressed in XML format as an input and produce an OWL representation of the corresponding IBT, mentioned in lines 1 and 2, respectively. To translate an IBT expressed in XML format to its OWL representation, we first declare the default elements of OWL syntax such as meta-classes, and individuals. These elements are the same for any IBTs. In OWL syntax, the Declaration keyword is used to declare or write classes, individuals as instances of OWL classes, and object properties. The OWL meta-classes and object properties are declared on line 3 and line 4, respectively, and the corresponding OWL representation is shown in Figure 7.11. The empty list for node attributes is initialised on line 5.

```
<!--Classes-->
<Class rdf:about="&BERDD-Ontology;BTNodes"/>
<Class rdf:about="&BERDD-Ontology;BehaviorName"/>
<Class rdf:about="&BERDD-Ontology;BehaviourType"/>
<Class rdf:about="&BERDD-Ontology;ComponentName"/>
<!-- Object Properties -->
<ObjectProperty rdf:about="&BERDD-Ontology;hasBehaviorName"/>
<ObjectProperty rdf:about="&BERDD-Ontology;hasBehaviorType"/>
<ObjectProperty rdf:about="&BERDD-Ontology;hasComponentName"/>
<ObjectProperty rdf:about="&BERDD-Ontology;nextNode"/>
<!-- Individuals Behavior Names -->
```

FIGURE 7.11: OWL meta-classes and object properties declaration

After that, we start reading IBT node attributes from the XML file and will continue until we have read the entire file. We will explain this process with a simple example, an SAS (Security Alarm System), and we consider only one requirement to explain this algorithm.

R1: *SAS is in the Deactivated state, once a SET-BUTTON is Pressed, the SAS is activated.*

The IBT (it is the same as the RBT as we listed only one requirement) of the SAS is shown in Figure 7.12, and the corresponding XML file is shown in Figure 7.13. The complete

OWL file obtained after translating the given XML is shown in Appendix B.1, but here we show some parts such as Figure 7.11 of the OWL file to explain the translation algorithm.

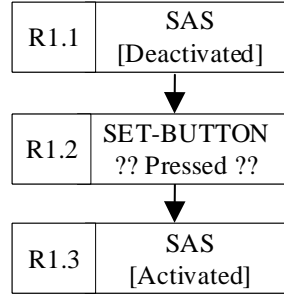


FIGURE 7.12: SAS IBT

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <graphbt:GraphBT xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
3    xmlns:graphbt="http://org.intellig.ire.core.graphbt/1.0.0/">
4    <nodes nodeId="1c3822c0-d2d4-49a6-819d-e723021ccfd0" componentName="SAS" behaviorName="Deactivated"
5      traceabilityLink="R1.1" behaviorType="State Realization" outgoing="0894451e-c291-4993-8ebc-49d845955b3e"/>
6    <nodes nodeId="c2ef4e2e-3ecf-41bd-af99-2b6c17f8cbb9" componentName="SET-BUTTON" behaviorName="Pressed"
7      traceabilityLink="R1.2" behaviorType="Event" outgoing="32dc9405-ae22-49e8-aa2a-63fcb53f631a"
8      incoming="0894451e-c291-4993-8ebc-49d845955b3e"/>
9    <nodes nodeId="62b61b2a-3cf5-4714-a0eb-d35267dbafc9" componentName="SAS" behaviorName="Activated"
10     traceabilityLink="R1.3" behaviorType="State Realization" incoming="32dc9405-ae22-49e8-aa2a-63fcb53f631a"/>
11    <transitions transitionId="0894451e-c291-4993-8ebc-49d845955b3e" source="1c3822c0-d2d4-49a6-819d-e723021ccfd0"
12     target="c2ef4e2e-3ecf-41bd-af99-2b6c17f8cbb9"/>
13    <transitions transitionId="32dc9405-ae22-49e8-aa2a-63fcb53f631a" source="c2ef4e2e-3ecf-41bd-af99-2b6c17f8cbb9"
14     target="62b61b2a-3cf5-4714-a0eb-d35267dbafc9"/>
15  </graphbt:GraphBT>
  
```

FIGURE 7.13: XML file for SAS-R1

For translating an individual node attributes to its OWL representation, first, we read a node tag with all node attributes such as component name, behaviour name, behaviour type, and traceability link from the XML file on line 8 of program 1. The outgoing attribute shows the next node id, and IBT node ID such as R1.1 is mentioned as a traceability link in the XML file. The next node(s) are mentioned in the transitions tag with source and target values, and it is optional because some nodes may not have any next node, such as leaf nodes of a tree.

After that, we start processing node (R1.1) attributes read from the node tag. Suppose a given IBT node attribute does not already exist in the list of individuals in the OWL file. In that case, we declare that attribute, associate it with a corresponding meta-class as an individual and add it to the corresponding list (lines 9 to 14). For example, the behaviour name “deactivated” is declared in the first two lines of the OWL file in Figure 7.14. If a given attribute already exists in the corresponding list during the translation process, that means the attribute has already been declared, and it will be ignored. The

reason for performing this check is to avoid duplication of the same attribute in the OWL file of the given IBT. The same process will be executed for all IBT node attributes, such as component name.

After that, we declare node ID on line 15 and associate it with a BT Nodes meta-class on line 16 of program 1. The node ID is unique, and it contains a requirement tag such as R1, R2, and its node index in a requirement. For example, node ID R1.1 is declared on lines 7 and 8 of the OWL file. After that, we associate node attributes such as component name, behaviour name, and behaviour type with the corresponding node ID (line 17). For example, the component name “SAS”, behaviour name “Deactivated”, and behaviour type “state realisation” are associated with node ID R1.1 on lines 9 to 11 of the OWL file.

Finally, we use `nextNode` object property to develop a relationship between nodes using source node ID and destination node ID. If there is more than one next node of a node, then line 18 will be executed more than once. Coming back to the given example, the translation of node R1.1 is shown in Figure 7.14.

```

1  <NamedIndividual rdf:about="&BERDD-Ontology;Deactivated">
2    <rdf:type rdf:resource="&BERDD-Ontology;BehaviorName"/></NamedIndividual>
3  <NamedIndividual rdf:about="&BERDD-Ontology;StateRealization">
4    <rdf:type rdf:resource="&BERDD-Ontology;BehaviorType"/></NamedIndividual>
5  <NamedIndividual rdf:about="&BERDD-Ontology;SAS">
6    <rdf:type rdf:resource="&BERDD-Ontology;ComponentName"/></NamedIndividual>
7  <NamedIndividual rdf:about="&BERDD-Ontology;R1.1">
8    <rdf:type rdf:resource="&BERDD-Ontology;BTNodes"/>
9    <BERDD-Ontology:hasComponentName rdf:resource="&BERDD-Ontology;SAS"/>
10   <BERDD-Ontology:hasBehaviorName rdf:resource="&BERDD-Ontology;Deactivated"/>
11   <BERDD-Ontology:hasBehaviorType rdf:resource="&BERDD-Ontology;StateRealization"/>
12   <BERDD-Ontology:nextNode rdf:resource="&BERDD-Ontology;R1.2"/></NamedIndividual>

```

FIGURE 7.14: OWL representation for node R1.1

7.6 An Example

In this section, we discuss step 3 of the BERDD workflow to detect and analyse RE defects. We implement this step with the RDI tool and SPARQL queries (steps 4 and 5 of the workflow). The user will enter requirements defects parameters through the RDI interface, and RDI will automatically build a query, execute it, and return a result. In this section, we listed only SPARQL query related to incompleteness defect, and the remaining queries are given in Appendix B.2.

To illustrate the effectiveness of the BERDD framework and RDI, we used a system named Ambulatory Infusion Pump (AIP). AIP is a medical device used in drug therapy for patients who are away from the direct care of health care professionals [309], and AIP requirements are shown in Table 7.6. There are 9 requirements related to the main functionality, and we took seven of them as the initial set of requirements and two of them (R8 and R9) as new requirements, as a result of the proposed RC. We took two requirements as new requirements received because of requirements change to show the applicability of BERDD during the RCM process. We opted to conduct this type of evaluation instead of an end-user survey on our proposed approach because we are interested in analysing our approach's applicability and utility.

TABLE 7.6: AIP Functional requirements

#	Requirement
R1	The system is turned on when the batteries are put in and is turned off when the batteries are out.
R2	To start the pump, when in the stopped state, the start-stop button is held down until it beeps three times and dots are displayed on the screen.
R3	Every time the system pumps 1 ml drug, and when the battery is low, it sends a single beep alarm.
R4	After a set time pump activates to pump 1ml of a drug through the line.
R5	When the volume reaches 5ml, the system does three beeps and displays a low volume message every 1ml as it counts down to empty.
R6	When there is no drug left, the pump enters stopped mode, and the system sounds a continuous beeping alarm.
R7	When the line is blocked, or air is detected, the pump is stopped and the beeper emits a continuous beep.
R8	When the battery is low, the system sends three beeps and displays a low battery message on the screen.
R9	To stop the pump, when in running state, the start-stop button is held down until it beeps three times and dots are displayed on the screen.

By following step 2a of the RDI workflow, we draw the first seven requirements and integrate them into an IBT by using the iRE tool, shown in Appendix B.3. The IBT with the initial 7 requirements has 50 nodes. The RBTs of the two new requirements (R8, R9) has a total of 18 nodes and are shown in Figure 7.15 and Figure 7.16, respectively. After that, in step 2b, we will export IBT as an XML file. In step 3, we will translate the IBT-XML file into OWL, discussed in the previous section.

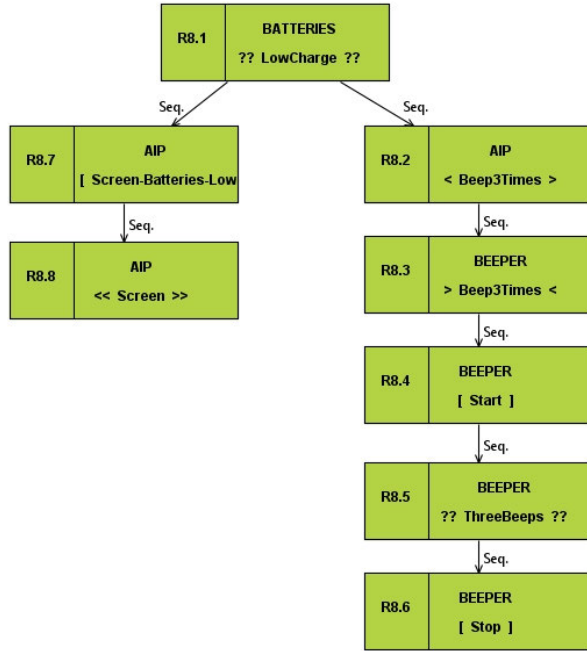


FIGURE 7.15: RBT for R8 of AIP

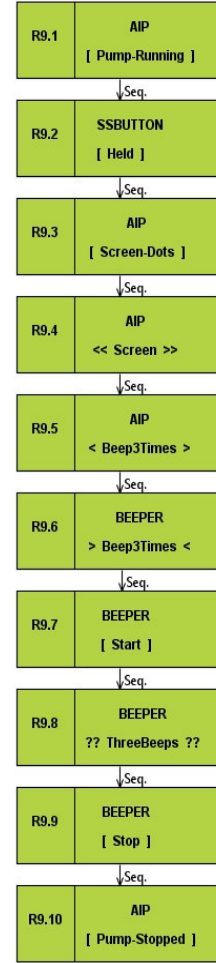


FIGURE 7.16: RBT for R9 of AIP

In steps 4 and 5, the user will enter/choose defects-related parameters and analyse the returned results. In this process, first, we check the incompleteness defects. Requirements incompleteness is one of the most common types of requirements defects that can be detected during RBT integration with an IBT. We check the completeness of R8 and R9 in the context of precondition axiom. According to the precondition axiom, the root node must match at least one node in the existing IBT to perform the integration. The result of RDI to check incompleteness of R8 and R9 is shown in Figure 7.17 and Figure 7.18, respectively. To start the checking, the user must select appropriate parameters and enter the required input values, then SPARQL queries will be generated and executed automatically. RDI shows that R9 is fulfils the precondition axiom; however, R8 is incomplete, and a precondition is missing, and it should be fixed. The SPARQL query to check the completeness of R8 is given in Program 2.

Looking at Program 2, the first four lines are default prefixes of any SPARQL query, and

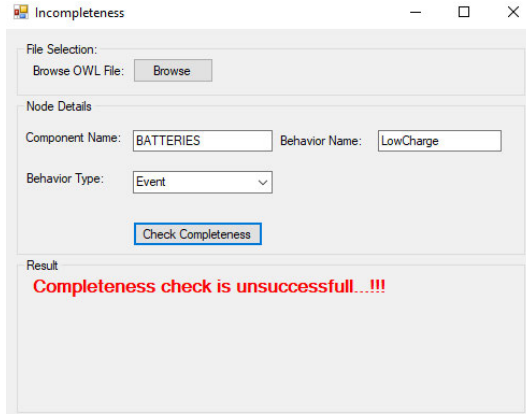


FIGURE 7.17: Incompleteness defect detection of R8 of AIP

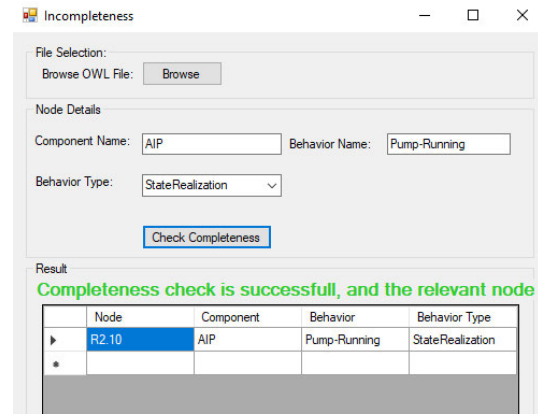


FIGURE 7.18: Incompleteness defect detection of R9 of AI

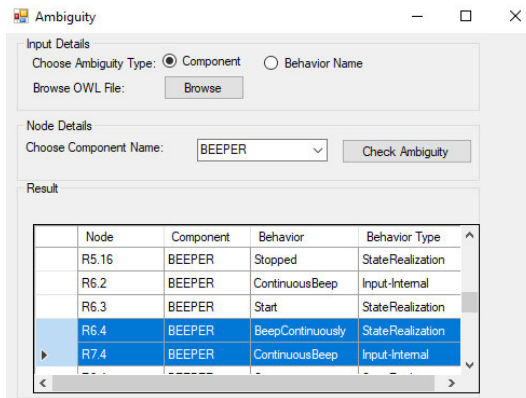
line 5 is the prefix related to our ontology defined in OWL syntax. The real query starts from line 6. The SELECT clause contains three variables (?cname, ?bname, ?btype), and the variables written in the select clause show the title of columns in the result list. The triple patterns and other conditions are enclosed in the WHERE clause that starts from line 7. The triple patterns are between lines 8 and 11. The condition on variables is in the FILTER clause between lines 12 and 14. In the FILTER clause, the @parm1, @parm2 and @parm3 are the values the user will enter through the RDI interface. This query will return an empty result for R8 because no node with these parameters exists in the AIP IBT.

1. PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2. PREFIX owl: <http://www.w3.org/2002/07/owl#>
3. PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
4. PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
5. PREFIX my: <http://www.semanticweb.org/s5105389/ontologies/2018/8/untitled-ontology-9#>
6. SELECT ?cname ?bname ?btype
7. WHERE {
8. ?node rdf:type my:BTNodes.
9. ?node my:hasComponentName ?cname.
10. ?node my:hasComponentName ?bname.
11. ?node my:hasBehaviorType ?btype.
12. FILTER (str (strafter(str(?cname), @prefix)) = @parm1)
13. FILTER (str (strafter(str(?bname), @prefix)) = @parm1)

14. FILTER (str (strafter(str(?btype), @prefix)) = @parm1)}

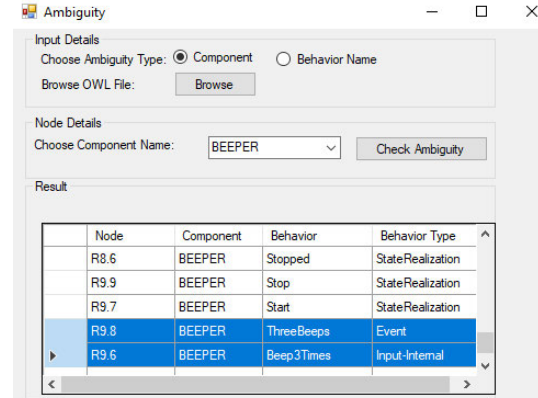
Program 2: SPARQL query to check incompleteness defect

We then look at ambiguity defects detection. We have implemented two different techniques in RDI, either based on component name or based on behaviour name. After integrating new requirements with the existing IBT, we will check ambiguity defects based on either technique. Using the component name-detection technique, we selected the BEEPER component to check whether the behaviour names associated with this component have clear and precise semantics or potential ambiguity. A thorough analysis reveals that there are two potential ambiguities, one is with R9.6 and R9.8 shown in Figure 7.20, where the behaviour names are Beep3Times and ThreeBeeps respectively, and both behaviour names refer to the same thing but with the use of two different terms. The other ambiguity is two different behaviour names (BeepContinuously, ContinuousBeep) used with the same component BEEPR, as shown in Figure 7.19. It seems that both behaviour names refer to the same thing. In both cases, requirement analysts can fix the defects based on their domain knowledge.



Node	Component	Behavior	Behavior Type
R5.16	BEEPER	Stopped	StateRealization
R6.2	BEEPER	ContinuousBeep	Input-Internal
R6.3	BEEPER	Start	StateRealization
R6.4	BEEPER	BeepContinuously	StateRealization
R7.4	BEEPER	ContinuousBeep	Input-Internal

FIGURE 7.19: AIP - ambiguity detection



Node	Component	Behavior	Behavior Type
R8.6	BEEPER	Stopped	StateRealization
R9.9	BEEPER	Stop	StateRealization
R9.7	BEEPER	Start	StateRealization
R9.8	BEEPER	ThreeBeeps	Event
R9.6	BEEPER	Beep3Times	Input-Internal

FIGURE 7.20: AIP - ambiguity detection

Following this, we check the redundancy defects. The two new RBTs created potential redundancies in the AIP system, as shown in Figure 7.21. The RDI found two potential redundancies in the AIP system, and both involve the new requirements (R8 and R9). R9 has the same precondition, event, and postcondition as in R2 and creates potential redundancy. Similarly, R8 has the same precondition, event and postcondition as R2, R5, and R9.

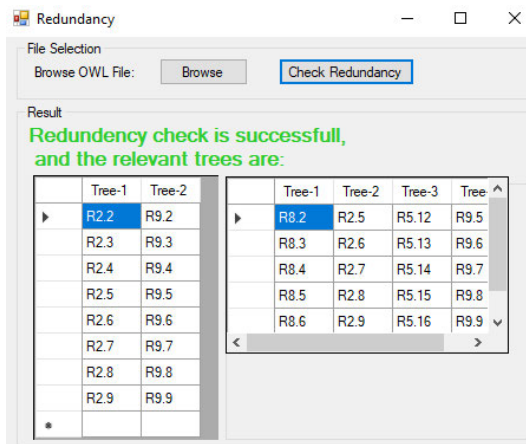


FIGURE 7.21: AIP - redundancy detection

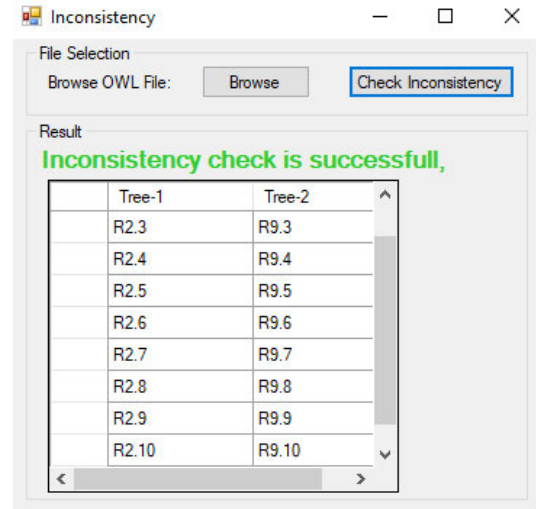


FIGURE 7.22: AIP - inconsistency detection

Finally, we check the inconsistency defects. RDI has found one potential inconsistency (shown in Figure 7.22). The detected inconsistency involves the newly introduced requirement R9 along with R2. The detailed investigation of both requirements reveals that both requirements have the same precondition and event but different postconditions.

Regarding performance evaluation of RDI, the experiment was carried out on a Dell latitude 5580 core i5 running Windows 10 with a 2.7GHz Intel Core i5 processor and 8GB DDR4 memory. It takes 7 Seconds to translate 68 nodes XML file to OWL file, 0.8 Seconds to check completeness, and 0.9 Seconds to check ambiguity. It takes 1.4 Seconds to check inconsistencies and 1.8 Seconds to check redundancies.

7.7 Discussion and Limitations

This section discusses some implications of this research, including scalability, advantages over other modelling notations, and a comparison with some existing studies. We have also discussed some limitations associated with this research.

7.7.1 Comparison with Existing Studies

Most of the research in requirements defects detection can be categorised into two groups: formal and semi-formal. In formal approaches, firstly, software requirements are translated into formal languages such as first-order logic, propositional logic, object constraint

languages etc., and then requirements defects detection is performed. In semi-formal approaches, software requirements are translated into semi-formal languages such as UML, BT before requirements defects are detected. In addition to these, NLP approaches are also used to detect requirements defects. Although we have discussed all the three groups in detail in the related work section, here, we will only compare some of the most important and relevant studies with our approach and then discuss the shortcomings of those approaches.

We are aware of a few works in ontology-based requirements defects detection that are close to our approach. For example, Nguyen et al. [170] proposed a DL based approach, and Nentwich et al. [82] proposed a first-order logic to check inconsistency and redundancy defects during the requirements analysis phase.

In semi-formal approaches, most of the existing studies have used UML to represent software requirements and performed requirement defects detection. El-Attar and Miller [310] proposed a use case based approach by using Simple Structure Use Case Description (SSUCD) to detect and eliminate the possible defects caused by inconsistencies. Kaneiwa and Satoh [78] introduced an approach for conducting restricted consistency checking of UML class diagrams by translating the identified inconsistencies to first-order logic.

After analysing the above approaches to detecting requirements defects and comparing them with our approach carefully, we have made the following conclusions:

- UML diagrams represent different views of the system through as many as 13 different types of diagrams, so most of the existing research focuses on detecting inconsistencies between different UML models rather than detecting requirement defects [311]. In contrast, behaviour engineering was developed with the goal to use a minimum set of coherent modelling notations throughout the modelling process.
- Most of the existing approaches that use formal languages translate software requirements from natural language directly into formal languages; this process requires an expert-level understanding of these languages [45]. Due to a lack of such understanding, the customers who provide the requirements may find it difficult to understand formal languages, while the engineers may not have the domain knowledge to interpret the customers' requirements correctly. That means the customer may not be able to verify if the formal representation is an accurate reflection of their

requirements. However, in this work, we first translate natural language requirements into the semi-formal modelling language, i.e. BT, and then convert BT into formal language, i.e. OWL. Customers and requirements analysts easily understand the semi-formal language used as a bridge, which overcomes the above-discussed shortcomings.

- To perform requirements defects detection effectively, good tool support is needed [278], and this becomes more important when using formal languages. Most of the existing approaches that use formal languages defined requirements defects through natural language descriptions, which might have ambiguous interpretations. Due to this, it is difficult to formalise defects, and as a result, it becomes challenging to develop an automated tool. However, in this work, we first formalise defects through semi-formal language, making it easy to design queries to detect them through an automated tool.
- Lastly, most of the existing research focuses on detecting inconsistency defects and ignoring other important requirements defects. However in this work, we address some other equally important requirements defects such as incompleteness, ambiguity, and redundancy.

7.7.2 Limitations

The proposed approach produces some promising results; however, there are a few limitations associated with it.

Firstly, we detect the four most common types of requirements defects that cover almost three-quarters of the total types of defects faced during the requirement engineering phase [274, 275]. Although these four defects cover almost three-quarters of the total number of defects, however, some types of defects are still not covered, which is a potential limitation of this approach.

Secondly, the proposed approach does not cover all aspects (100%) of these four types requirements defects. Regarding incompleteness, this research only covers the incompleteness of individual requirements and omits the other aspect of incompleteness, which is the incompleteness of overall system requirements. Regarding ambiguity, we only cover ambiguity, which involves using the same word to refer to more than one different term.

However, there are some other ambiguity definitions in the existing literature that we have not covered in this research.

Thirdly, we have covered one aspect of inconsistency and redundancy and we might have missed some other aspects of these two defects discussed in the existing research.

Lastly, it only captures requirements that could be described as "behaviours" and might not be able to handle some non-functional requirements such as constraints (a system should be able to run cross-platform).

7.8 Conclusion

In this chapter, we have investigated a novel approach to automatically detecting software requirements defects. Compared to existing approaches, this approach has significantly increased the coverage of requirement defects. This approach contains two elements.

Firstly, we formalise four types of requirements defects through a normal formed behaviour tree. To define and formalise requirements defects, we use Inertia Axiom, introduced in 6 as a foundation to define a normal formed RBT. Afterwards, the four common types of requirements defects including inconsistency, redundancy, incompleteness, and ambiguity are formally defined. Based on the formal defects definition, detection algorithms are designed.

Secondly, a new framework called BERDD is developed. In BERDD, we use BTs to model natural language software requirements, and we develop an algorithm to translate BTs into OWL. The process to translate natural language software requirements into BTs and then into a formal language has overcome the difficulties of verifying if a formal representation is faithful to its original natural language representation.

To validate the proposed approach, we have developed a tool called RDI that uses SPARQL queries to query the OWL formulation of the requirements and to detect requirements defects. The RDI tool plus the iRE tool provide end-to-end support for requirements analysts to identify requirements defects, and it they do not require knowledge of SPARQL. Finally, we apply this tool to a system, and the results show that all four types of requirements defects can be successfully detected.

Chapter 8

Conclusion

This chapter concludes the thesis by providing a summary of the contributions that are presented in the previous chapters and provides a brief insight into future research directions.

8.1 Summary of Thesis

Requirements Change Management (RCM) is an imperative activity executed during software development and software maintenance. In real-world systems, requirements changes are inevitable. Poorly managed requirements changes negatively impact the software development process; they increase the project cost and time and introduce requirements defects.

By considering the imperative need to study the RCM problem in detail, this thesis first conducted an evidence-based study, SLR, to identify RCM challenges. SLR is a very effective technique to perform an in-depth investigation of a problem. Following this, an RCM process is proposed for providing a formal set of guidelines to implement RCM. To move one, two approaches are proposed to address two highly cited RCM challenges, change impact analysis and requirements defects identified through SLR. Although all these sub-problems are segregated in nature, but all are related to the RCM domain. Therefore, this thesis explored challenges of RCM problem and proposed approaches to address a couple of key challenges of the RCM problem.

By considering the above-mentioned problems related to RCM, four research objectives are defined. In line with these objectives, the contributions of this thesis are summarised as follows.

Aiming the first objective of this research, Chapter 3 presents an SLR based approach to identifying RCM challenges both in in-house software development and Global Software Development (GSD). After that, a questionnaire-based survey is used to get industry practitioners' opinions related to our literature findings. We have identified 9 challenges for RCM in in-house software development and 3 additional challenges that are specific to RCM process in GSD. Based on both data sets, a chi-square test shows that cost/time estimation, requirement consistency, change prioritisation, artefacts documents management, and user involvement are more challenging in GSD as compared to in-house software development. Similarly, user involvement and change control board management are more challenging in centralised project management structures than distributed project management structures in GSD projects. The t-test of independence used for comparative analysis of both data sets, reveals that the research results and industry opinions are consistent regarding RCM challenges.

To achieve the second objective of this research, a novel RCM process is presented in Chapter 4. The process is defined in the format of an ISO/IEC standard. This work further expands the breadth of our work related to RCM challenges.

To thoroughly study this process, a seven-stage theoretical model is presented. Moreover, Composition Trees (CT) are used to compare the proposed process with Configuration Management Process (CMP), which is the most relevant process defined in ISO/IEC 12207:2017. In addition, a mapping has been developed to verify that the RCM process outcomes match the RCM challenges identified in Chapter 3. This mapping helps industry practitioners to understand what challenges they might face in achieving a particular process outcome. The mapping has been evaluated through a questionnaire-based survey in the industry. The results show that more than 90% of the industry practitioners agree with most of the mapping, except for one challenge. In the end, a set of best practices has also been collected from industry professionals.

Chapter 5 presents an approach to performing Change Impact Analysis (CIA), which is about the third objective of the thesis. CIA is one of the most cited challenges in RCM,

identified in this research. CIA helps to estimate the impacts of proposed changes on other software artefacts in a software system.

In this chapter, we have proposed a novel behaviour engineering-based approach to perform change impact analysis. In this work, we used an Integrated Behaviour Tree (IBT) to model system requirements and proposed algorithms to convert an IBT into an Integrated Composition Tree (ICT), and then converted the ICT into a Requirements Component Dependency Network (RCDN). The RCDN helps to identify a set of potentially impacted components. After that, we used CIDs retrieved from the IBT and the RCDN to investigate which components are impacted and the level of the impact. Lastly, we proposed a change impact indicator metric to quantify the change impact. The indicator gives more objective evidence to estimate the development cost of a proposed change.

Chapter 6 enhances the BE approach. It provides a more stable foundation to translate requirements from natural languages to BTs. This enhancement works as a pre-step to detecting requirements defects, which are discussed in Chapter 7. The two chapters address the fourth objective of this thesis and increase the depth of our work related to RCM because requirements defects such as inconsistency are among the RCM challenges identified in our research. On the other hand, requirements defects also arise during the requirements elicitation and analysis stage, therefore, detecting requirement defects is also a critical activity of Requirements Engineering (RE). Therefore, this work increased the breadth of our work to the RE phase.

In Chapter 6, we propose a two-step formal model to generate valid Behaviour Trees (BTs). Firstly, we defined normal formed RBTs based on the newly introduced Inertia Axiom, and then defined valid BTs as a normal formed Requirements Behaviour Tree (RBT) or an IBT integrated from a finite set of normal formed RBTs. Secondly, we developed a Context-Free Grammar (CFG) to generate valid BTs. This chapter uses a commonly used case study to demonstrate the applicability of this research. Moreover, a tool was developed to support and validate the proposed formal model. The tool results show that the proposed grammar successfully identified all syntactic errors in the original requirements of the case study and those requirements defects that had not been reported in previous studies.

In Chapter 7, we propose a two-step approach to formalise and detect requirements defects. Firstly, we formalised the four most common types of requirements defects, including inconsistency, redundancy, incompleteness, and ambiguity through normalised behaviour trees. And then, based on the formalisation, relevant algorithms were designed to detect those defects. Secondly, a behaviour engineering-based approach was introduced. In this approach, we used BTs to model natural language software requirements, and then developed an algorithm to translate these BTs into Web Ontology Language (OWL). The process of translating natural language software requirements into BTs and then into a formal language overcame the difficulty of verifying whether a formal representation is equivalent to its original natural language form. To verify this approach, we developed a tool and applied it to a real-world case study. The tool results show that all four types of requirements defects can be successfully detected by the tool.

To sum up, this thesis first conducted an SLR to identify RCM challenges. The analysis performed in Chapter 3 based on SLR and questionnaire survey findings identifies potential future research directions. We believe that these SLR findings form a solid foundation for proposing approaches to address different aspects of the RCM problem in both in-house and GSD domains. Moreover, by considering findings from SLR and questionnaire survey, this thesis proposed approaches to address a couple of key RCM challenges, change impact analysis and requirements defects detection. We believe these proposed approaches can become a foundation for solving the discussed problems at an extended scale in real-world systems.

8.2 Future Work

This thesis starts from conducting an SLR to identify RCM challenges and then explores four different approaches to tackle some of those challenges. Even though all those attempts have received positive results, they need to be continuously studied in the future.

Regarding identifying RCM challenges, we suggest:

- There is a need to include Gray Literature (GL) along with SLR to further enhance the scope of this approach. Over recent years, GL stands out as an essential source of knowledge to be used alone or to complement research findings within the traditional literature.

- There is a need to conduct a further empirical study to establish the inter-dependencies between the key challenges and their impacts on the success or failure of a project.

For the proposed RCM process, we suggest:

- It is worthwhile to closely work with the ISO/IEC committee to further refine the proposed RCM process, so it could be officially included in a later version of the standards.
- The mapping between RCM process outcomes and RCM challenges can be further validated by applying the proposed process in real-world industry projects.

Regarding the CIA, we suggest the following future works:

- There is a need to develop a prototyping tool to further enhance the proposed approach. It will also help to automate the proposed approach. After the tool building, it also be worthwhile to conduct a questionnaire with industry practitioners to empirically validate the developed tool.
- The applicability of the proposed approach in the GSD paradigm is also worthy of future investigation, especially in the context of different project management structures followed in the GSD projects.

Lastly, regarding requirements defects detection, we suggest:

- For BT formalisation, it worthwhile to extend BT modelling notation to further expand the scope of this approach. Furthermore, to validate the proposed approach, it will also be useful to perform empirical validation through industrial case studies.
- Regarding requirements defects detection, it will be valuable to conduct a questionnaire with industry professionals to empirically validate the proposed framework and developed tool.
- It is also worthwhile to increase the coverage of this approach by adding the definition and detection algorithms for other types of requirements defects.
- It will be useful to explore these requirements defects through composition trees that are used to model the static aspect of software-intensive systems.

Appendix A

Appendices for Chapter 3

A.1 Search Strings for Different Databases

A.1.1 IEEE Xplore

RQ1

((("Challenges" OR "problems" OR "difficulties" OR "complications" OR "obstacles" OR "barriers" OR "hurdles" OR "risks") AND "requirements change" OR "requirements volatility" OR "requirements creep" OR "requirements change management" OR "Requirements change difficulties" OR "requirements change analysis" OR "requirements change identification/type" OR "requirements change models/processes") AND "in-house software development" OR "onshore software development" OR "onsite software development")

RQ3

((("Challenges" OR "problems" OR "difficulties" OR "complications" OR "obstacles" OR "barriers" OR "hurdles" OR "risks") AND "requirements change" OR "requirements volatility" OR "requirements creep" OR "requirements change management" OR "Requirements change difficulties" OR "requirements change analysis" OR "requirements change identification/type" OR "requirements change models/processes") AND "Global software development projects" OR "global project management" OR "GSD" OR "Global Software Development" OR "Offshore software development" OR "Offshore Outsourcing" OR "distributed software development")

A.1.2 Science Direct

RQ1

(“Challenges” OR “problems” OR “difficulties” OR “complications” OR “obstacles” OR “barriers” OR “hurdles” OR “risks”) AND (“requirements change” OR “requirements volatility” OR “requirements creep” OR “requirements change management” OR “Requirements change difficulties” OR “requirements change analysis” OR “requirements change identification/type” OR “requirements change models/processes”) AND (“in-house software development” OR onshore software development” OR “onsite software development”)[All Sources(Computer Science)]

RQ3

(“Challenges” OR “problems” OR “difficulties” OR “complications” OR “obstacles” OR “barriers” OR “hurdles” OR “risks”) AND (“requirements change” OR “requirements volatility” OR “requirements creep” OR “requirements change management” OR “Requirements change difficulties” OR “requirements change analysis” OR “requirements change identification/type” OR “requirements change models/processes”) AND (“Global software development projects” OR “global project management” OR “GSD” OR “Global Software Development” OR “Offshore software development” OR “Offshore Outsourcing” OR “distributed software development”)[All Sources(Computer Science)]

A.1.3 SpringerLink

RQ1

(“Challenges” OR “problems” OR “difficulties” OR “complications” OR “obstacles” OR “barriers” OR “hurdles” OR “risks”) AND (“requirements change” OR “requirements volatility” OR “requirements creep” OR “requirements change management” OR “Requirements change difficulties” OR “requirements change analysis” OR “requirements change identification/type” OR “requirements change models/processes”) AND (“in-house software development” OR onshore software development” OR “onsite software development”)[Within Computer Science]

RQ3

(“Challenges” OR “problems” OR “difficulties” OR “complications” OR “obstacles” OR “barriers” OR “hurdles” OR “risks”) AND (“requirements change” OR “requirements

volatility” OR “requirements creep” OR “requirements change management” OR “Requirements change difficulties” OR “requirements change analysis” OR “requirements change identification/type” OR “requirements change models/processes”) AND (“Global software development projects” OR “global project management” OR “GSD” OR “Global Software Development” OR “Offshore software development” OR “Offshore Outsourcing” OR “distributed software development”)[Within Computer Science]

A.1.4 ACM

RQ1

((“Challenges” OR “problems” OR “difficulties” OR “complications” OR “obstacles” OR “barriers” OR “hurdles” OR “risks”) AND (“requirements change” OR “requirements volatility” OR “requirements creep” OR “requirements change management” OR “Requirements change difficulties” OR “requirements change analysis” OR “requirements change identification/type” OR “requirements change models/processes”) AND (“in-house software development” OR “onshore software development” OR “onsite software development”))

RQ3

((“Challenges” OR “problems” OR “difficulties” OR “complications” OR “obstacles” OR “barriers” OR “hurdles” OR “risks”) AND (“requirements change” OR “requirements volatility” OR “requirements creep” OR “requirements change management” OR “Requirements change difficulties” OR “requirements change analysis” OR “requirements change identification/type” OR “requirements change models/processes”) AND (“Global software development projects” OR “global project management” OR “GSD” OR “Global Software Development” OR “Offshore software development” OR “Offshore Outsourcing” OR “distributed software development”))

A.2 List of Primary Studies in SLR

A01: S. Ahn and K. Chong, “Requirements change management on feature-oriented requirements tracing,” in *International Conference on Computational Science and Its Applications*, 2007, pp. 296-307.

A02: S. Jayatilleke, R. Lai, and K. Reed, “A method of requirements change analysis,”

Requirements Engineering, vol. 23, no. 4, pp. 493-508, 2018.

A03: A. AlSanad and A. Chikh, "The Impact of Software Requirement Change—A Review," in *New Contributions in Information Systems and Technologies*, 2015, pp. 803-812.

A04: M. Kausar and A. Al-Yasiri, "Using Distributed Agile Patterns for Supporting the Requirements Engineering Process," in *Requirements Engineering for Service and Cloud Computing*, 2017, pp. 291-316.

A05: I. Navarro, N. Leveson, and K. Lunqvist, "Semantic decoupling: reducing the impact of requirement changes," *Requirements engineering*, vol. 15, no. 4, pp. 419-437, 2010.

A06: L. Lin and J. H. Poore, "Pushing requirements changes through to changes in specifications," *Frontiers of Computer Science in China*, vol. 2, no. 4, pp. 331-343, 2008.

A07: J. J. Lin and Y.-S. Lin, "Research and Development of a CMMI-Compliant Requirement Management System for Software Engineering," in *International Conference on Computer Supported Cooperative Work in Design*, 2007, pp. 76-86.

A08: K.-D. Mu, W. Liu, Z. Jin, J. Hong, and D. Bell, "Managing software requirements changes based on negotiation-style revision," *Journal of Computer Science and Technology*, vol. 26, no. 5, p. 890, 2011.

A09: M. A. Chauhan and C. W. Probst, "Architecturally Significant Requirements Identification, Classification and Change Management for Multi-tenant Cloud-Based Systems," in *Requirements Engineering for Service and Cloud Computing*: Springer, 2017, pp. 181-205.

A10: D. Damian, J. Chisan, L. Vaidyanathasamy, and Y. Pal, "Requirements engineering and downstream software development: Findings from a case study," *Empirical Software Engineering*, vol. 10, no. 3, pp. 255-283, 2005.

A11: J. Lockerbie, N. A. M. Maiden, J. Engmann, D. Randall, S. Jones, and D. Bush, "Exploring the impact of software requirements on system-wide goals: a method using satisfaction arguments and i* goal modelling," *Requirements Engineering*, vol. 17, no. 3, pp. 227-254, 2012.

A12: M. Heindl and S. Biffl, "Modeling of requirements tracing," in *Balancing Agility and Formalism in Software Engineering*: Springer, 2008, pp. 267-278.

A13: T. O. de Jesus and M. S. Soares, "An Event-Based Technique to Trace Requirements Modeled with SysML," in *International Conference on Computational Science and Its Applications*, 2017, pp. 145-159.

- A14: B. B. Chua, D. V. Bernardo, and J. Verner, "Criteria for estimating effort for requirements changes," in *European Conference on Software Process Improvement*, 2008, pp. 36-46.
- A15: T. Gorschek and M. Svahnberg, "Requirements experience in practice: Studies of six companies," in *Engineering and Managing Software Requirements*, 2005, pp. 405-426.
- A16: C. Ting, "The Control and Measure of Requirements Stability in Software Project," in *International Conference on Information and Management Engineering*, 2011, pp. 387-394.
- A17: J. Shah and N. Kama, "Issues of Using Function Point Analysis Method for Requirement Changes During Software Development Phase," in *Asia Pacific Requirements Engineering Conference*, 2017, pp. 156-163.
- A18: 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.
- A19: A. A. Khan, S. Basri, and P. Dominc, "A proposed framework for communication risks during RCM in GSD," *Procedia-Social and Behavioral Sciences*, vol. 129, pp. 496-503, 2014.
- A20: S. Jayatilleke and R. Lai, "A systematic review of requirements change management," *Information and Software Technology*, vol. 93, pp. 163-185, 2018.
- A21: D. Lloyd, R. Moawad, and M. Kadry, "A supporting tool for requirements change management in distributed agile development," *Future Computing and Informatics Journal*, vol. 2, no. 1, pp. 1-9, 2017.
- A22: R. A. Aziz and B. Wong, "The interplay between requirements relationships knowledge and requirements change towards software project success: an assessment using partial least square (PLS)," *Procedia Computer Science*, vol. 46, pp. 732-741, 2015.
- A23: A. Goknil, I. Kurtev, K. van den Berg, and W. Spijkerman, "Change impact analysis for requirements: A metamodeling approach," *Information and Software Technology*, vol. 56, no. 8, pp. 950-972, 2014.
- A24: C.-Y. Chen and P.-C. Chen, "A holistic approach to managing software change impact," *Journal of Systems and Software*, vol. 82, no. 12, pp. 2051-2067, 2009.
- A25: R. Sugden and M. Strens, "Strategies, tactics and methods for handling change," in *ecbs*, 1996.
- A26: A. A. Khan, J. Keung, S. Hussain, and K. E. Bennin, "Effects of geographical, socio-cultural and temporal distances on communication in global software development during requirements change management a pilot study," in *International Conference on*

Evaluation of Novel Approaches to Software Engineering, 2015, pp. 159-168.

A27: N. Assawamekin, "An ontology-based approach for multiperspective requirements traceability between analysis models," in IEEE/ACIS 9th International Conference on Computer and Information Science, 2010, pp. 673-678.

A28: L. Lin, S. J. Prowell, and J. H. Poore, "The impact of requirements changes on specifications and state machines," *Software: Practice and Experience*, vol. 39, no. 6, pp. 573-610, 2009.

A29: M. Shafiq et al., "Effect of project management in requirements engineering and requirements change management processes for global software development," *IEEE Access*, 2018.

A30: A. Teka, N. Condori-Fernandez, I. Kurtev, D. Quartel, and W. Engelsman, "Change impact analysis of indirect goal relations: Comparison of NFR and TROPOS approaches based on industrial case study," in IEEE Model-Driven Requirements Engineering Workshop (MoDRE 2012), 2012, pp. 58-67.

A31: M. Weber and J. Weisbrod, "Requirements engineering in automotive development-experiences and challenges," in IEEE International Conference on Requirements Engineering, 2002, pp. 331-340.

A32: 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 IEEE 24th International Requirements Engineering Conference (RE), 2016, pp. 126-135.

A33: F. Mokammel, E. Coatanéa, M. Bakhouya, F. Christophe, and S. Nonsiri, "Impact analysis of graph-based requirements models using PageRank algorithm," in IEEE International Systems Conference (SysCon) , 2013, pp. 731-736.

A34: S. Imtiaz, N. Ikram, and S. Imtiaz, "Impact analysis from multiple perspectives: Evaluation of traceability techniques, in third International Conference on Software Engineering Advances, 2008, pp. 457-464.

A35: I. Keshta, M. Niazi, and M. Alshayeb, "Towards Implementation of Requirements Management Specific Practices (SP1. 3 and SP1. 4) for Saudi Arabian Small and Medium Sized Software Development Organizations," *IEEE Access*, vol. 5, pp. 24162-24183, 2017.

A36: A. A. Khan, S. Basri, and P. Dominic, "Communication risks in GSD during RCM: Results from SLR," in International Conference on Computer and Information Sciences, 2014, pp. 1-6.

A37: A. Khatoon, Y. H. Motla, M. Azeem, H. Naz, and S. Nazir, "Requirement change management for global software development using ontology," in IEEE 9th International

Conference on Emerging Technologies (ICET) , 2013, pp. 1-6.

A38: Y. Hafeez et al., “A requirement change management framework for distributed software environment,” in 7th International Conference on Computing and Convergence Technology (ICCCT), 2012, pp. 944-948.

A39: Y. Jin, J. Zhang, P. Ma, W. Hao, S. Luo, and Z. Li, “Applying pagerank algorithm in requirement concern impact analysis,” in 33rd Annual IEEE International Computer Software and Applications Conference, 2009, vol. 1, pp. 361-366.

A40: 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,” IEEE Workshop on Application-Specific Software Engineering Technology, 1998, pp. 34-39.

A41: V. Sinha, B. Sengupta, and S. Chandra, “Enabling collaboration in distributed requirements management,” IEEE software, vol. 23, no. 5, pp. 52-61, 2006.

A42: R. Lai and N. Ali, “A requirements management method for global software development,” AIS: Advances in Information Sciences, vol. 1, no. 1, pp. 38-58, 2013.

A43: H. Ahmed, A. Hussain, and F. Baharom, “Current challenges of requirement change management,” Journal of Telecommunication, Electronic and Computer Engineering (JTEC), vol. 8, no. 10, pp. 173-176, 2016.

A.3 RCM Challenges Categories Identified via SLR

List of challenges for RCM process in in-house software development	
Final list of RCM Challenges	RCM Challenges – sub categories
Impact analysis	<ul style="list-style-type: none"> • Impact analysis • Change consequences
Cost/Time estimation	<ul style="list-style-type: none"> • Cost estimation • Time estimation • Effort estimation • Change cost
Requirements traceability	<ul style="list-style-type: none"> • Requirements traceability
Artefacts documents management	<ul style="list-style-type: none"> • Artefacts documents management • Artefacts documents updation • SDLC products management • Documents consistency management
Requirement dependency	<ul style="list-style-type: none"> • Requirements dependency • Requirements inter-dependency

Requirements consistency	<ul style="list-style-type: none"> • Requirements consistency • Change conflicts with existing requirements
Change prioritisation	<ul style="list-style-type: none"> • Change prioritisation
User involvement	<ul style="list-style-type: none"> • User involvement
System instability	<ul style="list-style-type: none"> • System instability
List of RCM challenges for GSD projects	
Communication and coordination	<ul style="list-style-type: none"> • Communication and coordination • Coordination
Knowledge management and sharing	<ul style="list-style-type: none"> • Knowledge management • Knowledge sharing • Use of similar terminology
Change control board management	<ul style="list-style-type: none"> • Change control board management

A.4 Questionnaire Survey

Practitioner's Details

Position/ Job Title:

Experience in Years:

--Select Experience--

Email:

Company's country in which it is located?:

--Select One--

Section 1

What is primary business function of your company? (you may tick more than one option)

☐ In-house Software development

☐ Outsource/GSD development

What is the scope of your company? (Please tick as appropriate)

☐ National

☐ Multinational

☐ Don't Know

☐ Other:

What type of Project Management Model typically used in your organization for GSD projects?

- ☐ Centralized Project Management-- All or most of the team members report directly to project manager, who may work at other geographical site and responsible for the planning and execution of projects.
- ☐ Distributed Project Management with Local Coordinators-- All or most of the team members report to local coordinators, who are responsible for the planning and execution of sub-projects or work packages and report to project manager.

Approximately how many staff are employed by your company? (Please tick as appropriate)

- ☐ Less than 25
- ☐ 26-199
- ☐ Greater than 200
- ☐ Not Sure

Approximately how many staff are employed directly in the production/maintenance of software? (Please tick as appropriate)

- ☐ Less than 25
- ☐ 26-199
- ☐ Greater than 200
- ☐ Not sure

Approximately how many different geographical sites are used by your company?

- ☐ 1-5
- ☐ 6-100
- ☐ Greater than 10
- ☐ Not sure

What type of systems are your company concerned with? (You may tick more than one)

What type of systems are your company concerned with? (You may tick more than one)

- ☐ Safety Critical
- ☐ Business Systems
- ☐ Telecommunications
- ☐ Real Time Systems
- ☐ Data Processing
- ☐ System Software
- ☐ Windows-based
- ☐ Embedded Systems
- ☐ Android Applications
- ☐ IOS Applications
- ☐ Other:

Section 2

2.1. Evaluation of the challenges of Software Requirement Change Management Processes

For each challenge, please select the appropriate box based on your experience in software projects.

Challenges of software requirement Change Management Process				
	Strongly Agree	Agree	Disagree	Strongly Disagree
Impact Analysis	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Cost/Time Estimation	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Requirements Traceability	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Artifacts Documents Managements	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Requirements Dependency	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Requirments Consistency	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Change Prioritisation	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
User Involvement	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
System Instability**	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

**System Instability-- means that when the requested change is in process, how the functionality impacted by requested change will be handled?

2.3. Please list the challenges that you think are important for requirement change management in addition to the above challenges identified from literature.

Section 3 Global Software Development

3.1. Evaluation of the challenges of Software Requirement Change Management Processes in GSD projects.

For each challenge, please select the appropriate box based on your experience in GSD projects.

Challenges faced to manage requirement change in GSD Projects				
	Strongly Agree	Agree	Disagree	Strongly Disagree
Communication & Coordination	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Knowledge Management & Sharing	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Change Control Board Management	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

3.3. Please list the challenges that you think are important for requirement change management in addition to the above challenges identified from literature in GSD projects.

A.5 Ethical Clearance to Conduct Questionnaire Survey

Full Research Ethics Clearance 2018/591

RIMS Griffith <rims@griffith.edu.au>

Thu 7/26/2018 10:45 AM

To: Zhe Wang <zhe.wang@griffith.edu.au>; Larry Wen <l.wen@griffith.edu.au>; Sajid Anwer <sajid.anwer@griffithuni.edu.au>

Cc: research-ethics <research-ethics@griffith.edu.au>; Kim Madison <k.madison@griffith.edu.au>

GRIFFITH UNIVERSITY HUMAN RESEARCH ETHICS REVIEW

Dear Dr Larry Wen

I write further to the additional information provided in relation to the provisional approval granted to your application for ethical clearance for your project "Empirical Analysis of Requirement change management challenges in Global Software Development" (GU Ref No: 2018/591).

This is to confirm that this response has addressed the comments and concerns of the HREC.

The ethics reviewers resolved to grant your application a clearance status of "Fully Approved".

Consequently, you are authorised to immediately commence this research on this basis.

Regards

Kim Madison | Human Research Ethics

Office for Research

Griffith University | Nathan | QLD 4111 | Level 0, Bray Centre

T +61 7 373 58043 | email k.madison@griffith.edu.au

A.6 Participants Demographic Details

Job Title	Experience (Years)	Company size	No. of sites	Types of Systems
Project Manager	9-11	Less than 25	1-5	Safety Critical, Real Time systems
Development Manager	7-8	Less than 25	6-10	Business Systems, Android Applications
Software Engineer	3-5	Greater than 200	1-5	Safety Critical, Business Systems, Data processing
Requirements Engineer	3-5	Greater than 200	1-5	Safety Critical, Real Time systems, Data processing

System Manager	7-8	Greater than 200	1-5	System Software, Android Applications
Software Engineer	3-5	Greater than 200	6-10	Business Systems, Windows based, IOS Applications
Software Engineer	0-2	26-199	1-5	Business Systems
Business Intelligence Engineer	3-5	Less than 25	Greater than 10	Business Systems, Real Time systems, Data processing
Sr. Software Engineer	3-5	26-199	6-10	Real Time systems, Android Applications, IOS Applications
BI Developer	3-5	26-199	6-10	Business Systems, Data processing
Team Lead	7-8	Greater than 200	6-10	Business Systems, Telecommunications, Real Time systems
Sr. PeopleSoft Consultant	7-8	26-199	6-10	Business Systems, System Software, IOS Applications
Business Intelligence Solution Developer	3-5	Greater than 200	6-10	Business Systems, Data processing
Sr. Developer	9-11	Greater than 200	6-10	Business Systems, Data processing
Project Manager	9-11	Less than 25	1-5	Safety Critical, Data processing, Android Applications, IOS Applications
Senior Software Engineer	7-8	Greater than 200	6-10	Business Systems, Telecommunications, Data processing
Sr. Software Engineer	3-5	26-199	1-5	Business Systems
Development Lead	7-8	Less than 25	1-5	Safety Critical
Software Engineer	3-5	26-199	6-10	Business Systems
Software Engineer	3-5	Less than 25	1-5	Business Systems, Data processing, System Software

Software Quality Analyst	3-5	Greater than 200	6-10	Business Systems, Real Time systems, Data processing
Sr. Requirements Engineer	3-5	Greater than 200	Greater than 10	Safety Critical, Windows based, IOS Applications
Team Lead	7-8	26-199	1-5	Business Systems
Software Engineer	3-5	26-199	6-10	Real Time systems, Data processing, System Software
Software Engineer	0-2	Less than 25	6-10	Business Systems, Telecommunications, Real Time systems
Sr. Software Engineer	3-5	Greater than 200	Greater than 10	Safety Critical, Business Systems, Embedded Systems
Software Engineer	3-5	Greater than 200	1-5	Safety Critical, Business Systems, System Software, Embedded Systems
Software Engineer	7-8	Greater than 200	6-10	Business Systems, Data processing
Sr. Software Engineer	9-11	Greater than 200	6-10	Real Time systems, Data processing, System Software
Team Lead	7-8	Greater than 200	6-10	Business Systems, Real Time systems
Sr. Software Engineer	7-8	Greater than 200	6-10	Safety Critical, Telecommunications, System Software, Embedded Systems
Software Design Engineer	3-5	Greater than 200	6-10	Business Systems, Data processing, Embedded Systems
Development Lead	7-8	26-199	Greater than 10	Business Systems, Real Time systems, Android Application
Project Manager	9-11	26-199	6-10	Business Systems, Android Applications, IOS Applications
Team Lead	7-8	26-199	6-10	Business Systems, Real Time systems
Sr. Software Engineer	7-8	Less than 25		Business Systems, Data processing, System Software
Requirement Engineer	3-5	Less than 25	1-5	Business Systems, Data processing, System Software

Sr. Software Engineer	3-5	Less than 25	1-5	Business Systems
Sr. ISO Developer	3-5	26-199	Greater than 10	Business Systems, Data processing, Android Applications, IOS Applications
Development Lead	3-5	Less than 25	6-10	Business Systems, Telecommunications, Real Time systems
Software Engineer	0-2	Less than 25	1-5	Business Systems, Telecommunications, Real Time systems, System Software
Project Manager	9-11	Greater than 200	1-5	Business Systems, Telecommunications
Sr. PHP Developer	3-5	Less than 25	Greater than 10	Business Systems, Android Applications, IOS Applications
Team Lead	7-8	Greater than 200	1-5	Business Systems, Real Time systems, Data processing, Android Applications
Server Engineer	0-2	26-199	1-5	Games
Sr. Software Engineer	3-5	Greater than 200	No Site information in in-house	Windows based
Team Lead	7-8	Greater than 200		Windows based
Sr. Requirements Engineer	7-8	Greater than 200		Business Systems, Real Time systems, Android Applications
Software Engineer	3-5	Greater than 200		Safety Critical, Business Systems, Real Time systems
Software Engineer	3-5	26-199		Security
Project Manager	9-11	26-199		System Software
Software Engineer	3-5	Less than 25		IOS Applications
Development Lead	7-8	26-199		Business Systems, Data processing, IOS Applications

Principal Software Engineer	9-11	Greater than 200		Safety Critical, Business Systems, Real Time systems
Sr. Software Engineer	7-8	Greater than 200		Business Systems, Data processing, Android Applications
Team Lead	7-8	Less than 25		Business Systems, Android Applications
Software Engineer	3-5	Less than 25		Safety Critical, Business Systems, Real Time systems, Data processing
Team Lead	7-8	Less than 25		System Software, Windows based, Android Applications, IOS Applications
Sr. Software Engineer	7-8	Less than 25		Business Systems, System Software, Android Applications, IOS Applications
Software Engineer	0-2	Less than 25		Block chain
Software Engineer	3-5	26-199		Business Systems, Real Time systems, Data processing, System Software
Principal Software Engineer	7-8	26-199		Business Systems, Real Time systems, Data processing, System Software
Project Manager	9-11	Greater than 200		Data processing, Windows based, Android Applications
Software Engineer	3-5	Greater than 200		Safety Critical, Business Systems, Real Time systems, System Software
Sr. Software Engineer	3-5	Greater than 200		Safety Critical, Business Systems, Real Time systems
Development Lead	7-8	Greater than 200		Business Systems, Real Time systems, Data processing, Android Application
Software Engineer	3-5	Greater than 200		Business Systems, Real Time systems, System Software, Android Applications
Sr. Software engineer	3-5	26-199		Branch-less banking

Team Lead	7-8	Greater than 200		Safety Critical, Business Systems, Real Time systems, Data processing
-----------	-----	------------------	--	---

A.7 SLR Primary Studies Quality Assessment Results

#	Paper ID	Q1	Q2	Q3	Q4	Q5	Total Score	Quality (%)
1	A01	1	0.5	0.5	1	0.5	3.5	70
2	A02	1	1	1	1	0.5	4.5	90
3	A03	1	0.5	1	0.5	0	3	60
4	A04	1	1	0.5	1	0.5	4	80
5	A05	1	0.5	1	0.5	0.5	3.5	70
6	A06	1	0.5	1	0.5	0.5	3.5	70
7	A07	1	0.5	0.5	1	0	3	60
8	A08	1	0.5	1	1	1	4.5	90
9	A09	1	0.5	0.5	1	0	3	60
10	A10	1	0.5	1	1	0.5	4	80
12	A12	0.5	0.5	1	1	0.5	3.5	70
13	A13	1	0.5	1	1	0.5	4	80
14	A14	1	0.5	1	1	0.5	4	80
15	A15	0.5	0.5	1	1	0	3	60
16	A16	1	1	0.5	1.0`	0	3.5	70
17	A17	1	0.5	1	1	0.5	4	80
18	A18	1	1	1	1	0.5	4.5	90
19	A19	1	0.5	1	1	0.5	4	80
20	A20	1	1	1	1	1	5	100
21	A21	0.5	1	0.5	1	0	3	60
22	A22	1	1	1	0.5	0.5	4	80
23	A23	1	0.5	1	0.5	0.5	3.5	70
24	A24	1	0.5	1	1	0	3.5	70
25	A25	1	0.5	1	0.5	0.5	3.5	70
26	A26	1	1	1	1	0.5	4.5	90
27	A27	0.5	0.5	1	1	0.5	3.5	70

28	A28	1	1	1	1	0.5	4.5	90
29	A29	1	1	0.5	1	0.5	4.5	90
30	A30	0.5	0.5	1	1	0	3	60
31	A31	1	0.5	1	1	0.5	4	80
32	A32	1	1	1	1	0.5	4.5	90
33	A33	1	0.5	1	1	0.5	4	80
34	A34	1	0.5	1	1	0.5	4	80
35	A35	1	1	0.5	1	1	4.5	90
36	A36	1	1	1	1	0.5	4.5	90
37	A37	1	0.5	0.5	1	0.5	3.5	70
38	A38	0.5	0.5	1	1	0	3	60
39	A39	0.5	1	0.5	1	0	3	60
40	A40	1	0.5	0.5	1	0	3	60
41	A41	1	0.5	1	1	0.5	4	80
42	A42	1	1	0.5	1	0.5	4	80
43	A43	1	0.5	0.5	1	0	3.5	70
Average		0.91	0.69	0.84	0.93	0.39	3.77	

Appendix B

Appendices for Chapter 7

B.1 R1 OWL File

```
1 <?xml version="1.0"?>
2 <!DOCTYPE rdf:RDF [
3   <ENTITY owl "http://www.w3.org/2002/07/owl#"
4   <ENTITY xsd "http://www.w3.org/2001/XMLSchema#"
5   <ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#"
6   <ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
7   <ENTITY BERDD-Ontology "http://www.semanticweb.org/s5105389/ontologies/2020/1/untitled-ontology-33#"
8 ]>
9 <rdf:RDF xmlns = "http://www.semanticweb.org/s5105389/ontologies/2020/1/untitled-ontology-33#"
10   xml:base = "http://www.semanticweb.org/s5105389/ontologies/2020/1/untitled-ontology-33"
11   xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
12   xmlns:owl = "http://www.w3.org/2002/07/owl#"
13   xmlns:xsd = "http://www.w3.org/2001/XMLSchema#"
14   xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
15   xmlns:BERDD-Ontology="http://www.semanticweb.org/s5105389/ontologies/2020/1/untitled-ontology-33#"
16   <Ontology rdf:about="http://www.semanticweb.org/s5105389/ontologies/2020/1/untitled-ontology-33/"
17     <!-- Classes -->
18     <Class rdf:about="<BERDD-Ontology:BTNodes"/>
19     <Class rdf:about="<BERDD-Ontology:BehaviorName"/>
20     <Class rdf:about="<BERDD-Ontology:BehaviourType"/>
21     <Class rdf:about="<BERDD-Ontology:ComponentName"/>
22     <!-- Object Properties -->
23     <ObjectProperty rdf:about="<BERDD-Ontology:hasBehaviorName"/>
24     <ObjectProperty rdf:about="<BERDD-Ontology:hasBehaviorType"/>
25     <ObjectProperty rdf:about="<BERDD-Ontology:hasComponentName"/>
26     <ObjectProperty rdf:about="<BERDD-Ontology:nextNode"/>
27     <!-- Individuals Behavior Names -->
28     <NamedIndividual rdf:about="<BERDD-Ontology:Activated">
29       <rdf:type rdf:resource="<BERDD-Ontology:BehaviorName"/></NamedIndividual>
30     <NamedIndividual rdf:about="<BERDD-Ontology:Pressed">
31       <rdf:type rdf:resource="<BERDD-Ontology:BehaviorName"/></NamedIndividual>
32     <NamedIndividual rdf:about="<BERDD-Ontology:Deactivated">
33       <rdf:type rdf:resource="<BERDD-Ontology:BehaviorName"/></NamedIndividual>
34     <!-- Individuals Behavior Types -->
35     <NamedIndividual rdf:about="<BERDD-Ontology:Event">
36       <rdf:type rdf:resource="<BERDD-Ontology:BehaviorType"/></NamedIndividual>
37     <NamedIndividual rdf:about="<BERDD-Ontology:StateRealization">
38       <rdf:type rdf:resource="<BERDD-Ontology:BehaviorType"/></NamedIndividual>
39     <!-- Individuals Component Names -->
40     <NamedIndividual rdf:about="<BERDD-Ontology:SET-BUTTON">
41       <rdf:type rdf:resource="<BERDD-Ontology:ComponentName"/></NamedIndividual>
42     <NamedIndividual rdf:about="<BERDD-Ontology:SAS">
43       <rdf:type rdf:resource="<BERDD-Ontology:ComponentName"/></NamedIndividual>
44     <!-- Individuals Nodes -->
45     <NamedIndividual rdf:about="<BERDD-Ontology:R1.1">
46       <rdf:type rdf:resource="<BERDD-Ontology:BTNodes"/>
47       <BERDD-Ontology:hasComponentName rdf:resource="<BERDD-Ontology:SAS"/>
48       <BERDD-Ontology:hasBehaviorName rdf:resource="<BERDD-Ontology:Deactivated"/>
49       <BERDD-Ontology:hasBehaviorType rdf:resource="<BERDD-Ontology:StateRealization"/>
50       <BERDD-Ontology:nextNode rdf:resource="<BERDD-Ontology:R1.2"/></NamedIndividual>
51     <NamedIndividual rdf:about="<BERDD-Ontology:R1.2">
52       <rdf:type rdf:resource="<BERDD-Ontology:BTNodes"/>
53       <BERDD-Ontology:hasComponentName rdf:resource="<BERDD-Ontology:SET-BUTTON"/>
54       <BERDD-Ontology:hasBehaviorName rdf:resource="<BERDD-Ontology:Pressed"/>
55       <BERDD-Ontology:hasBehaviorType rdf:resource="<BERDD-Ontology:Event"/>
56       <BERDD-Ontology:nextNode rdf:resource="<BERDD-Ontology:R1.3"/></NamedIndividual>
57     <NamedIndividual rdf:about="<BERDD-Ontology:R1.3">
58       <rdf:type rdf:resource="<BERDD-Ontology:BTNodes"/>
59       <BERDD-Ontology:hasComponentName rdf:resource="<BERDD-Ontology:SAS"/>
60       <BERDD-Ontology:hasBehaviorName rdf:resource="<BERDD-Ontology:Activated"/>
61       <BERDD-Ontology:hasBehaviorType rdf:resource="<BERDD-Ontology:StateRealization"/>
62       <BERDD-Ontology:nextNode rdf:resource="<BERDD-Ontology:"/></NamedIndividual>
63   </rdf:RDF>
```

B.2 SPARQL Queries

Ambiguity Query

Ambiguity Based on Behaviour Name

```
PREFIX owl: http://www.w3.org/2002/07/owl#
PREFIX xsd: http://www.w3.org/2001/XMLSchema#
PREFIX rdfs: http://www.w3.org/2000/01/rdf-schema#
SELECT DISTINCT ?node ?comname ?bname ?btype
WHERE {
  ?node rdf:type my:BTNodes.
  ?node my:hasComponentName ?comname.
  ?node my:hasBehaviorType ?btype.
  ?node my:hasBehaviorName ?bname
  FILTER REGEX(str( strafter(str(?bname), @prefix)), @parm1)}
```

Here the @parm1 is a behaviour name provided through tool interface.

Ambiguity Based on Component Name

```
PREFIX owl: http://www.w3.org/2002/07/owl#
PREFIX xsd: http://www.w3.org/2001/XMLSchema#
PREFIX rdfs: http://www.w3.org/2000/01/rdf-schema#
SELECT DISTINCT ?node ?comname ?bname ?btype
WHERE {
  ?node rdf:type my:BTNodes.
  ?node my:hasComponentName ?comname.
  ?node my:hasBehaviorType ?btype.
  ?node my:hasBehaviorName ?bname
  FILTER REGEX(str( strafter(str(?comname), @prefix)), @parm1)}
```

Here the @parm1 is a component name provided through tool interface.

Consistency and Redundancy Queries

```
PREFIX owl: http://www.w3.org/2002/07/owl#
PREFIX xsd: http://www.w3.org/2001/XMLSchema#
PREFIX rdfs: http://www.w3.org/2000/01/rdf-schema#
SELECT ?node1 ?node11 ?node2 ?node22 ?node222
where {
  ?node1 rdf:type my:BTNodes. ?node2 rdf:type my:BTNodes.
  ?node1 my:hasComponentName ?comname1. ?node1 my:hasBehaviorName ?bname1.
  ?node1 my:hasBehaviorType ?btype1. ?node2 my:hasComponentName ?comname2.
  ?node2 my:hasBehaviorName ?bname2. ?node2 my:hasBehaviorType ?btype2.
  ?node1 my:nextNode ?node11. ?node2 my:nextNode ?node22.

  ?node11 my:hasComponentName ?comname11. ?node11 my:hasBehaviorName ?bname11.
  ?node11 my:hasBehaviorType ?btype11. ?node22 my:hasComponentName ?comname22.
  ?node22 my:hasBehaviorName ?bname22. ?node22 my:hasBehaviorType ?btype22.

  FILTER (str( strafter(str(?comname1), @prefix)) = (str( strafter(str(?comname2), @prefix))))
  FILTER (str( strafter(str(?bname1), @prefix)) = (str( strafter(str(?bname2), @prefix))))
  FILTER (str( strafter(str(?btype1), @prefix)) = (str( strafter(str(?btype2), @prefix))))
  FILTER(?node1!=?node2)

  FILTER (str( strafter(str(?comname11), @prefix)) = (str( strafter(str(?comname22), @prefix))))
  FILTER (str( strafter(str(?bname11), @prefix)) = (str( strafter(str(?bname22), @prefix))))
  FILTER (str( strafter(str(?btype11), @prefix)) = (str( strafter(str(?btype22), @prefix))))

  FILTER (str( strafter(str(?comname111), @prefix)) = (str( strafter(str(?comname222), @prefix))))
  FILTER (str( strafter(str(?bname111), @prefix)) = (str( strafter(str(?bname222), @prefix))))
  FILTER (str( strafter(str(?btype111), @prefix)) = (str( strafter(str(?btype222), @prefix))))
} order by ?node1 ?node2 ?node11 ?node22 ?node111 ?node222";
```

This query can be used for both redundancy and consistency defects because both work is a same way except the last step. Here, we find two consecutive same nodes, and then we will run the similar query to get the next same nodes. Lastly, we will process the end result to obtain defects information on application side.

B.3 AIP IBT File



Publications and Submitted Papers

Journals:

1. S. Anwer, L. Wen, Z. Wang and S. Mahmood. Comparative Analysis of Requirement Change Management Challenges Between in-House and Global Software Development: Findings of Literature and Industry Survey, in IEEE Access, vol. 7, pp. 116585-116611, 2019.
2. S. Anwer, L. Wen and Z. Wang. BECIA: A Behaviour Engineering based Approach for Change Impact Analysis, (Submitted in Journal of Software: Evolution and Process)
3. S. Zhang, L. Wen, S. Anwer, B. Liu. Using Composition Trees to Matching Software Requirements – an External Agency’s Approach to Support Software Acquisition. (Revision Submitted) in Journal of Software: Practices and Experience
4. S. Anwer, L. Wen and Z. Wang, ”BERDD: A Behaviour Engineering based Approach for Requirements Defects Detection” (In preparation).

Conferences:

1. S. Anwer, L. Wen, T. Rout, and Z. Wang, Introducing Requirements Change Management Process into ISO/IEC 12207. In: Software Process Improvement and Capability Determination. SPICE 2018. Springer, Cham.
2. S. Anwer, L. Wen, and Z. Wang. A Systematic Approach for Identifying Requirement Change Management Challenges: Preliminary Results. In Proceedings of the Evaluation and Assessment on Software Engineering (EASE ’19), 230–235.
3. S. Anwer, L. Wen and Z. Wang, ”A Formal Model for Behavior Trees Based on Context-Free Grammar,” 27th Asia-Pacific Software Engineering Conference (APSEC), Singapore, 2020, pp. 465-469.

Bibliography

- [1] K. Wiegers and J. Beatty. *Software requirements*. Pearson Education, 2013.
- [2] J. Binder. *Global project management: communication, collaboration and management across borders*. Routledge, 2016.
- [3] L. J. Osterweil. A future for software engineering? In *Future of Software Engineering (FOSE'07)*, pages 1–11, 2007.
- [4] G. Booch. The history of software engineering. *IEEE Software*, 35(5):108–114, 2018.
- [5] I. Sommerville. *Software engineering 9th Edition*. Addison-Wesley Publishing Company, 2011. ISBN 10137035152.
- [6] R. S. Pressman and B. R. Maxim. *Software engineering: a practitioner's approach*. McGraw-Hill Education, 2015.
- [7] T. Hall, S. Beecham, and A. Rainer. Requirements problems in twelve software companies: an empirical analysis. *IEE Proceedings-Software*, 149:153–160, 2002.
- [8] C. Jones. Software project management practices: Failure versus success. *CrossTalk: The Journal of Defense Software Engineering*, 17(10):5–9, 2004.
- [9] B. Berenbach, D. Paulish, J. Kazmeier, and A. Rudorfer. *Software & systems requirements engineering: in practice*. McGraw-Hill, Inc., 2009.
- [10] V. S. Kumar. Effective requirements management. In *PMI® Global Congress —EMEA, Madrid, Spain.*, 2006.
- [11] E. J. Barry, T. Mukhopadhyay, and S. A. Slaughter. Software project duration and effort: an empirical study. *Information Technology and Management*, 3:113–136, 2002.

- [12] N. Nurmuliani, D. Zowghi, and S. Powell. Analysis of requirements volatility during software development life cycle. *Australian Software Engineering Conference. Proceedings.*, pages 28–37, 2004.
- [13] J. Kasser. *Object-oriented requirements engineering and management*. PhD thesis, Systems Engineering Society of Australia, 2003.
- [14] M. Bano, S. Imtiaz, N. Ikram, M. Niazi, and M. Usman. Causes of requirement change - a systematic literature review. In *16th International Conference on Evaluation & Assessment in Software Engineering*, 2012.
- [15] G. Arias, D. Vilches, C. Banchoff, I. Harari, V. Harari, and P. Iuliano. The 7 key factors to get successful results in the IT development projects. *Procedia technology*, 5:199–207, 2012.
- [16] *Standish-Group International Chaos - the state of the software industry.*, 2019.
- [17] J. T. Marchewka. The FBI virtual case file: A case study. *Communications of the IIMA*, 10, 2010.
- [18] L. Zhang, J. Tian, J. Jiang, Y. Liu, M. Pu, and T. Yue. Empirical research in software engineering—a literature survey. *Journal of Computer Science and Technology*, 33(5):876–899, 2018.
- [19] C. Wohlin, M. Höst, and K. Henningsson. Empirical research methods in software engineering. In *Empirical methods and studies in software engineering*, pages 7–23. Springer, 2003.
- [20] S. Jayatilleke and R. Lai. A systematic review of requirements change management. *Information and Software Technology*, 93:163–185, 2018.
- [21] M. A. Akbar, A. A. Khan, A. W. Khan, and S. Mahmood. Requirement change management challenges in GSD: an analytical hierarchy process approach. *Journal of Software: Evolution and Process*, 32(7), 2020.
- [22] A. A. Khan and M. A. Akbar. Systematic literature review and empirical investigation of motivators for requirements change management process in global software development. *Journal of Software: Evolution and Process*, 32(4), 2020.

- [23] S. Ahmad. A systematic literature review on requirement change management challenges faced by developers. *International Journal of Emerging Technologies in Engineering Research*, 2020.
- [24] M. El Bajta, A. Idri, J. N. Ros, J. L. Fernández-Alemán, J. M. C. de Gea, F. García, and A. Toval. Software project management approaches for global software development: a systematic mapping study. *Tsinghua Science and Technology*, 23:690–714, 2018.
- [25] A. Calderón, M. Ruiz, and R. V. O’Connor. A serious game to support the iso 21500 standard education in the context of software project management. *Computer Standards & Interfaces*, 60:80–92, 2018.
- [26] A. I. Wasserman. Toward a discipline of software engineering. *IEEE software*, 13: 23–31, 1996.
- [27] L. J. Osterweil. Formalisms to support the definition of processes. *Journal of Computer Science and Technology*, 24:198–211, 2009.
- [28] ISO/IEC 12207: 2017, - information technology - system engineering- software life cycle process. Technical report, International Organization for Standardization, Geneva, Switzerland, 2017.
- [29] ISO/IEC 15288: 2015, - information technology - system engineering- system life cycle process. Technical report, International Organization for Standardization, Geneva, Switzerland, 2015.
- [30] S. McGee and D. Greer. Sources of software requirements change from the perspectives of development and maintenance. *International Journal on Advances in Software*, 2010.
- [31] S. A. Bohner. Software change impacts-an evolving perspective. In *International Conference on Software Maintenance*, pages 263–272, 2002.
- [32] L. G. Yu and S. Ramaswamy. Component dependency in object-oriented software. *Journal of Computer Science and Technology*, 22:379–386, 2007.
- [33] A. AlSanad and A. Chikh. Software requirements change management—a comprehensive model. In *World Conference on Information Systems and Technologies*, pages 821–830, 2017.

- [34] A. Goknil, I. Kurtev, and K. Berg. A rule-based change impact analysis approach in software architecture for requirements changes. *arXiv preprint arXiv:1608.02757*, 2016.
- [35] M. Mondal, B. Roy, C. K. Roy, and K. A. Schneider. Associating code clones with association rules for change impact analysis. In *IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 93–103, 2020.
- [36] F. Angerer, A. Grimmer, H. Prähofer, and P. Grünbacher. Change impact analysis for maintenance and evolution of variable software systems. *Automated Software Engineering*, 26:417–461, 2019.
- [37] S. Moiseyenko and V. Ermolayev. Conceptualizing and formalizing requirements for ontology engineering. In *PhD@ ICTERI*, pages 35–44, 2018.
- [38] J. Chanda, A. Kanjilal, and S. Sengupta. UML-compiler: a framework for syntactic and semantic verification of UML diagrams. In *International Conference on Distributed Computing and Internet Technology*, pages 194–205, 2010.
- [39] F. Shull, V. Basili, B. Boehm, A. W. Brown, P. Costa, M. Lindvall, D. Port, I. Rus, R. Tesoriero, and M. Zelkowitz. What we have learned about fighting defects. In *Proceedings eighth IEEE symposium on software metrics*, pages 249–258, 2002.
- [40] B. Boehm and V. R. Basili. Software defect reduction top 10 list. *Foundations of empirical software engineering*, 426:426–431, 2005.
- [41] R. J. Kosman. A two-step methodology to reduce requirement defects. *Annals of Software Engineering*, 3:477–494, 1997.
- [42] N. Moha, Y. Guéhéneuc, and P. Leduc. Automatic generation of detection algorithms for design defects. In *21st IEEE/ACM International Conference on Automated Software Engineering*, pages 297–300, 2006.
- [43] V. Gervasi and D. Zowghi. Reasoning about inconsistencies in natural language requirements. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 14:277–330, 2005.
- [44] M. J. Cresswell. *Logics and languages*. Routledge, 2016.

- [45] M. Gogolla. Benefits and problems of formal methods. In *International Conference on Reliable Software Technologies*, pages 1–15, 2004.
- [46] R. V. O’Connor. Developing software and systems engineering standards. In *Proceedings of the 16th International Conference on Computer Systems and Technologies*, pages 13–21, 2015.
- [47] L. Wen, D. Tuffley, and T. Rout. Using composition trees to model and compare software process. In *International Conference on Software Process Improvement and Capability Determination*, pages 1–15, 2011.
- [48] A. Finkelstein and J. Dowell. A comedy of errors: the london ambulance service case study. In *Proceedings of the 8th International Workshop on Software Specification and Design*, pages 2–4, 1996.
- [49] R. Eden and D. Sedera. The largest admitted IT project failure in the southern hemisphere: a teaching case. In *Proceedings of the 35th International Conference on Information Systems*, pages 1–15, 2014.
- [50] C. Pacheco, I. García, and M. Reyes. Requirements elicitation techniques: a systematic literature review based on the maturity of the techniques. *IET Software*, 12:365–378, 2018.
- [51] M. Niazi, D. Wilson, and D. Zowghi. Critical success factors for software process improvement implementation: an empirical study. *Software Process: Improvement and Practice*, 11:193–211, 2006.
- [52] *Standish-Group International Chaos - the state of the software industry.*, 2017.
- [53] M. Krigsman. Shared services in the department for transport and its agencies, 2008.
- [54] B. Kitchenham. Evidence-based software engineering and systematic literature reviews. In *Product-Focused Software Process Improvement*, 2006.
- [55] E. Schön, J. Thomaschewski, and M. J. Escalona. Agile requirements engineering: A systematic literature review. *Computer Standards & Interfaces*, 49:79–91, 2017.
- [56] G. K. Hanssen, D. Šmite, and N. B. Moe. Signs of agile trends in global software engineering research: A tertiary study. In *IEEE Sixth International Conference on Global Software Engineering Workshop*, pages 17–23, 2011.

- [57] D. Albuquerque, E. Guimaraes, M. Perkusich, A. Costa, E. Dantas, F. Ramos, and H. Almeida. Defining agile requirements change management: A mapping study. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, page 1421–1424, 2020.
- [58] K. Batool and I. Inayat. An empirical investigation on requirements change management practices in pakistani agile based industry. In *International Conference on Frontiers of Information Technology*, pages 7–75, 2019.
- [59] S. McGee and D. Greer. Towards an understanding of the causes and effects of software requirements change: two case studies. *Requirements Engineering*, 17: 133–155, 2012.
- [60] M. A. Akbar, W. Naveed, A. A. Alsanad, L. Alsuwaidan, A. Alsanad, A. Gumaei, M. Shafiq, and M. T. Riaz. Requirements change management challenges of global software development: An empirical investigation. *IEEE Access*, 8:203070–203085, 2020.
- [61] A. A. Khan, S. Basri, and P. Dominic. Communication risks in GSD during RCM: Results from SLR. In *International Conference on Computer and Information Sciences (ICCOINS)*, pages 1–6, 2014.
- [62] A. A. Khan, J. Keung, S. Hussain, and K. E. Bennin. Effects of geographical, socio-cultural and temporal distances on communication in global software development during requirements change management a pilot study. In *International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, pages 159–168, 2015.
- [63] M. Shafiq, Q. Zhang, M. A. Akbar, A. A. Khan, S. Hussain, F. Amin, A. Khan, and A. A. Soofi. Effect of project management in requirements engineering and requirements change management processes for global software development. *IEEE Access*, 6:25747–25763, 2018.
- [64] ISO/IEC 29110: 2011, - software engineering- lifecycle profiles for very small entities (vses). Technical report, International Organization for Standardization, Geneva, Switzerland, 2011.

- [65] S. Lehnert. A taxonomy for software change impact analysis. In *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution*, pages 41–50, 2011.
- [66] J. W. Wilkerson. A software change impact analysis taxonomy. In *28th IEEE International Conference on Software Maintenance (ICSM)*, pages 625–628, 2012.
- [67] W. Spijkerman. Tool support for change impact analysis in requirement models: exploiting semantics of requirement relations as traceability relations. Master’s thesis, University of Twente, 2010.
- [68] A. McNair, D. M. German, and J. Weber-Jahnke. Visualizing software architecture evolution using change-sets. In *14th Working Conference on Reverse Engineering (WCRE 2007)*, pages 130–139, 2007.
- [69] M. A. Hoffman. Automated impact analysis of object-oriented software systems. In *18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, page 72–73, 2003.
- [70] M. N. Sudin and S. Ahmed-Kristensen. Change in requirements during the design process. In *Proceedings of the 18th International Conference on Engineering Design*, pages 200–208, 2011.
- [71] M. Hammad, M. L. Collard, and J. I. Maletic. Automatically identifying changes that impact code-to-design traceability during evolution. *Software Quality Journal*, 19:35–64, 2011.
- [72] N. Ali and R. Lai. A method of requirements change management for global software development. *Information and Software Technology*, 70:49–67, 2016.
- [73] P. Henderson. Why large it projects fail. *ACM Trans. Program. Lang. Syst*, 15: 795–825, 2006.
- [74] S. Lauesen and O. Vinter. Preventing requirement defects: An experiment in process improvement. *Requirements Engineering*, 6:37–50, 2001.
- [75] E. Serna and A. Serna. Process and progress of requirement formalization in software engineering. *INGENIARE-Revista Chilena de Ingeniería*, 28, 2020.

- [76] S. Gulan, S. Johr, R. Kretschmer, S. Rieger, and M. Ditze. Graphical modelling meets formal methods. In *11th IEEE International Conference on Industrial Informatics (INDIN)*, pages 716–721, 2013.
- [77] A. H. Khan and I. Porres. Consistency of UML class, object and statechart diagrams using ontology reasoners. *Journal of Visual Languages & Computing*, 26:42–65, 2015.
- [78] K. Kaneiwa and K. Satoh. On the complexities of consistency checking for restricted UML class diagrams. *Theoretical Computer Science*, 411:301–323, 2010.
- [79] T. Mens, R. Van Der Straeten, and J. Simmonds. Maintaining consistency between UML models with description logic tools. In *ECOOOP workshop on object-oriented reengineering*, volume 3031, 2003.
- [80] S. Zafar, M. Ahmed, T. Fatima, and Z. Aslam. SimTee: an automated environment for simulation and analysis of requirements. In *Future of Information and Communication Conference*, pages 341–356, 2019.
- [81] X. Chen, Z. Zhong, Z. Jin, M. Zhang, T. Li, X. Chen, and T. Zhou. Automating consistency verification of safety requirements for railway interlocking systems. In *27th International Requirements Engineering Conference*, pages 308–318, 2019.
- [82] C. Nentwich, W. Emmerich, A. Finkelstein, and E. Ellmer. Flexible consistency checking. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12:28–63, 2003.
- [83] F. Weitzl, M. Jakšić, and B. Freitag. Towards the automated verification of semi-structured documents. *Data & Knowledge Engineering*, 68:292–317, 2009.
- [84] A. Ferrari, G. Gori, B. Rosadini, I. Trotta, S. Bacherini, A. Fantechi, and S. Gnesi. Detecting requirements defects with NLP patterns: an industrial experience in the railway domain. *Empirical Software Engineering*, 23(6):3684–3733, 2018.
- [85] T. C. Lethbridge, S. E. Sim, and J. Singer. Studying software engineers: Data collection techniques for software field studies. *Empirical software engineering*, 10: 311–341, 2005.

- [86] P. Clarke and R. V. O'Connor. The situational factors that affect the software development process: Towards a comprehensive reference framework. *Information and Software Technology*, 54:433–447, 2012.
- [87] A. Fuggetta and E. Di Nitto. Software process. In *Future of Software Engineering Proceedings*, pages 1–12, 2014.
- [88] S. T. Acuna, N. Juristo, A. M. Moreno, and A. Mon. *A Software Process Model Handbook for Incorporating People's Capabilities*. Springer Science & Business Media, 2006.
- [89] D. Jain. Importance of processes and standards in software development. the code project. *Retrieved June*, 17, 2007.
- [90] ISO/IEC 12207: 2008, - information technology - system engineering- system life cycle process.”. Technical report, International Organization for Standardization, Geneva, Switzerland, 2008.
- [91] L. Wen and R. G. Dromey. From requirements change to design change: A formal path. In *Proceedings of the Second International Conference on Software Engineering and Formal Methods*, pages 104–113, 2004.
- [92] R. G. Dromey. Climbing over the” no silver bullet” brick wall. *IEEE Software*, 23: 120–119, 2006.
- [93] R. G. Dromey. Genetic software engineering-simplifying design using requirements integration. In *IEEE Working Conference on Complex and Dynamic Systems Architecture*, pages 251–257, 2001.
- [94] R. G. Dromey. System composition: constructive support for the analysis and design of large systems. In *Systems Engineering/Test and Evaluation Conference, Brisbane, Australia*, 2005.
- [95] R. L. Glass. Is this a revolutionary idea, or not? *Communications of the ACM*, 47: 23–25, 2004.
- [96] L. Wen. Behavior engineering world (resources). 2012. <http://www.beworld.org/BE/home/be-resources/>. Accessed: 06-12-2021.

- [97] T. Myers. *The Foundation for a Scaleable Methodology for Systems Design*. PhD thesis, Griffith University, 2010.
- [98] C. Gonzalez-Perez, B. Henderson-Sellers, and G. Dromey. A metamodel for the behavior trees modelling technique. In *Third International Conference on Information Technology and Applications*, volume 1, pages 35–39, 2005.
- [99] R. J. Colvin and I. J. Hayes. A semantics for behavior trees using CSP with specification commands. *Science of Computer Programming*, 76:891–914, 2011.
- [100] L. Grunske, P. Lindsay, N. Yatapanage, and K. Winter. An automated failure mode and effect analysis based on high-level design specification with behavior trees. In *International Conference on Integrated Formal Methods*, pages 129–149, 2005.
- [101] L. Grunske, K. Winter, and R. Colvin. Timed behavior trees and their application to verifying real-time systems. In *Australian Software Engineering Conference (ASWEC’07)*, pages 211–222, 2007.
- [102] R. Colvin and I. J Hayes. A semantics for behavior trees. Technical report, University of Queensland, 2007.
- [103] L. W. Chan, R. Hexel, and L. Wen. Integrating non-monotonic reasoning into high level component-based modelling using behavior trees. In *SoMeT*, pages 21–40, 2012.
- [104] K. Ahmed, L. Wen, and A. Sattar. iRE: a semantic network based interactive requirements engineering framework. In *Second World Conference on Complex Systems (WCCS)*, pages 171–177, 2014.
- [105] R. Colvin, L. Grunske, and K. Winter. Probabilistic timed behavior trees. In *International Conference on Integrated Formal Methods*, pages 156–175, 2007.
- [106] S. Shlaer and S. J. Mellor. *Object lifecycles: modeling the world in states*. Yourdon Press, 1992.
- [107] G. Jäger and J. Rogers. Formal language theory: refining the chomsky hierarchy. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 367:1956–1970, 2012.

- [108] J. A. Reggia and G. S. Wilkinson. A framework for the comparative study of language. In *Biolinguistic Investigations and the Formal Language Hierarchy*, pages 179–198. Routledge, 2018.
- [109] R. C. Metzger. 14 - the way of the computer scientist. In *Debugging by Thinking*, pages 473 – 507. Digital Press, 2004.
- [110] I. Horrocks and P. F. Patel-Schneider. Knowledge representation and reasoning on the semantic web: Owl. *Handbook of Semantic Web Technologies*, pages 365–398, 2011.
- [111] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph. OWL 2 web ontology language primer. *W3C recommendation*, 27:123, 2009.
- [112] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *journal of web semantics*, 2007.
- [113] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, and Z. Wang. HermiT: an OWL 2 reasoner. *Journal of Automated Reasoning*, 53:245–269, 2014.
- [114] W3C OWL Working Group. OWL 2 web ontology language document overview. Technical report, Word Wide Web Consortium, 2012.
- [115] I. Horrocks, P. F. Patel-Schneider, and F. Van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of web semantics*, 1:7–26, 2003.
- [116] L. Giordano, V. Gliozzi, N. Olivetti, and G. L. Pozzato. Semantic characterization of rational closure: From propositional logic to description logics. *Artificial Intelligence*, 226:1–33, 2015.
- [117] O. Corcho and A. Gómez-Pérez. A roadmap to ontology specification languages. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 80–96, 2000.
- [118] D. Tsarkov and I. Horrocks. FaCT++ description logic reasoner: System description. In *International joint conference on automated reasoning*, pages 292–297, 2006.
- [119] D. Dermeval, J. Vilela, I. Bittencourt, J. Castro, S. Isotani, P. Brito, and A. Silva. Applications of ontologies in requirements engineering: a systematic review of the literature. *Requirements Engineering*, 21(4):405–437, 2016.

- [120] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems (TODS)*, 34:1–45, 2009.
- [121] A. Seaborne, G. Manjunath, C. Bizer, J. Breslin, S. Das, I. Davis, S. Harris, K. Idehen, O. Corby, and K Kjernsmo. SPARQL/update: A language for updating RDF graphs. *W3c member submission*, 15, 2008.
- [122] P. Lago, H. Muccini, and M. A. Babar. Developing a course on designing software in globally distributed teams. In *IEEE International Conference on Global Software Engineering*, pages 249–253, 2008.
- [123] T. Kern and L. Willcocks. Exploring information technology outsourcing relationships: theory and practice. *The Journal of Strategic Information Systems*, 9:321–350, 2000.
- [124] K. Mohan, P. Xu, L. Cao, and B. Ramesh. Improving change management in software development: Integrating traceability and software configuration management. *Decision Support Systems*, 45:922–936, 2008.
- [125] 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 *24th International Requirements Engineering Conference*, pages 126–135, 2016.
- [126] M. Niazi, C. Hickman, R. Ahmad, and M. A. Babar. A model for requirements change management: Implementation of CMMI level 2 specific practice. In *International Conference on Product Focused Software Process Improvement*, pages 143–157, 2008.
- [127] M. A. Akbar, J. Sang, A. A. Khan, S. Mahmood, S. F. Qadri, H. Hu, and H. Xi-ang. Success factors influencing requirements change management process in global software development. *Journal of Computer Languages*, 51:112–130, 2019.
- [128] J. M. Verner and W. M. Evanco. In-house software development: what project management practices lead to success? *IEEE software*, 22:86–93, 2005.
- [129] E. Carmel, J. A. Espinosa, and Y. Dubinsky. ” follow the sun” workflow in global software development. *Journal of Management Information Systems*, 27:17–38, 2010.

- [130] S. Ambler. IT project success rates survey results. Technical report, Ambysoft, 2018.
- [131] D. Bradstreet. Dun & bradstreet’s barometer of global outsourcing. *Dun & Bradstreet*, 20, 2000.
- [132] L. McLaughlin. An eye on india: outsourcing debate continues. *IEEE Software*, 20: 114–117, 2003.
- [133] S. Deshpande, S. Beecham, and I. Richardson. Using the PMBOK guide to frame GSD coordination strategies. In *8th International Conference on Global Software Engineering*, pages 188–196, 2013.
- [134] M. Niazi, S. Mahmood, M. Alshayeb, M. R. Riaz, K. Faisal, N. Cerpa, S. U. Khan, and I. Richardson. Challenges of project management in global software development: A client-vendor analysis. *Information and Software Technology*, 80:1–19, 2016.
- [135] M. Mäkräinen. Software change management processes in the development of embedded software. *VTT Publications*, 2000.
- [136] J. Ren, X. Zhang, and Z. Pan. Analysis of software requirements change priorities based on complex networks. In *6th International Conference on Dependable Systems and Their Applications (DSA)*, pages 255–261, 2020.
- [137] A. A. Alsanad, A. Chikh, and A. Mirza. A domain ontology for software requirements change management in global software development environment. *IEEE Access*, 7:49352–49361, 2019.
- [138] B. B. Chua and J. Verner. Examining requirements change rework effort: A study. *arXiv preprint arXiv:1007.5126*, 2010.
- [139] S. Jayatilleke and R. Lai. A method of assessing rework for implementing software requirements changes. *Computer Science and Information Systems*, pages 32–32, 2020.
- [140] M. A. Akbar, S. Mahmood, Z. Huang, A. A. Khan, and M. Shameem. Readiness model for requirements change management in global software development. *Journal of Software: Evolution and Process*, 32(10), apr 2020.

- [141] H. Ahmed, A. Hussain, and F. Baharom. Current challenges of requirement change management. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, 8(10):173–176, 2016.
- [142] M. A. Akbar, M. Shameem, J. Ahmad, A. Maqbool, and K. Abbas. Investigation of project administration related challenging factors of requirements change management in global software development: A systematic literature review. In *International Conference on Computing, Electronic and Electrical Engineering (ICE Cube)*, pages 1–7, 2018.
- [143] B. Kitchenham and S. Charters. Guidelines for performing systematic literature reviews in software engineering. Technical report, Keele University and Durham University, 2007.
- [144] D. Budgen, M. Turner, P. Brereton, and B. A. Kitchenham. Using mapping studies in software engineering. In *PPIG*, volume 8, pages 195–204, 2008.
- [145] S. Mahdavi-Hezavehi, M. Galster, and P. Avgeriou. Variability in quality attributes of service-based software systems: A systematic literature review. *Information and Software Technology*, 55:320–343, 2013.
- [146] T. Dybå and T. Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and software technology*, 50:833–859, 2008.
- [147] P. Achimugu, A. Selamat, R. Ibrahim, and M. N. Mahrin. A systematic literature review of software requirements prioritization research. *Information and software technology*, 56:568–585, 2014.
- [148] W. Ding, P. Liang, A. Tang, and H. Van Vliet. Knowledge-based approaches in software documentation: A systematic literature review. *Information and Software Technology*, 56:545–567, 2014.
- [149] J. M. Corbin and A. Strauss. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative sociology*, 13:3–21, 1990.
- [150] I. Walsh. Using quantitative data in mixed-design grounded theory studies: an enhanced path to formal grounded theory in information systems. *European Journal of Information Systems*, 24(5):531–557, 2015.

- [151] A. Von Eye and E. Y. Mun. *Analyzing rater agreement: Manifest variable methods*. Psychology Press, 2014.
- [152] M. Niazi, M. A. Babar, and J. M. Verner. Software process improvement barriers: A cross-cultural comparison. *Information and software technology*, 52:1204–1216, 2010.
- [153] M. Spreen. Rare populations, hidden populations, and link-tracing designs: What and why? *Bulletin of Sociological Methodology/Bulletin de Methodologie Sociologique*, 36:34–58, 1992.
- [154] D. A. Dillman, J. D. Smyth, and L. M. Christian. *Internet, phone, mail, and mixed-mode surveys: the tailored design method*. John Wiley & Sons, 2014.
- [155] S. L. Pfleeger and B. A. Kitchenham. Principles of survey research: part 1: turning lemons into lemonade. *ACM SIGSOFT Software Engineering Notes*, 26:16–18, 2001.
- [156] M. A. Revilla, W. E. Saris, and J. A. Krosnick. Choosing the number of categories in agree–disagree scales. *Sociological Methods & Research*, 43:73–97, 2013.
- [157] S. Easterbrook, J. Singer, M. Storey, and D. Damian. Selecting empirical methods for software engineering research. In *Guide to advanced empirical software engineering*, pages 285–311. Springer, 2008.
- [158] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.
- [159] A. A. Khan, S. Basri, and PDD Dominc. A proposed framework for communication risks during rcm in gsd. *Procedia-Social and Behavioral Sciences*, 129:496–503, 2014.
- [160] S. Lock and G. Kotonya. An integrated framework for requirement change impact analysis. In *4th Australian Conference on Requirements Engineering*, 1999.
- [161] S. Ibrahim, N. B. Idris, M. Munro, and A. Deraman. A requirements traceability to support change impact analysis. *Asian Journal of Information Tech*, 4:345–355, 2005.
- [162] S. Rajper and Z. A. Shaikh. Software development cost estimation: a survey. *Indian Journal of Science and Technology*, 9(31), 2016.

- [163] R. Wang, P. Peng, L. Xu, X. Huang, and X. Qiao. A novel algorithm for software development cost estimation based on fuzzy rough set. *Journal of Engineering Science and Technology Review*, 9(4):217–223, 2016.
- [164] N. Fenton and J. Bieman. *Software metrics: a rigorous and practical approach*. CRC press, 2014.
- [165] M. Kamalrudin and S. Sidek. A review on software requirements validation and consistency management. *International Journal of Software Engineering and Its Applications*, 9:39–58, 2015.
- [166] O. Gotel and A. Finkelstein. Extended requirements traceability: results of an industrial case study. In *Proceedings of 3rd IEEE International Symposium on Requirements Engineering*, pages 169–178, 1997.
- [167] N. Almasri, L. Tahat, and B. Korel. Toward automatically quantifying the impact of a change in systems. *Software Quality Journal*, 25(3):601–640, 2017.
- [168] D. Zowghi and V. Gervasi. On the interplay between consistency, completeness, and correctness in requirements evolution. *Information and Software technology*, 45:993–1009, 2003.
- [169] S. Robertson and J. Robertson. *Mastering the requirements process: Getting requirements right*. Addison-wesley, 2012.
- [170] T. H. Nguyen, B. Q. Vo, M. Lumpe, and J. Grundy. KBRE: a framework for knowledge-based requirements engineering. *Software Quality Journal*, 22:87–119, 2014.
- [171] F. Kaymaz. Prioritisation and selection of the right business and IT requirements in the software engineering process. *Vorgehensmodelle*, 2013.
- [172] *Standish-Group International Chaos - the state of the software industry.*, 2014.
- [173] D. Zowghi, F. da Rimini, and M. Bano. Problems and challenges of user involvement in software development: an empirical study. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, page 9, 2015.

- [174] A. Begel and N. Nagappan. Global software development: Who does it? In *IEEE International Conference on Global Software Engineering*, pages 195–199, 2008.
- [175] C. K. West. Four common reasons why projects fail, 2010.
- [176] Q. Yang, S. Kherbachi, Y. S. Hong, and C. Shan. Identifying and managing coordination complexity in global product development project. *International Journal of Project Management*, 33:1464–1475, 2015.
- [177] J. Noll, S. Beecham, and I. Richardson. Global software development and collaboration: barriers and solutions. *ACM inroads*, 1:66–78, 2010.
- [178] C. B. Seaman. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on software engineering*, 25:557–572, 1999.
- [179] O. F. Althuwaynee, B. Pradhan, H. Park, and J. H. Lee. A novel ensemble decision tree-based CHi-squared automatic interaction detection (chaid) and multivariate logistic regression models in landslide susceptibility mapping. *Landslides*, 11:1063–1078, 2014.
- [180] M. Niazi, M. A. Babar, and S. Ibrahim. An empirical study identifying high perceived value practices of CMMI level 2. In *International Conference on Product Focused Software Process Improvement*, pages 427–441, 2008.
- [181] S. U. Khan, M. Niazi, and R. Ahmad. Factors influencing clients in the selection of offshore software outsourcing vendors: An exploratory study using a systematic literature review. *Journal of systems and software*, 84:686–699, 2011.
- [182] S. Mahmood, S. Anwer, M. Niazi, M. Alshayeb, and I. Richardson. Key factors that influence task allocation in global software development. *Information and Software Technology*, 91:102–122, 2017.
- [183] S. Ambler. IT project success rates survey results. Technical report, Ambysoft, 2014.
- [184] S. Ambler. The non-existent software crisis: Debunking the chaos report, online: <http://www.drdoobs.com/architecture-and-design/the-non-existent-software-crisis-debunki/240165910>,. Technical report, Ambysoft, 2014.

- [185] M. Niazi. Do systematic literature reviews outperform informal literature reviews in the software engineering domain? an initial case study. *Arabian Journal for Science and Engineering*, 40:845–855, 2015.
- [186] A. Solinski and K. Petersen. Prioritizing agile benefits and limitations in relation to practice usage. *Software Quality Journal*, 24:447–482, 2014.
- [187] P. Savolainen, J. J. Ahonen, and I. Richardson. Software development project success and failure from the supplier’s perspective: A systematic literature review. *International Journal of Project Management*, 30:458–469, 2012.
- [188] H. Leung and Z. Fan. Software cost estimation. In *Handbook of Software Engineering and Knowledge Engineering*, pages 307–324. World Scientific, 2002.
- [189] J. Cleland-Huang, C.K. Chang, and M. Christensen. Event-based traceability for managing evolutionary change. *IEEE Transactions on Software Engineering*, 29: 796–810, 2003.
- [190] S. Imtiaz, N. Ikram, and S. Imtiaz. A process model for managing requirement change. In *Proceedings of the Fourth IASTED International Conference on Advances in Computer Science and Technology*, 2008.
- [191] PMI. *Practice Standard for Project Configuration Management*. Project Management Institute, 2007.
- [192] 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 *Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 266–272, 1993.
- [193] "ISO/IEC TR 24774 (2007). - software and systems engineering – life cycle management – guidelines for process description. Technical report, International Organization for Standardization, Geneva, Switzerland, 2007.
- [194] D. C. Ince. *Introduction to software quality assurance and its implementation*. McGraw-Hill, Inc., 1995.
- [195] J. Tomyim and A. Pohthong. Requirements change management based on object-oriented software engineering with unified modeling language. In *7th International Conference on Software Engineering and Service Science*, pages 7–10, 2016.

- [196] K. El Emam, D. Holtje, and N. H. Madhavji. Causal analysis of the requirements change process for a large system. In *Proceedings International Conference on Software Maintenance*, pages 214–221, 1997.
- [197] M. W. Bhatti, F. Hayat, N. Ehsan, A. Ishaque, S. Ahmed, and E. Mirza. A methodology to manage the changing requirements of a software project. *International Conference on Computer Information Systems and Industrial Management Applications (CISIM)*, pages 319–322, 2010.
- [198] D. Zowghi, R. Offen, and N. Nurmuliani. Impact of requirements volatility on the software development lifecycle. In *16th IFIP World Computer Conference: Software Theory & Practice*, pages 19–27, 2000.
- [199] N. C. Olsen. The software rush hour (software engineering). *IEEE Software*, 10: 29–37, 1993.
- [200] D. Tuffley and T. Rout. Behavior engineering as process model verification tool. In *The proceedings of the 10th International SPICE conference*, 2010.
- [201] S. Anwer, L. Wen, and Z. Wang. A systematic approach for identifying requirement change management challenges: preliminary results. In *Proceedings of the Evaluation and Assessment on Software Engineering*, pages 230–235, 2019.
- [202] L. Erlikh. Leveraging legacy system dollars for e-business. *IT professional*, 2:17–23, 2000.
- [203] R. S. Arnold and S. A. Bohner. Impact analysis-towards a framework for comparison. In *Conference on Software Maintenance*, pages 292–301, 1993.
- [204] H. Zhang, J. Li, L. Zhu, R. Jeffery, Y. Liu, Q. Wang, and M. Li. Investigating dependencies in software requirements for change propagation analysis. *Information and Software Technology*, 56:40–53, 2014.
- [205] C. Arora, M. Sabetzadeh, A. Goknil, L. C. Briand, and F. Zimmer. Change impact analysis for natural language requirements: An NLP approach. In *23rd International Requirements Engineering Conference (RE)*, pages 6–15, 2015.
- [206] X. Sun, B. Li, H. Leung, B. Li, and J. Zhu. Static change impact analysis techniques: A comparative study. *Journal of Systems and Software*, 109:137–149, 2015.

- [207] N. Ajenka, A. Capiluppi, and S. Counsell. An empirical study on the interplay between semantic coupling and co-change of software classes. *Empirical Software Engineering*, 23(3):1791–1825, 2018.
- [208] M. O. Hassan, L. Deruelle, and H. Basson. A knowledge-based system for change impact analysis on software architecture. In *Fourth International Conference on Research Challenges in Information Science (RCIS)*, pages 545–556, 2010.
- [209] K. Rostami, R. Heinrich, A. Busch, and R. Reussner. Architecture-based change impact analysis in information systems and business processes. In *International Conference on Software Architecture (ICSA)*, pages 179–188, 2017.
- [210] S. Jayatilleke, R. Lai, and K. Reed. A method of requirements change analysis. *Requirements Engineering*, 23(4):493–508, 2018.
- [211] B. Li, X. Sun, H. Leung, and S. Zhang. A survey of code-based change impact analysis techniques. *Software Testing, Verification and Reliability*, 23:613–646, 2013.
- [212] F. Palomba, G. Bavota, M. Di Penta, F. Fasano, R. Oliveto, and A. De Lucia. On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation. *Empirical Software Engineering*, 23(3):1188–1221, 2018.
- [213] A. R. Yazdanshenas and L. Moonen. Fine-grained change impact analysis for component-based product families. In *28th IEEE International Conference on Software Maintenance (ICSM)*, pages 119–128, 2012.
- [214] M. Hammad. Design observer: A framework to monitor design evolution. In *Information Technology-New Generations*, pages 635–640, 2018.
- [215] N. Al-Saiyd and E. Zriqat. Analyzing the impact of requirement changing on software design. *European Journal of Scientific Research*, 136, 2015.
- [216] A. Goknil, I. Kurtev, K. Van Den Berg, and W. Spijkerman. Change impact analysis for requirements: A metamodeling approach. *Information and Software Technology*, 56:950–972, 2014.
- [217] M. Li and P. Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer, 2008.

- [218] S. Jayatilleke and R. Lai. A method of specifying and classifying requirements change. In *22nd Australian Software Engineering Conference*, 2013.
- [219] S. Jayatilleke, R. Lai, and K. Reed. Managing software requirements changes through change specification and classification. *Computer Science and Information Systems*, 15(2):321–346, 2018.
- [220] S. McGee and D. Greer. Software requirements change taxonomy: Evaluation by case study. In *IEEE 19th International Requirements Engineering Conference*, pages 25–34, 2011.
- [221] L. Lavazza and G. Robiolo. Using functional complexity measures in software development effort estimation. *International Journal on Advances in Software*, 5: 263–277, 2012.
- [222] A. Gates, V. Kreinovich, and L. Longpre. Kolmogorov complexity justifies software engineering heuristics. *Departmental Technical Reports (CS)*, 1998.
- [223] South facing alliance. <http://www.xaxn-tech.com>. Accessed: 28-06-2021.
- [224] M. Kretsou, E. Arvanitou, A. Ampatzoglou, I. Deligiannis, and V. Gerogiannis. Change impact analysis: A systematic mapping study. *Journal of Systems and Software*, 2020.
- [225] M. S. Kilpinen, C. Eckert, and P. Clarkson. The emergence of change at the interface of system and embedded software design. In *Conference on System Engineering Research*, 2007.
- [226] 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*, pages 100–111, 2008.
- [227] A. Goknil, I. Kurtev, K. Berg, and J. Veldhuis. Semantics of trace relations in requirements models for consistency checking and inferencing. *Software & Systems Modeling*, 10:31–54, 2009.
- [228] N. A. Ernst, A. Borgida, and I. Jureta. Finding incremental solutions for evolving requirements. In *IEEE 19th International Requirements Engineering Conference*, 2011.

- [229] A. M. Grubb, G. Song, and M. Chechik. Growingleaf: Supporting requirements evolution over time. In *Proceedings of the Ninth International i* Workshop co-located with 24th International Conference on Requirements Engineering*, volume 1674, pages 31–36, 2016.
- [230] H. Alkaf, J. Hassine, T. Binalialhag, and D. Amyot. An automated change impact analysis approach for user requirements notation models. *Journal of Systems and Software*, 157:110397, 2019.
- [231] W. Lee, W. Deng, J. Lee, and S. Lee. Change impact analysis with a goal-driven traceability-based approach. *Int. J. Intell. Syst.*, 25:878–908, 2010.
- [232] J. Hassine, J. Rilling, and J. Hewitt. Change impact analysis for requirement evolution using use case maps. In *Eighth International Workshop on Principles of Software Evolution (IWPSE'05)*, 2005.
- [233] R. Lai and N. Ali. A requirements management method for global software development. *AIS: Advances in Information Sciences*, 1:38–58, 2013.
- [234] L. C. Briand, Y. Labiche, and L. O’Sullivan. Impact analysis and change management of UML models. In *International Conference on Software Maintenance*, pages 256–265, 2003.
- [235] L.C. Briand, Y. Labiche, L. O’Sullivan, and M.M. Sówka. Automated impact analysis of UML models. *Journal of Systems and Software*, 79:339–352, 2006.
- [236] Z. Xing and E. Stroulia. UMLDiff: an algorithm for object-oriented design differencing. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, page 54–65, 2005.
- [237] D. Kchaou, N. Bouassida, and H. Ben-Abdallah. Uml models change impact analysis using a text similarity technique. *IET Software*, 2016.
- [238] T. Feng and J.I. Maletic. Applying dynamic change impact analysis in component-based architecture design. In *Seventh ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD'06)*, 2006.

- [239] N. Kama and F. Azli. A change impact analysis approach for the software development phase. In *19th Asia-Pacific Software Engineering Conference*, volume 1, pages 583–592, 2012.
- [240] M. Rahimi and J. Cleland-Huang. Evolving software trace links between requirements and source code. *Empirical Software Engineering*, 23(4):2198–2231, 2017.
- [241] M. Malhotra and J. Chhabra. Improved computation of change impact analysis in software using all applicable dependencies. In *Futuristic Trends in Network and Communication Technologies*, pages 367–381, 01 2019.
- [242] T. Lutellier, D. Chollak, J. Garcia, L. Tan, D. Rayside, N. Medvidovic, and R. Kroeger. Comparing software architecture recovery techniques using accurate dependencies. In *37th IEEE International Conference on Software Engineering*, 2015.
- [243] B. B. P. Cafeo, E. Cirilo, A. Garcia, F. Dantas, and J. Lee. Feature dependencies as change propagators: An exploratory study of software product lines. *Information and Software Technology*, 69:37–49, 2016.
- [244] P. Behnamghader, R. Alfayez, K. Srisopha, and B. Boehm. Towards better understanding of software quality evolution through commit-impact analysis. In *International Conference on Software Quality, Reliability and Security (QRS)*, 2017.
- [245] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In *35th International Conference on Software Engineering*, 2013.
- [246] S. N. Ahsan and F. Wotawa. Impact analysis of SCRs using single and multi-label machine learning classification. In *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, 2010.
- [247] A. Beszedes, T. Gergely, S. Farago, T. Gyimothy, and F. Fischer. The dynamic function coupling metric and its use in software evolution. In *11th European Conference on Software Maintenance and Reengineering*, 2007.
- [248] D. Poshyvanyk, A. Marcus, R. Ferenc, and T. Gyimóthy. Using information retrieval based coupling measures for impact analysis. *Empirical Software Engineering*, 14: 5–32, 2008.

- [249] J. Lalchandani and R. Mall. Integrated state-based dynamic slicing technique for UML models. *IET Software*, 4:55–78, 2010.
- [250] J. Lalchandani and R. Mall. A dynamic slicing technique for UML architectural models. *IEEE Transactions on Software Engineering*, 37:737–771, 2011.
- [251] G. Canfora and M. Di Penta. New frontiers of reverse engineering. In *FoSE 2007: Future of Software Engineering*, pages 326 – 341, 2007.
- [252] M. F. Bashir and M. A. Qadir. Traceability techniques: A critical study. In *IEEE International Multitopic Conference*, 2006.
- [253] T. Verhanneman, F. Piessens, B. Win, and W. Joosen. Requirements traceability to support evolution of access control. *ACM SIGSOFT Software Engineering Notes*, 30:1–7, 2005.
- [254] P. A. Laplante. *Requirements engineering for software and systems*. Auerbach Publications, 2017.
- [255] M. R. V. Chaudron, W. Heijstek, and A. Nugroho. How effective is UML modeling ? *Software & Systems Modeling*, 11(4):571–580, 2012.
- [256] R.G. Dromey and D. Powell. Early requirements defect detection. *TickIT Journal*, 4:3–13, 2005.
- [257] K. Ahmed, T. Myers, L. Wen, and A. Sattar. Detecting requirements defects utilizing a mathematical framework for behavior engineering. *arXiv preprint arXiv:1401.5198*, 2014.
- [258] K. Ahmed, M. H. Newton, L. Wen, and A. Sattar. Formalisation of the integration of behavior trees. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, pages 779–784, 2014.
- [259] L. Grunske, K. Winter, N. Yatapanage, S. Zafar, and P. A. Lindsay. Experience with fault injection experiments for FMEA. *Software: Practice and Experience*, 41: 1233–1258, 2011.
- [260] L. Grunske, K. Winter, and N. Yatapanage. Defining the abstract syntax of visual languages with advanced graph grammars—a case study based on behavior trees. *Journal of Visual Languages & Computing*, 19:343–379, 2008.

- [261] P. A. Lindsay, S. Kromodimoeljo, P. A. Strooper, and M. Almorsy. Automation of test case generation from behavior tree requirements models. In *24th Australasian Software Engineering Conference*, pages 118–127, 2015.
- [262] L. Wen and R. G. Dromey. A hierarchical architecture for modeling complex software intensive systems using behavior trees. In *Proceedings of the 9th Asia-Pacific Complex Systems Conference*, pages 292–299, 2009.
- [263] S. Zafar, N. Farooq-Khan, and M. Ahmed. Requirements simulation for early validation using behavior trees and datalog. *Information and Software Technology*, 61: 52–70, 2015.
- [264] G. Besova, D. Steenken, and H. Wehrheim. Grammar-based model transformations: Definition, execution, and quality properties. *Computer Languages, Systems & Structures*, 43:116–138, 2015.
- [265] C. D. N. Damasceno, M. R. Mousavi, and A. Simao. Learning from difference: an automated approach for learning family models from software product lines. In *Proceedings of the 23rd International Systems and Software Product Line Conference*, pages 52–63, 2019.
- [266] F. Javed, M. Mernik, B. R. Bryant, and J. Gray. A grammar-based approach to class diagram validation. In *Fourth International Workshop on Scenarios and State Machines: Models, Algorithms and Tools*, 2005.
- [267] I. Ruiz-Rube, T. Person, J. M. Dodero, J. M. Mota, and Javier M. Sánchez-Jara. Applying static code analysis for domain-specific languages. *Software and Systems Modeling*, 19:95–110, 2020.
- [268] A. Mandal, D. Mohan, R. Jetley, S. Nair, and M. D’Souza. A generic static analysis framework for domain-specific languages. In *23rd International Conference on Emerging Technologies and Factory Automation*, pages 27–34, 2018.
- [269] B. Bernárdez, M. Genero, A. Durán, and M. Toro. A controlled experiment for evaluating a metric-based reading technique for requirements inspection. In *10th International Symposium on Software Metrics*, pages 257–268, 2004.

- [270] S. Banerjee and A. Sarkar. Domain-specific requirements analysis framework: ontology-driven approach. *International Journal of Computers and Applications*, pages 1–25, 2019.
- [271] A. Goknil, I. Kurtev, K. van den Berg, and J. Veldhuis. Semantics of trace relations in requirements models for consistency checking and inferencing. *Software & Systems Modeling*, 10:31–54, 2011.
- [272] A. Reder and A. Egyed. Incremental consistency checking for complex design rules and larger model changes. In *International Conference on Model Driven Engineering Languages and Systems*, pages 202–218, 2012.
- [273] K. Mu, W. Liu, Z. Jin, R. Lu, A. Yue, and D. Bell. A merging-based approach to handling inconsistency in locally prioritized software requirements. In *International Conference on Knowledge Science, Engineering and Management*, pages 103–114, 2007.
- [274] J. H. Hayes. Building a requirement fault taxonomy: Experiences from a NASA verification and validation research project. In *14th International Symposium on Software Reliability Engineering*, pages 49–59, 2003.
- [275] V. Langenfeld, A. Post, and A. Podelski. Requirements defects over a project lifetime: an empirical analysis of defect data from a 5-year automotive project at bosch. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 145–160, 2016.
- [276] E. Hull, K. Jackson, and J. Dick. DOORS: a tool to manage requirements. In *Requirements engineering*, pages 187–204. Springer, 2002.
- [277] P. Zielczynski. *Requirements Management Using IBM Rational RequisitePro*. IBM Press/Pearson plc, 2008.
- [278] Y. Xie, T. Tang, T. Xu, and L. Zhao. Research on requirement management for complex systems. In *2nd International Conference on Computer Engineering and Technology*, volume 1, pages V1–113, 2010.
- [279] M. Geisser, T. Hildenbrand, and N. Riegel. Evaluating the applicability of requirements engineering tools for distributed software development. Technical report, Business School: Sonstige - Fakultät für Betriebswirtschaftslehre, 2007.

- [280] A. Artale and E. Franconi. A temporal description logic for reasoning about actions and plans. *Journal of Artificial Intelligence Research*, 9:463–506, 1998.
- [281] M. Kamalrudin, J. Hosking, and J. Grundy. MaramaAIC: tool support for consistency management and validation of requirements. *Automated software engineering*, 24(1):1–45, 2017.
- [282] S. Liu, J. Sun, Y. Liu, Y. Zhang, B. Wadhwa, J. S. Dong, and X. Wang. Automatic early defects detection in use case documents. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, pages 785–790, 2014.
- [283] P. Kroha, R. Janetzko, and J. E. Labra. Ontologies in checking for inconsistency of requirements specification. In *Third International Conference on Advances in Semantic Processing*, pages 32–37, 2009.
- [284] A. Mavin, P. Wilksinson, S. Gregory, and E. Uusitalo. Listens learned (8 lessons learned applying ears). In *24th International Requirements Engineering Conference*, pages 276–282, 2016.
- [285] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer. Automated checking of conformance to requirements templates using natural language processing. *IEEE Transactions on Software Engineering*, 41(10):944–968, 2015.
- [286] S. F. Tjong and D. M. Berry. The design of SREE—a prototype potential ambiguity finder for requirements specifications and lessons learned. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 80–95, 2013.
- [287] H. Hasso, M. Dembach, H. Geppert, and D. Toews. Detection of defective requirements using rule-based scripts. In *REFSQ Workshops*, 2019.
- [288] H. Femmer, D. M. Fernández, S. Wagner, and S. Eder. Rapid quality assurance with requirements smells. *Journal of Systems and Software*, 123:190–213, 2017.
- [289] B. Wei, J. Sun, and Y. Wang. A knowledge engineering approach to uml modeling (s). In *International Conference on Software Engineering & Knowledge Engineering*, pages 60–63, 2018.
- [290] M. Sadowska and Z. Huzar. Representation of UML class diagrams in OWL 2 on the background of domain ontologies. *e-Informatica Vol. XIII*, 2018.

- [291] K. Siegemund, E. J. Thomas, Y. Zhao, J. Pan, and U. Assmann. Towards ontology-driven requirements engineering. In *Workshop semantic web enabled software engineering at 10th international semantic web conference (ISWC), Bonn*, 2011.
- [292] K. Verma and A. Kass. Requirements analysis tool: A tool for automatically analyzing software requirements documents. In *The Semantic Web - ISWC*, pages 751–763, 2008.
- [293] J. Muskens, R. J. Bril, and M. R. V. Chaudron. Generalizing consistency checking between software views. In *5th Working IEEE/IFIP Conference on Software Architecture*, pages 169–180, 2005.
- [294] J. Holt. *UML for Systems Engineering: watching the wheels*, volume 4. IET, 2004.
- [295] A. Kossiakoff, S. M. Biemer, S. J. Seymour, and D. A. Flanagan. *Systems engineering: Principles and practices*. Wiley Online Library, 2003.
- [296] L. Zhao, W. Alhoshan, A. Ferrari, K. J. Letsholo, M. A. Ajagbe, E. Chioasca, Batista-Navarro, and T. Riza. Natural language processing (NLP) for requirements engineering: A systematic mapping study. *arXiv preprint arXiv:2004.01099*, 2020.
- [297] F. Dalpiaz, A. Ferrari, X. Franch, and C. Palomares. Natural language processing for requirements engineering: The best is yet to come. *IEEE software*, 35(5):115–119, 2018.
- [298] I. L. Margarido, J. P. Faria, R. M. Vidal, and M. Vieira. Classification of defect types in requirements specifications: Literature review, proposal and assessment. In *6th Iberian Conference on Information Systems and Technologies*, pages 1–6, 2011.
- [299] S. L. Pfleeger and J. M. Atlee. *Software engineering: theory and practice*. Pearson Education India, 1998.
- [300] H. Yang, A. De Roeck, V. Gervasi, A. Willis, and B. Nuseibeh. Analysing anaphoric ambiguity in natural language requirements. *Requirements engineering*, 16:163, 2011.
- [301] B. Nuseibeh. To be and not to be: on managing inconsistency in software development. In *Proceedings of the 8th International Workshop on Software Specification and Design*, 2000.

- [302] A. Goffi, A. Gorla, A. Mattavelli, and M. Pezzè. Intrinsic redundancy for reliability and beyond. In *Present and Ulterior Software Engineering*, pages 153–171. Springer, 2017.
- [303] A. Carzaniga, A. Gorla, A. Mattavelli, N. Perino, and M. Pezze. Automatic recovery from runtime failures. In *35th International Conference on Software Engineering (ICSE)*, pages 782–791, 2013.
- [304] A. Carzaniga, A. Goffi, A. Gorla, A. Mattavelli, and M. Pezzè. Cross-checking oracles from intrinsic software redundancy. In *Proceedings of the 36th International Conference on Software Engineering*, pages 931–942, 2014.
- [305] S. Sidiroglou-Douskos, E. Lahtinen, F. Long, and M. Rinard. Automatic error elimination by horizontal code transfer across multiple applications. *SIGPLAN Not.*, 50:43–54, 2015.
- [306] A. Carzaniga, A. Mattavelli, and M. Pezzè. Measuring software redundancy. In *37th IEEE International Conference on Software Engineering*, pages 156–166, 2015.
- [307] M. Van den Brand and J. F. Groote. Software engineering: Redundancy is key. *Science of Computer programming*, 97:75–81, 2015.
- [308] dotNetRDF. <https://www.dotnetrdf.org/>. Accessed: 25-12-2020.
- [309] S. Zafar and R. G. Dromey. Integrating safety and security requirements into design of an embedded system. In *12th Asia-Pacific Software Engineering Conference*, 2005.
- [310] M. El-Attar and J. Miller. Producing robust use case diagrams via reverse engineering of use case descriptions. *Software & Systems Modeling*, 7:67–83, 2008.
- [311] S. Kovacevic. UML and user interface modeling. In *International Conference on the Unified Modeling Language*, pages 253–266, 1998.