AN EMPIRICAL STUDY ABOUT POSITIVE IMPLICATIONS OF REQUIREMENTS VOLATILITY ON THE SOFTWARE ARCHITECTURE

By:

SUMAIRA ANWAR BAIG

NATIONAL UNIVERSITY OF MODERN LANGUAGES
ISLAMABAD

December, 2022

AN EMPIRICAL STUDY ABOUT POSITIVE IMPLICATIONS OF REQUIREMENTS VOLATILITY ON THE SOFTWARE ARCHITECTURE

By:

SUMAIRA ANWAR BAIG

BS in Computer Sciences, University of the Punjab, Lahore, 2013

A THESIS SUBMIT IN PARTIAL FULFILMENT OF
THE REQUIREMENT FOR THE DEGREE OF

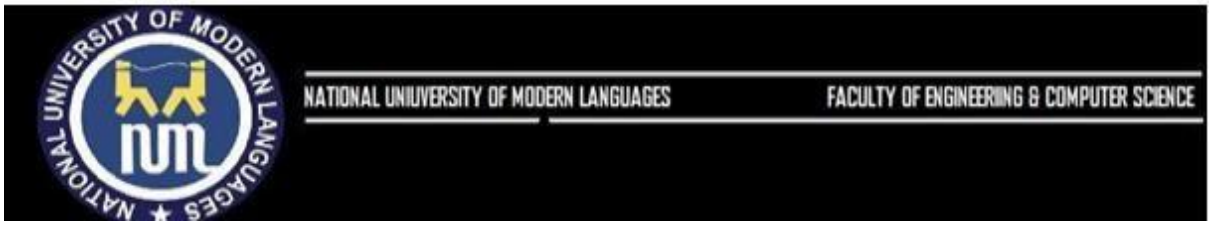
MASTER OF SCIENCE

In **Software Engineering**

To

FACULTY OF SOFTWARE ENGINEERING & COMPUTER SCIENCES

NATIONAL UNIVERSITY OF MODERN LANGUAGES ISLAMABAD



THESIS AND DEFENSE APPROVAL FORM

The undersigned certify that they have read the following thesis, examined the defense, are satisfied with overall exam performance, and recommend the thesis to the Faculty of Engineering and Computer Sciences for acceptance.

Thesis Title: An Empirical study about the positive implications of Requirements Volatility on the Software Architecture

Submitted by: Sumaira Anwar Baig **Registration #:** 18MSSE/Ibd/S19

Master of Science in Software Engineering
Degree Name in full

Software Engineering
Name of Discipline

Dr. Huma Hayat Khan
Name of Research Supervisor

Signature of Research Supervisor

Dr. Basit Shahzad
Name of Dean (FE&CS)

Signature of Dean (FE&CS)

Prof. Dr. Aamir Ijaz
Pro-Rector Academics

Signature of Pro-Rector Academics

7th December, 2022

Date

AUTHOR'S DECLARATION

I Sumaira Anwar Baig

Daughter of Muhammad Anwar Baig

Registration # 18MSSE/Ibd/S19

Discipline Software Engineering

Candidate of **Master of Science in Software Engineering (MSSE)** at the National University of Modern Languages do hereby declare that the thesis **An Empirical study about positive Implications of Requirements Volatility on the Software Architecture** submitted by me in partial fulfillment of MSSE degree, is my original work, and has not been submitted or published earlier. I also solemnly declare that it shall not, in the future, be submitted by me for obtaining any other degree from this or any other university or institution. I also understand that if evidence of plagiarism is found in my thesis/dissertation at any stage, even after the award of a degree, the work may be canceled and the degree revoked.

_____

Signature of Candidate

Sumaira Anwar Baig

Name of Candidate

7th December, 2022

Date

ABSTRACT

An Empirical study about the positive implications of Requirements Volatility on the Software Architecture

Requirement volatility is a fundamental activity that occurs throughout the software development life cycle. But, nowadays, it is becoming a striking reason for software project failures, such as software defects and resource management issues, especially in the context of the software architecture. A software architecture that indicates the complete vision of the upcoming system is one of the major areas that could be adversely affected by the requirements volatility. This phenomenon indicated the close connection and equal worth of both these twin peaks of the Software Development Life Cycle (SDLC) i.e. 'requirement volatility' and 'software architecture'. Moreover, modern software development models are fragile, wherein, the software architectures must be designed flexibly to accommodate future changes. However, the fragile nature of requirement volatility indicated their positive activity, and nor does it means an uncontrolled state of existence. Nevertheless, it is a challenging activity but it could be achieved through sound knowledge. For implementation, this study adopted a systematic literature review to identify the list of factors related to the software architecture which are also validated by the experts of the domain. In the end, an industrial survey was conducted to propose the positive implications of identified factors on software architecture. Accordingly, this study contributed a refined and validated list of 27 factors along with their positive implications. Moreover, this study revealed that communication issues and dependencies are the main factors that are causing requirement volatility and factors related to architecture i.e. traceability, design implementations, documentation, and architectural complexity having major implications on the software architecture. Accordingly, to better assist in the development process, the practitioners or developers must have to consider these factors to deal with the upcoming changes more, effectively.

TABLE OF CONTENTS

Chapter	TITLE	PAGE
	AUTHOR'S DECLARATION	iii
	ABSTRACT	iv
	LIST OF TABLES	ix
	LIST OF FIGURES	x
	LIST OF ABBREVIATIONS	xi
	LIST OF APPENDICES	xii
	ACKNOWLEDGEMENT	xiii
	DEDICATION	xiv
1	**INTRODUCTION**	**1**
	1.1 Overview	1
	1.2 Background of Research	1
	1.3 Research Problem	2
	1.4 Research Questions	2
	1.5 Research Objective	3
	1.6 Aim of the Research	3
	1.7 Scope of the Research	3
	1.8 Contribution of the Research	4
	1.9 Thesis Outline	4
	1.10 Summary of the Chapter	5
2	**LITERATURE REVIEW**	**6**
	2.1 Overview	6
	2.2 Definitions	6
	2.2.1 Requirement Volatility	6
	2.2.2 Software Architecture	7
	2.2.3 Relationship between Requirement Volatility and Software Architecture	8
	2.2.4 Positive Implications of Requirement Volatility on the Software Architecture	9

2.3 Preliminary Studies 9
2.3.1 Representation of Existing Studies 15
2.4 Summary of the Chapter 19

3 METHODOLOGY 20

3.1 Introduction 20
3.2 Systematic Literature Review 20
3.2.1 Review Planning 21
3.2.2 Review Conduction 26
3.3 Grounded Theory 28
3.4 Expert Review 29
3.4.1 Expert Identification 29
3.4.2 Selection Criteria 30
3.4.3 Expert Selection 31
3.4.4 Issue Familiarization 31
3.4.5 Collection of responses 31
3.5 Presentations of Results 31
3.6 Industrial Survey 32
3.6.1 Research Question and Research Objective 32
3.6.2 Identification of Research Objective 33
3.6.3 Identification & Characterization of Target Audience 33
3.6.4 Designing of Sampling Plan 33
3.6.5 Designing of Questionnaire 33
3.6.6 Pilot Test Questionnaire 34
3.6.7 Distribution of Questionnaire 34
3.6.8 Analyzing the Final Results & Writing a Report 34
3.7 Phases of Research Study 34
3.8 Summary of the Chapter 36

4 REQUIREMENT VOLATILITY FACTORS RELATED TO THE SOFTWARE ARCHITECTURE 37

4.1 Introduction 37
4.2 SLR Findings 37
4.2.1 Distribution of Studies based on years 38
4.2.2 Distribution on basis of Type of Research Studies 39
4.2.3 Distribution of studies on the basis journal type 40
4.2.4 Selected Conference 41

4.2.5 List of Journals 42
4.2.6 Distribution of factors based on sub-factors/data units 43
4.3 Findings from Grounded Theory 44
4.4 Conduction of Expert Review 47
4.4.1 Expert Evaluation and Suggestion Table 47
4.5 Description of the Identified Factors 60
4.5.1 Software Defects 60
4.5.2 Resource Management 61
4.5.3 Knowledge 61
4.5.4 Communication Issues 61
4.5.5 Dependencies 62
4.5.6 Traceability 62
4.5.7 Dynamic Business Environment 62
4.5.8 Stakeholder 63
4.5.9 Architecture 63
4.5.10 SW Design and Design Implementation 63
4.5.11 Organizational Leadership 64
4.5.12 Adaption to Change 64
4.5.13 SQW Maintenance 65
4.5.14 Artifacts 65
4.5.15 Integration of Usage 65
4.5.16 Trade-off 66
4.5.17 Code 66
4.5.18 Technical Debt. 66
4.5.19 Human Behavior 67
4.5.20 Team 67
4.5.21 Integration of Linkage 67
4.5.22 Documentation 68
4.5.23 Architectural Complexity 68
4.5.24 Requirement Volatility 68
4.5.25 Quality Assurance 69
4.5.26 Security 69
4.5.27 Self-Healing Mechanism 69
4.6 Summary of the Chapter 70

5 INDUSTRIAL SURVEY 71

5.1 Introduction 71

5.2 Industrial Survey Findings 71

5.2.1 Distribution of Respondent's Experiences based on Software Development 72

5.2.2 Distribution of companies based on the study domain 73

5.2.3 Distribution of responses based on the Scope of the Company 73

5.2.4 Distribution of companies based on the working strength 74

5.2.5 Distribution of companies based on SPI Certifications 74

5.2.6 Distribution of respondent firms based on the type of development 75

5.3 Results Analysis and Reporting 75

5.4 Positive implications of Requirement Volatility Factors on the SW Architecture 76

5.4.1 Testing Results/Statistics 76

5.4.2 Analysis of Survey 90

5.4.3 Survey Results from Weightage Values 91

5.5 Positive Implications of identified factors (from the top level to bottom) 92

5.6 Summary of the chapter 93

6 CONCLUSION AND FUTURE WORK 94

6.1 Introduction 94

6.2 Contribution of the Study 94

6.3 Threats Validity 95

6.4 Future Work 95

6.5 Conclusion 95

REFERENCES 97

Appendices A- I 104 -175

LIST OF TABLES

TABLE NO.	TITLE	PAGE
2.1	Preliminary Studies	10
3.1	Research Questions	22
3.2	Digital Data Sources	22
3.3	Identified Keywords	23
3.4	Search Strings	24
3.5	Study Inclusion Exclusion Criteria	25
3.6	Quality Assessment Criteria	26
3.7	The scale of the Quality Assessment Checklist	26
3.8	Study Selection Criteria	27
3.9	Data Extraction Form	28
3.10	Selection Criteria for Expert Review	30
3.11	Personal Detail of Evaluators	31
3.12	Research Question for the conduct of the Industrial Survey	32
3.13	Scale Defining the Level of Factors	34
4.1	Distribution of Research Studies Journals	41
4.2	Distribution of Studies Conferences	41
4.3	Distribution of Studies based on Journals	42
4.4	Distribution of Factors based on data units.	43
4.5	Example of Data Encoding	45
4.6	Example of Explicit/Implicit Removal	46
4.7	Expert Review Suggestion and Implementation	47
4.8	After the implementation of the Expert's suggestions the Final list of factors along with their category type and sub-factors.	50
4.9	Tabulated representation of RV Factors related to the SW Architecture	55
5.1	Findings of the Survey	90
5.2	Survey Accepted/Rejected Values	91
5.3	Positive Implications of factors from top to bottom based on the frequency	92

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE
3.1	Steps of Systematic Literature Review	21
3.2	Selection Strategies of Studies	28
3.3	Process of Expert Review	30
3.4	Steps for Survey Conduction	32
3.5	Flow Diagram of Research Study	35
4.1	Graph of included studies as per the publication year	38
4.2	Graph of Studies distribution based on the type	39
4.3	Graph of distribution of studies based on the journal type	40
5.1	Distribution of responders based on Experiences	72
5.2	Distribution of companies based on the study domain	73
5.3	Distribution of the company's responses based on the scope	73
5.4	Distribution of companies based on the working strength	74
5.5	Distribution of companies based on SPI certifications	74
5.6	Distribution of respondent firms based on the type of development	75
5.7	Frequency distribution against the factor Software Defects	77
5.8	Frequency distribution against the factor Resource Management	77
5.9	Frequency distribution against the factor Knowledge	78
5.10	Frequency distribution against the factor Communication Issues	78
5.11	Frequency distribution against the factor Dependencies	79
5.12	Frequency distribution against the factor Traceability	79
5.13	Frequency distribution against the factor Dynamic Business Environment	80
5.14	Frequency distribution against the factor Stakeholder Synchronization	80
5.15	Frequency distribution against the factor Architecture	81
5.16	Frequency distribution against the factor Design Implementation	81
5.17	Frequency distribution against the factor of Organizational Leadership	82
5.18	Frequency distribution against the factor Adaption to Change	82
5.19	Frequency distribution against the factor SQW Maintenance	83
5.20	Frequency distribution against the factor Artefacts	83
5.21	Frequency distribution against the factor Integration of Usage	84
5.22	Frequency distribution against the factor Trade-off	84
5.23	Frequency distribution against the factor Code	85
5.24	Frequency distribution against the factor Technical Debt.	85
5.25	Frequency distribution against the factor of Human Behavior	86
5.26	Frequency distribution against the factor Team	86
5.27	Frequency distribution against the factor Integration of Linkage	87
5.28	Frequency distribution against the factor Documentation	87
5.29	Frequency distribution against the factor of Architectural Complexity	88
5.30	Frequency distribution against the factor Requirement Volatility	88
5.31	Frequency distribution against the factor Quality Assurance	89
5.32	Frequency distribution against the factor Security	89
5.33	Frequency distribution against the factor self-healing mechanism	90

LIST OF ABBREVIATIONS

SDLC	-	Software Development Life Cycle
RV	-	Requirement Volatility
SW	-	Software
RQ	-	Research Question
SDP	-	Software Development Process

LIST OF APPENDICES

APPENDIX	TITLE	PAGE
A	The list of designed search strings for the conduct of SLR.	104
B	Quality Assessment including distribution of studies and participants.	108
C	The data Extractions Forms of the conducted SLR.	114
D	Execution of Data Encoding Technique.	134
E	Execution Of Implicit/Explicit Removal	147
F	Expert Review Evaluation Form	150
G	Survey Form	159
H	The final list of Identified factors along with their categories	170
I	List of included primary studies along with the paper IDs	173

ACKNOWLEDGEMENTS

In the name of Allah, the most gracious and the most merciful. I would like to thank Almighty Allah, Who gave me enough strength to accomplish my thesis work during the COVID-19 pandemic.

I would like to thank ***Dr. Huma Hayat Khan***, who put her maximum potential, valuable constant supervision, encouragement, and constructive suggestions to meet this destination. She is the single one who made me stand again like a pillar of strength in my worst situation. I would also like to thank my research pioneer ***Dr. Muhammad Nauman Malik***, who introduced me to the research methodologies and put his generous expenditure of time and profound interest to familiarize me with the research platform. Moreover, I would like to express my deepest gratitude to the worthy Dean ***Dr. Basit Shahzad***, who has been my best instructor at this esteemed university, and the rest of the Faculty members of the Software Engineering Department who put their maximum potential into making bright future of all of us.

Moreover, I would like to show my gratitude to all of the Experts, who evaluated and validated my research work especially ***Dr. JayaLetchmi A/P T. Sambantha Moorthy*** for her valuable suggestions and recommendations. I would like to express my heartfelt feelings to my Ex-Boss, ***Dr. Faheem Jahangir Khan***, (Senior Research Economist, PIDE), who taught me about the actual worth of ‘writing’. I would also express my gratitude to my current Boss **Mr. Nazir Ahmad**, (Joint Secretary, Cabinet Division), who taught me how to bear the pressure in a challenging environment.

For all whom I did not mention but I shall not neglect their significant contribution, thanks for everything.

DEDICATION

Alhamdulillah… All gratitude be to Almighty Allah for molding me into the person I am today and allowing me to realize my ambition…I dedicate this thesis work to my Supervisor, namely **'Dr. Huma Hayat Khan'** for guiding me and giving me her precious time whenever, I needed it the most during Research work, Co-Supervisor, **'Dr. Muhammad Nauman Malik'** and last but not least my father **'Muhammad Anwar Baig (Late)'**, who raised me like THE STRONGEST AND PROUDEST DAUGHTER but left us at a very early age of life.

I extend this dedication to my beloved '**Mama'**, an iron lady who raised me in the dark time with the light of hope and patience. Today, I am achieving my goals just because of her endless guidance, love, and moral support. Moreover, I am thankful to all of my lovely and gorgeous sisters namely **Humaira**, **Iqra,** and **Dr. Nimra** for their valuable support, love, care, and huge respect.

CHAPTER 1

INTRODUCTION

1.1 Overview

This chapter contains an introduction to the twin peaks of the Software Development Life Cycle (SDLC) i.e. Requirements volatility and Software Architecture. Further, this chapter highlighted the strong relationship between these twin peaks and more crucially focused on the positive implications of requirements volatility and their impact on the software architecture. The background of this chapter discussed the existing research work in the area of the conducted research. Accordingly, the research problem, aim of the research, scope of research, and research questions, are also mentioned here. Moreover, to consider the contributions of the existing research work, this chapter highlighted the existing research gaps, and their limitations and further intimated the basic core purpose or need to conduct this research.

1.2 Background of Research

Requirement volatility is a fundamental activity which requires throughout the software development life cycle (SDLC). It could be raised from the very initial step of elicitation to the end phase of maintenance. Accordingly, requirements are needed to be added, deleted, and modified throughout the rest of the development phases [1][2]. Therefore, it is a challenging activity to adopt, however, it is not necessarily a negative activity, and nor does it means an uncontrolled state of existence. Exiting research studies indicated their positive correlation with the accomplishment of software development projects [3]. In general different terms requirement change, requirement uncertainty, and requirement instability is commonly associated with the same phenomenon of requirement volatility. However, as the name depicts, the rest of these associated terms have negative implications or illustrate an uncontrolled state of existence. Therefore, this study focuses on the term requirements volatility due to its fragile nature, where, it deals with the change during the rest of the development phases and illustrates the control state of existence. Accordingly, it has valuable worth towards the successful accomplishment of the development projects and also has positive worth towards success [1][4].

On the other hand, software architecture shows the complete picture of the software system or product. The core purpose of a software architecture foundation is to provide a complete vision of the upcoming product which is going to develop [5].

However, software architecture itself is complex because it contains various diagrams, use cases, semantics, etc. Therefore, there is a dire need to consider competent software architects to efficiently handle these architecture-related matters. It is pertinent to mention here that the progress of the architecture development is not dependent on the decision of the software architects. The involvement of different stakeholders, their consent, or their vision about the end product also plays a vital role. In short, to better get clear objectives about the behavior of the upcoming product and its environment, there is another essential need to consider the opinions and sound consents of their stakeholders [1][5].

1.3 Research Problem

As requirement volatility is a fundamental activity of the software development life cycle, therefore, it is an essential need to consider the factors that arise during the development process. As a result, the upcoming products would be able to reach the desired and satisfactory level of end-products, in the middle of the volatility [1]. Despite this, requirement volatility is treated as a single phenomenon; however, in reality, different factors can lead to different practical implications and their impact on the development process of software architecture. Therefore, there is a dire need to consider those implications that occur during the development process in different ways. Moreover, to the best of my knowledge, there is less research conducted that intimated their positive implications. Although, several studies have been conducted that came up with different results and motivations.

This existing research gap motivated me to discover all possible factors i.e. external and internal to overlook the requirements volatility on the SW architecture. Hence, the prior studies mentioned the need to conduct this empirical study on the positive implications of requirements volatility on software architecture [1][5][9].

1.4 Research questions

This research study comprises the following listed below research questions.

RQ1: What are the external and internal factors of requirement volatility related to software architecture?

RQ2: What are the positive implications of requirements volatility factors on the software architecture?

1.5 Research Objective

To answer the above-mentioned research questions. The research objectives of this study are listed below:

Objective 1: To identify the possible internal and external factors of requirement volatility related to the software architecture from practitioners and the existing literature.

Objective 2: The objective of RQ2 is to seek the positive implications of identified factors of requirement volatility on software architecture.

1.6 Aim of the Research

This study aims to achieve two main goals. Where the first goal is to study the phenomenon of requirement volatility in respect of the software architecture. The second objective is to seek the sound consent of the industrial practitioners by surveying a list of factors. Besides this, to accomplish the first objective this study aims to conduct a systematic literature review (SLR). As a result, the findings of SLR are analyzed by conducting the Expert Review. For implementation, this study applied the technique of grounded theory i.e. data encoding. After removal of explicit and implicit removal, this study got a refined list of factors. This identified list of factors was validated by the experts of the domain in phase two (02) of Expert Review conduction. In the end, to meet the second objective, this study aims to identify the identified list of factors by conducting an industrial survey and more curiously focused on part of their positive implications of factors on software architecture.

1.7 Scope of the Research

The scope of this study is to identify all possible factors of requirement volatility regarding the software architecture. For this systematic literature review is conducted. For the implementation of this, the primary studies are selected for the period of the last ten years i.e. from 2010 to 2020. At which point, high-quality research papers i.e. journals, mature conferences, and accepted manuscripts are selected. Moreover, this study more curiously focused on the positive implications of the requirement volatility in respect of the software architecture. For evaluation purposes of the SLR results, the expert review was conducted, where, the domain of the experts are selected having experience of at least five years and must be specialized in their domain. On the other hand, to conduct the survey, the industrial people are selected having experience of at least two years in the field of Requirement Engineering or have worked in the domain of the twin's peaks of the software development life cycle i.e. Requirement volatility and Software Architecture.

1.8 Contribution of the Research

This conducted study contributes a list of all possible factors i.e. Internal and External factors of requirement volatility regarding the software architecture. The identified list of factors is vetted through the implementation of the grounded theory technique. As a result, the list is passed out for the removal of data consistency and redundancy. After the execution of the grounded theory, the list of identified factors is refined through the conduct of the expert review. Where, the experts of the domain validated the identified list of factors in three different categories, i.e. internal factors, external factors, and both, respectively. Moreover, the experts also verified their naming conventions for certain factors and suggested their classification against each identified factor. In the end, the industrial survey was conducted which indicated their results and proposed more solutions in a more focused way towards the conduct of this research.

1.9 Thesis Outline

This thesis outline is comprised in numbers of six (06) chapters. Where, the first Chapter contains the overview along with the research background, the research problem, research questions, research objectives, the aim of the research, the scope of the research, and the contribution. In the end, this chapter also contains the outline of the thesis. Chapter two (02) contains the overview and complete information about the requirement volatility, and software architecture and more specifically elaborated on the positive implications of requirement volatility concerning the software architecture. Besides this, this chapter contains part of the literature review and the detail of existing research studies related to the domain of this study. Chapter three (03) contains the complete detail of adopted methodologies against the designed research questions. Where, the systematic literature review (SLR), grounded theory, expert review, and survey were conducted on account of research methodologies. Accordingly, chapter four (04) contains the material related to the findings of the systematic literature review including grounded theory and expert review. Where chapter five (05) contains the results of the industrial survey. In the end, chapter six (06) discussed the conclusions and contributions of this conduct of study along with future work and threats of validity.

1.10 Summary of the Chapter

This summary contains the complete representation of chapter one. Which, the research gap is highlighted along with the background of the study. Moreover, this chapter addressed the purpose of this conduct along with the research questions. Accordingly, reported the research objective, scope of the study, and contributions. In the end, shows the complete representation of the thesis.

CHAPTER 2

LITERATURE REVIEW

2.1 Overview

This chapter contains the existing studies 'literature review' work in terms of requirements volatility and its impact on software architecture. This chapter also highlighted the strong relationship of these twins peaks i.e. 'requirements volatility' and 'software architecture' in the field of software engineering. Accordingly, this chapter discussed the core purpose or need to carry out this research. Moreover, the existing studies are reported hereby along with their contributions and gaps in the research.

2.2 Definitions

2.2.1 Requirement Volatility

The requirements volatility is a fundamental activity which requires throughout the software development life cycle (SDLC). The nature of change is an integral element that triggers during the rest of the phases of software development. Accordingly, 'Nurmuliani et al define requirements volatility as the tendency of requirements to change over time. While requirements changes occurred as a result of the natural evolution of the user needs over time, the actions of stakeholders in the various stages of the requirements engineering process including elicitation, analysis, validation, and management can also contribute to requirements volatility [1].

Prior studies reported on the two main strategies to consider the requirements volatility i.e. 'defensive strategy' and 'reactive strategy'. While the defensive strategy deals with the avoid change or reduce change mechanism. On the other hand, reactive strategy deals with all upcoming changes. Deals with the changes during the development phase and consider a flexible platform for requirement volatility. In the same context, the approach of this study is to consider the reactive strategy and indicates its positive worth towards changing environment [3].

Moreover, it is important to consider the changes as early as possible during the software development life cycle (SDLC) because the upcoming system artifacts are deploying in a dynamic environment. As a result, the requirements are changed continuously and new requirements are triggered, rapidly. Therefore, to reduce the financial implications of new

requirements there is a necessity to consider the requirements as a priority [10]. In the same context, the existing studies also reported that change is an essential and important factor in the software development life cycle and software systems must respond to these upcoming changes, evolving requirements, dynamic platforms, and other requirements volatility pressures [11].

Besides this, the requirement change is a risk-oriented activity, wherein, adopting or predicting the change is a challenging task. Therefore, to consider the changes developers must have to consider the challenges related to the economic or financial impact first then its technical and managerial complexity [12]. Accordingly, selecting the appropriate developer is crucial for getting better results toward change requests in terms of their economic feasibility and time constraints. In the same context, the literature also reported that change requests could be treated as software artifacts that could consider software defects through managing its repository. Where, the repository plays a vital role in software maintenance, predicting changes, and provision of a common platform for communication and coordination purposes among the stakeholders. This activity is also known as change request triage, wherein, the most appropriate and expert developer is selected to consider the upcoming changes, to handle the matter efficiently [13].

2.2.2 Software Architecture

Software architecture is the backbone of the software systems that are going to build. It is, therefore, a software architecture that provides the complete vision of the upcoming software system including diagrams, use cases, and semantics [1]. During the designing process, software architecture plays a vital role in the implementation of source code [14]. Software Architecture has been established for almost forty years in the field of Software Engineering. Inherently, Software Architecture is complex and its manufacturing does not depend on the software architect only. It must involve the different stakeholder's concerns, as well. As a result, the complexity of software architecture increased towards its development [1].

Moreover, the existing empirical studies reported that nowadays software architecture is built based on experiences or intuitions rather than different designing tools and techniques. However, at the initial step, the software architects are always considered as the core persons or have primary concerns for designing the basic structure of the upcoming system architecture. Later on, the active involvement of different stakeholders' concerns such as product managers, developers, and customers is highly considered to fulfill the requirements and to meet the satisfaction level of the end product [1].

Besides this, software architecture is the structure of structures of the upcoming system which contains the software components, and properties of their components and indicates their relationship with each other [15]. Accordingly, software architecture is considered the central element of the software development life cycle. Therefore, it is used for communication and documentation purposes for reasoning about their important properties towards designing, as a blueprint for the upcoming system manufacturing [16].

Software Architecture design implementation is another important element that needed to be addressed. There is an essential need to keep a record of the architecture design rationale, because, the knowledge behind the architectural design decision could be challenging during the software system maintenance and its evolution process [17]. Moreover, the architectural technical debt issues or financial implications also needed to consider most stickily. Because this can adversely influence the development process due to sub-optimal architecture decisions and trade-offs. However, technical debt matters related to coding issues could be detected during the development process by adopting various tools [1] [18].

2.2.3 Relationship between Requirement Volatility and Software Architecture

The prior studies indicated a strong relationship between requirements volatility and software architecture. These both terms are considered as ‘Twin Peaks’ of the software development life cycle and have equal worth towards the development process [1]. In forging this, the requirement engineering process and software architecture are closely interlinked, In what respect; implementing decisions regarding one could also affect the other [5]. Nevertheless, the traditional waterfall model considered the requirements at the preliminary step and freezes the requirements in this stage before moving the design implementations. As a result, it is becoming more difficult to consider the change architecture decisions. However, in reality, the changes could be occurring in both areas of the twin peaks of the SDLC. Therefore, modern iterative development processes such as agile implementations are considering the changes throughout the software development life cycle. This is indicated by their strong relationship [1] [5].

Moreover, this strong relationship indicated about requirement volatility huge impact on the software development life cycle. Where the requirements could adversely affect the software architecture releases and have a strong impact on project schedule, cost, or budget-related matters and their performances [19]. Hence, there is a dire need to consider these twin peaks of the software development life cycle (SDLC) because these terms having equally worth while for the building of the upcoming software system [5]. As a result, the twin peaks

model emphasizes iteratively carrying out the implications of requirements volatility and software architecture, concurrently, to meet the multiple benefits such as rapid changes and uncovering new requirements, etc. [1]. Accordingly, in the modern research community requirements, volatility effects on the software architecture are becoming an important element to consider and the existing literature also reported that there is an essential need to scientifically explore this phenomenon [20].

2.2.4 Positive Implications of Requirement Volatility on the Software Architecture

Prior studies and empirical studies discussed the worth of the requirement engineering process and their strong and positive correlation with the accomplishment of software projects. As evidenced, by the chaos report of the Standish group, four out of ten main success factors are interlinked with the requirement engineering process i.e. user involvement, clear business objectives, minimized scope, and core business needs [3]. Existing studies reported more positive than negative experiences about the requirement engineering process, in which the practitioners intimated that integration of usage, decision knowledge, accountability, and traceability have major roles. In the end, concluded that partially it is difficult to cope with requirements volatility or change, but it can be achieved through the adoption of integration of usage and decision knowledge, positively [21].

Prior empirical studies reported that structural dependencies within code are an upright strategy to identify the change impact set, positively. Where, a change impact set is a group of entities, that have a fragile attitude toward changing behavior, to ensure a consistent and complete change request. Researchers perceived that developers must know the decision to analysis that from where to start the implementations of upcoming changes, even if they may not know all the changes. As a result, revealed that a better understanding of data-sharing dependencies, in addition, has a huge positive impact on the actual change impact set [17].

2.3 Preliminary Studies

This section contains the complete information of prior studies related to the study domain and in terms of studies purpose, used methodologies along with their contributions and limitations. In the same context, this part represented the existing study's significant importance in the field of requirement engineering and software architecture. Where, most of the studies indicated the lacking area of results and proposed success factors towards implementation of requirement volatility or requirement change management process which are not extensively addressed, especially in the context of the industry. Moreover, the two studies came up with the same conclusion and intimated that there is still a gap exist in

understanding the positive impact of requirement volatility on software architecture. Further, intimated that there is an essential need to conduct an empirical investigation on the positive implications of requirement volatility on software architecture and proposed that it will be an authentic platform to conduct better contributed future research [1][5].

Table 2.1 Preliminary Studies

Sr No.	Author/Year	Domain	Methodology	Contribution	Limitation/Future Work
1.	Amjad AbuHassan, 2020	Requirement Engineering	Systematic Literature Review	This study proposed code smell deduction techniques for fixing the changes or modifications during the software design implementations.	There is an essential need to validate the findings of the conducted SLR at the industrial level or from the practitioners to bring more attention in the future.
2.	Halima Sadia, Syed Qamar Abbas, and Mohammad Faisal, 2020	Requirement Engineering, Volatile Requirements, and Prioritization	Conducted case study and empirical analysis	This case study proposed a requirements prioritization technique to handle the requirements volatility.	The case study contains a total number of eighty (80) volatile requirements and the functionality of fifty-four (54) was validated through the proposed solution. But, there is a need to handle the requirements volatility in more generic ways and to identify the actual cause of its

					occurrence.
3.	Sandun Dasanayake, 2019	Requirements Management, Requirements Volatility, and Software Architecture	Conducted an industrial case study.	Identified the factors that contribute towards the requirements volatility and their implications on software architecture with mitigation factors.	The results of the conducted case study were sticker to a European software company. There is a gap found to understand the positive implications of requirements volatility on software architecture at the industry end.
4.	Arif Ali Khan, Muhammad Azeem Akbar, 2019	Requirement Engineering and Requirement Change Management	Conducted SLR and survey	Developed taxonomies of the identified motivator based on the framework proposed by Ramasubbu and PMBOK.	Classified identified motivators were twenty-five in number and sticker to the factor of organization size only. There is a gap found to identify the motivators based on the wide scope and other human resources to validate the empirical findings.
5.	Jan Ole Johanssen,	Requirement Engineering	Conducted semi-	Proposed the two main factors having	The conducted interview was

	2019		structured interview	positive experiences towards continuous software engineering.	limited to 24 practitioners only. However, their attitude towards implementation has positive attitude.
6.	Muhammad Azeem Akbar, 2019	Requirement Change Management	Conducted SLR using Kitchenham and Charters	This study proposed a framework for the GSD to improve the RCM.	The results are based on SLR, which was not validated by the industry. They plan to conduct QS with practitioners in the future, to investigate the success factors that have positive impacts on RCM in GSD.
7.	Xiaoyu Liu, 2018	Requirement Engineering	Empirical Investigation	This study proposed a tool namely CHIP to predict the software's actual change impact set.	The proposed tool has limitations that may be overcome by adopting the F-2 score by comparing the predictor's dependency graph and evolutionary coupling in the future.
8.	Sanja Aaramaa, 2017	Requirement Management and Software Architecture	Conducted an Exploratory case study.	This study identified the challenges that requirements volatility constituted	The result of this case study needed implementation of findings in

				in software architecture.	different sizes and into the different domains to get the validation of identified challenges. Then, better solutions could get to refine the list of challenges posed by the requirement volatility in software architecture.
9.	Mauricio Pena and Ricardo Valerdi, 2014	Requirement Volatility, System Engineering, and Requirement Engineering	Conducted Survey; in five workshops and summarize the requirement volatility into five observations categories	This study, summarize the requirement volatility into five observation categories and proposed a Model to estimate the impact of requirement volatility on System Engineering.	Results of workshop discussions and surveys can be used to develop a better framework with objectives to improve the economic implications of requirements volatility on system engineering efforts.
10.	Andrea Janes, Tadas Remencius, Alberto Sillitti and Giancarlo Succi	Requirement Engineering and Requirement Management	Conducted Interviews through a questionnaire	This study concluded with some factors that indicated some potential areas to improve the RE process on account	This study emphasized one of the parts of the Requirement definition process i.e. ‘defensive strategies’ only.

				of defensive strategies to reduce or avoid changes. As a result, improve customer satisfaction.	There is still a gap that exists to focus on another part of the requirement definition process i.e. 'reactive strategy' to address the requirement volatility towards implementation of flexible software solutions which may require anticipation capabilities.
11.	Michael W. Grenn, 2013	Requirement Engineering and System Engineering	Conducted Simulation Experiment	This study, introduced the Requirement entropy framework to evaluate the upcoming requirements and for estimating the requirement engineering effort.	The results of the simulation are computer-based and may need to evaluate through empirical investigation or a case study to reveal more generic results against the upcoming requirements and the practical utility of the proposed model and its effectiveness.
12.	M. P. Singh, Rajnish	Requirement Engineering	Conducted Exploratory	This study reported on the requirements	The results were based on the

	Vyas, 2012	and Change Management	Review	volatility and its impact on project development phases.	exploratory discussion there is a need to conduct a case study or empirical investigation to get the validation and need to identify the positive impactions of results (if any).

As shown in Table 1. There are numerous studies conducted on requirements volatility and software architecture. Although they have significantly discussed the twin peaks of the software development life cycle i.e. Requirement Volatility and software architecture. However, none of them identified the positive implications of requirements volatility on the software architecture. This is more specifically mentioned by the researchers [1] [5] [19]. Hence, prior studies visibly intimated the need for this conduct of a study on the positive implications of requirements volatility on software architecture [1] [5] [17] [21].

2.3.1 Representation of Existing Studies

This section contains the reported studies description in terms of the study purpose, used methodologies along with their contribution and future work. Amjad Abu Hassan, 2020 [22], discussed that software smell indicated the software codes and design changes related issues. Identifying these kinds of issues is a challenging task. The author further intimated the different smell detection techniques at the code level during the design implementations. For this author carried out a systematic literature review (SLR) to identify the primary studies related to the phenomenon of code smell deduction during software design and coding. As a result, the author proposed different code smell deduction techniques for adopting the changing demands and different concerns of the stakeholders in terms of fixing emerging bugs, adding new functionality or changes, and deleting some old ones. The contributions of this study provide more attention to the research community to consider the several opportunities to consider future research in terms of software development.

Halima Sadia, 2020 [6], reported on the usability of the volatile requirements to improve the software development life cycle on account of prioritization and intimated the importance of requirements volatility. To meet customer satisfaction, the authors considered

the requirements volatility at each step and further intimated to consider each change with prioritization towards the provision of successful project development. In this context, authors proposed prioritization techniques to adopt the volatile requirements using fuzzy logic. The authors discussed prioritizing each raised requirements volatility. But, there is still a gap that exists to identify that what are the actual cause of requirements volatility towards implementation of the proposed requirement prioritization technique, in more generic ways, to handle its occurrence.

Sandun Dasanayake, 2019 [1], reported that requirements volatility is a major cause of project failures on account of cost overrun, project delays, and huge defect density issues. In the same context, the author further reported on the threat of software architecture, which interprets the complete vision of software products. The authors highlighted the factors that become the cause of requirements volatility and further highlighted inadequate architecture documentation, incomplete design rationale, and complexity as the major implications of requirements volatility on software architecture. The author discussed that there is a dire need to handle contributed factors of requirements volatility to mitigate its implications and for the provision of healthy software architecture process development.

Arif Ali Khan, 2019 [2], reported on the adoption of high-quality and low-cost products, which were strongly associated with challenges of requirements change management. They highlighted the severe threat of requirements change, on account of top priority challenges, in the field of global software development. The authors discussed the results of different identified motivators and highlighted the result which was correlated to the requirements of change management. In forging this, they introduce a framework for tracking those challenges, which were found significant for the success and evolution of software development firms.

Jan Ole Johanssen, 2019 [21], reported on continuous software engineering and intimated about the unexploited areas to consider the utilization of usage and decision knowledge towards software development. For this, the author conducted semi-structured interviews with 24 practitioners from different 17 companies. In this context, the practitioners reported more positive than negative experiences. As a result, the study proposed the validation of two main factors i.e. integration of usage and decision knowledge into continuous software engineering, and intimated that partially it is difficult to adopt but it has a positive attitude towards changes or adoption in continuous software engineering.

Muhammad Azim Akbar, 2019 [4], reported on the challenges faced by global software development firms that were strongly related to requirement change management. In

the same context, authors identified thirty challenges that were further classified in the domain of client and vendor end. The core purpose of the author was to get a sound vision of the requirements to change the process and its challenges at both ends of software development firms. Authors intimated that there are fewer studies exist, that investigate the factors that have positive impacts on the requirement change management process. They further highlighted that there is a dire need to evaluate industrial investigation for getting the sound implications of requirements to change practices, which is important for the progress of software development organizations.

Xiaoyu Liu, 2018 [17] discussed the structural dependencies within code and reported on its worth in predicting the requirement change impact set towards implementation of the requirements volatility. In some respect, the results indicated that a better understanding of data dependencies instead of calling the dependencies greatly improves the change impact set. For this author proposed a new improved tool namely CHIP to predict the software's actual change impact set. The execution of this proposed tool intimated novel extensions to reduce the false positives and suggested that developers must be aware from the initial changes that from where to start the changes in the source code. This approach has been evaluated empirically on the four large-scale open-source systems. The author contributed that demonstration on the data sharing dependencies having a sound impact on the software's actual impact set predictions as compared to call dependencies.

Sanja Aaramaa, 2017 [5], discussed the relationship between SW architecture design and requirements volatility and intimated about less research work in this domain. For this, the authors conducted an exploratory case study to report how requirements volatility affects the software architecture design. The results of this study intimated that requirements uncertainty and dynamic business environment-related factors are the root cause of requirements Volatility. The authors identified the challenges of requirements volatility that constitute software architecture design at worst, especially, in the context of scheduling and architectural technical debt. The authors discussed the possible mitigations factors and further intimated about the strongly influenced factors of requirements volatility on software architecture. In the end, this study also highlighted that there is an essential need to fill the gap that what are the factors needed to identify towards mitigate the volatility risks and intimated its higher industrial relevance, for future research purposes.

Mauricio Pena, 2014 [7], reported on the causes of Requirements Volatility and its impacts on System Engineering toward dynamic environment over the system development life cycle. The core objective of this study was to improve the competency of the system

analyst to positively accommodate the requirements volatility. The authors proposed a model to better quantify the impact of requirements volatility and on account of the results identified the five observations that summarize the key activities of requirements volatility. Authors intimated that project organizational, technical and contextual are the baseline factors that have a higher influence on the rise of requirements volatility, due to poor understanding of the required system and customer needs on account of their first observations. The authors discussed their results and recommendation with an intimation that fewer considerations of requirements volatility could increase the higher requirement change management issues during transitions towards the software development life cycle. In forgoing this, they intimated that volatility has strongly influenced system engineering efforts and it could increase the functional size of the project. As result, major re-designing work could enhance. The authors highlighted that impact of requirements volatility varies through the adoption of changes that are dependent on added, deleted, and modified. For future considerations, the authors highlighted that there is an essential need to work for the improvement of the proposed model for the provision of better economic implications of requirements volatility on system engineering efforts.

Andrea Janes, 2013, discussed the two main strategies that deal with the changes in the requirements i.e. defensive strategy and reactive strategy, respectively. The study reported some factors that can be adopted to reduce or avoid changes e.g. through using an effective requirements definition strategy and indicated one of the main reactive strategies to address the requirement volatility to produce flexible software solutions. This study concluded with proposed factors having capabilities to handle the defensive strategies and presented some solutions to implement the flexible solutions, recommended generating the special clauses for requirement changes, and suggested adopting a well-defined change request procedure, to better assist towards changing in the project.

Michael W. Green, 2013 [8], discussed information quality to address the system engineering process. They reported that requirements are the core part of system engineering at the technical end of designing, implementation, and integration. They further reported that both are the core concept of Software Engineering. The authors introduced a requirements entropy framework for information quality. They intimated that information must require refreshing till the end state of the project and the end state could be getting only when it contained the maximum nomenclature of quality attributes as per the demanded end state. They proposed that the impact of addition, deletion, and modification could be measured through raised inconsistency against the information.

The results of this study suggested that the requirement entropy framework is a robust method to evaluate the upcoming requirements and for estimating the requirements engineering effort for system development programs. M.p. Singh, and Rajnish Vyas, 2012 [9], discussed the requirements volatility and also reported on the causes of it. The author emphasized the impact of requirements volatility on account of project schedule, cost, performance, Quality, and Maintenance. This study was a sticker to report the aspects of requirements volatility and intimated that exploring the positive implications of requirements volatility is a good platform for future research.

2.4 Summary of the Chapter

This chapter reported on the existing literature related to the twin peaks of the software development life cycle (SDLC) i.e. ‘Requirement Volatility’ and ‘Software Architecture’. Further, reported their strong relationship during the software development process. Besides this, this chapter also contains the complete representation of existing studies in descriptive and as well as in tabulated form. Where, most of the prior studies reported on requirement volatility and software architecture, in terms of their success factors, prioritization techniques, considerations at the initial phase of development, and adoption through the software development phase. This chapter more clearly reported the gap in the study, as none of them identified the positive implications of requirements volatility on the software architecture. A further report on the gap, in light of the existing literature work, is more specifically mentioned by the other researcher. In the end, this chapter visibly intimated the core purpose of this conduct of a study on the positive implications of requirements volatility on software architecture.

CHAPTER 3

METHODOLOGY

3.1 Introduction

The second chapter intimated about the part of this study conducted literature review that was carried out to highlight the existing gap in the field of requirement engineering and further highlighted the twin peaks of the software development life cycle (SDLC) i.e. Requirement Volatility and Software Architecture. Whereas, the existing literature more curiously focused on the positive implications of requirement volatility and their impact on the software architecture. Moreover, this chapter contains complete detail in all aspects of the adopted set of methodologies in terms to carry out this research.

3.1.1 Overview

As mentioned earlier, this chapter indicates the part of the conducted research methodology. Where all the sets of adopted research methodologies and chosen designs were elaborated. To proceed, the most reliable, familiar, and well-established protocols are used on account of research methodologies. Besides this, a 'Systematic Literature Review' (SLR) is conducted, to identify the factors of the twin's peaks of the SDLC i.e. requirement volatility and SW architecture. As a result, at the preliminary stage, the collected information visibly has some redundancy and inconsistency. To rectify this, a technique of Implicit and explicit removal has been chosen. For implementation, the grounded theory technique namely data encoding was used. After execution, a refined list of factors is identified. To avoid biases these identified factors were passed through the second phase of the research methodology i.e. 'Expert Review'. Wherein, generated factors were validated and evaluated by the Experts in the domain. In the end, the final validated and evaluated a list of factors was passed through the next phase i.e. 'Industrial Survey' for obtaining better results from the practitioners or industrial people.

3.2 Systematic Literature Review

For conducting research, a well-addressed, familiar, and extensively used protocol namely Systematic Literature Review (SLR) is conducted. Which comprises three different phases i.e. 'Review Planning', 'Review Conduction', and 'Results Reporting', respectively. For implementation, this study implemented the guidelines of Kitchenham [23]. The core purpose of this conduct of SLR is to identify the factors of the twin peaks of the software

development life cycle i.e. Requirement volatility and software architecture. In the same context, each adopted step of systematic literature review (SLR) is manifest below:

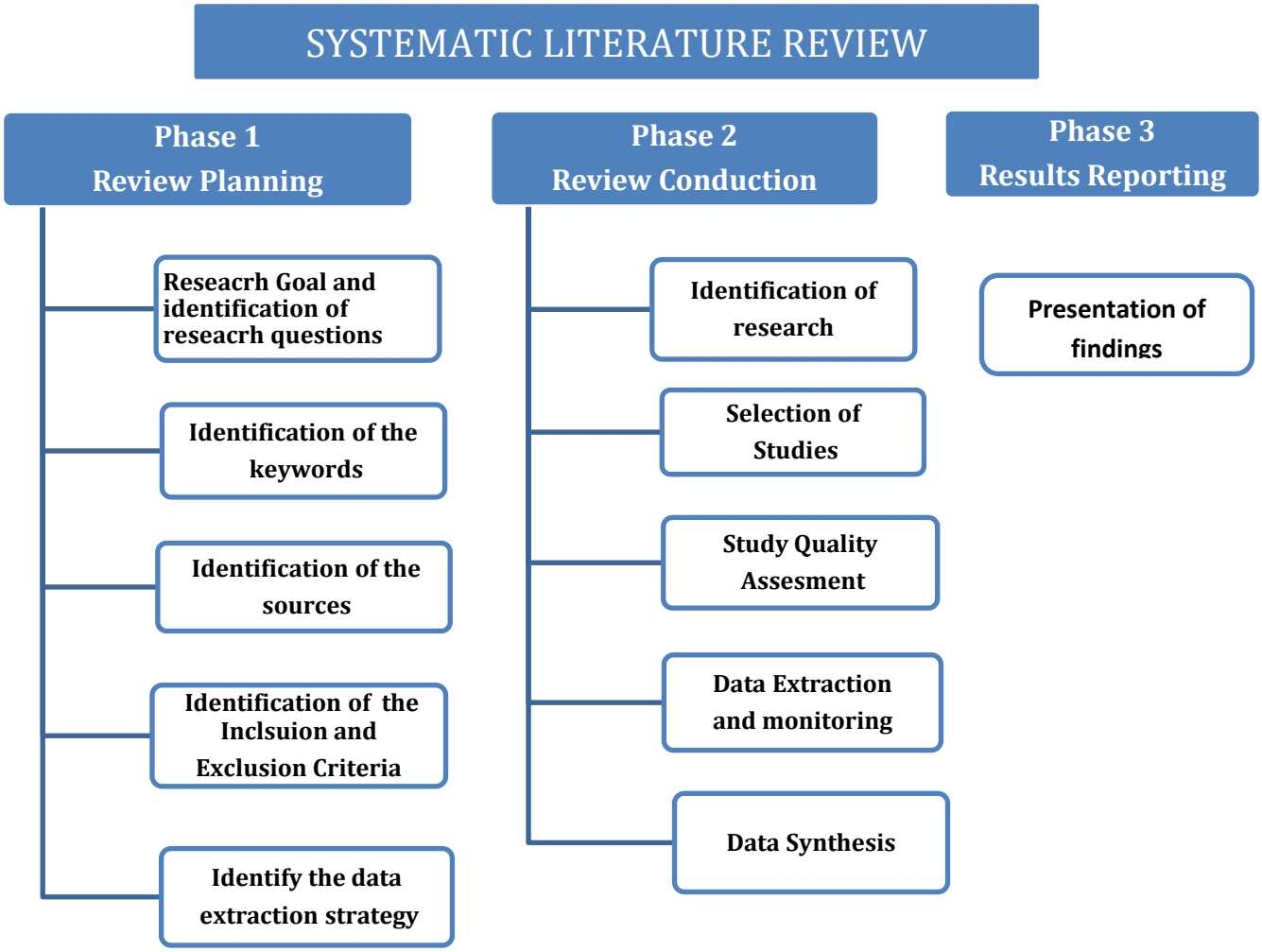


Figure 3.1 Steps of Systematic Literature Review

3.2.1 Review Planning

Review planning is the first phase of Systematic Literature Review (SLR), where, the planning strategies are initiated to conduct the research.

3.2.1.1 Research Goal

This study being piloted has a specific goal, at which point, the core purpose of this conduct is to identify all possible factors of requirement volatility on software architecture. Accordingly, the primary goal of this study is to identify the internal and external factors of requirement volatility related to the SW architecture.

3.2.1.2 Research Questions

To meet the goal of this study, hereby, the two research questions are developed for the smooth conduct of the Systematic Literature Review (SLR). In the same context, the complete detail of the research questions and their rationale are tabulated below:

Table 3.1 Research Questions

ID	Research Question	Rationales
RQ1	What are the external and internal factors of Requirement Volatility related to the Software Architecture?	This research question would be able to identify all possible factors of Requirement Volatility related to Software Architecture from practitioners and the existing literature, in terms of their three different categories i.e. external, internal, and both.
RQ2	What are the positive implications of Requirement volatility factors on the software architecture?	This research question aims to seek the positive implications of the identified list of factors specifically regarding the software architecture.

3.2.1.3 Strategy

As the name depicts, this part intimated the study adopted strategy. Where the resources are chosen for retrieval of primary studies through generated research terms or keywords.

3.2.1.4 Resources

To retrieve the data or primary studies on the electronic medium different resources are used, where, the majority of the considered studies are journals articles, accepted manuscripts, mature conferences, and special issue papers. In the same context, the literature related to the requirement engineering process, software engineering, and computer sciences are considered to achieve the goal. Whereas, books and simple printed articles are not chosen to retrieve the study material. Accordingly, the list of databases on account of chosen resources is tabulated below:

Table 3.2: Digital Data Sources

Electronic database	URL
IEEE	https://ieeexplore.ieee.org/Xplore
Willey	https://onlinelibrary.wiley.com
Science Direct	https://www.sciencedirect.com
ACM	https://dl.acm.org

3.2.1.5 Search Terms

As mentioned above, for extraction or retrieval purpose of data the search queries are designed, here. For implementation, the mechanism of search query generation is listed below:

- At first instance, the most striking words of this study are considered on account of the core keywords that are three in number i.e. Requirement, Volatility, and software architecture, respectively.
- In the second step, similar words or synonyms are generated against the derived core keywords. As a result, this study finds similar words i.e. thirteen (13), fifteen (15), and six (06), respectively against the derived words. As a result, collectively, in numbers thirty-four (34) similar words are listed.
- In the last step, the most appropriate and vetted synonyms are considered which are four (04), five (05), and four (04) against these derived words, respectively. As a result, collectively, in numbers thirteen (13), the most striking or appropriate keywords are considered for making the search strings. Accordingly, the search terms are tabulated on the upcoming page:

Table 3.3 Identified Keywords

Sr No.	Keyword	Synonyms	Considered Synonyms
1.	Requirement	Demand	Demand
		Condition	Condition
		Essential	Essential
		Need	Need
		Precondition	
		Specification	
		Stipulation	
		Fulfillment	
		Imperative	
		Provision	
		Prerequisite	
		Necessity	
		Needful	
Total		**13 Nos.**	**04 Nos.**
Sr No.	**Keyword**	**Synonyms**	**Considered Synonyms**
2.	Volatility	Change	Change
		Changeable	Changeability
		Fluctuating	Uncertainty
		Fickle	Unstable
		Inconsistent	Inconsistent
		Mutable	
		Uncertain	
		Unstable	

		Unsettle	
		Unstable	
		Unsteady	
		Capricious	
		Unpredictable	
		Unreliable	
		Untrustworthy	
Total		**15 Nos.**	**05 Nos.**
Sr No.	**Keyword**	**Synonyms**	**Considered Synonyms**
3.	SW Architecture	SW Design	SW Design
		SW Structure	SW Structure
		SW Construction	SW Construction
		SW Building	SW Building
		SW Planning	
		SW Architecture	
Total		**06 Nos.**	**04 Nos.**
Grand Total		**34 Nos.**	**13 Nos.**

To proceed, the selected keywords are used simultaneously along with the rest of the identified keywords via using the Boolean operator. As a result, the one hundred and fifty (150) in numbers generic search strings are derived for execution or retrieval purposes of the study material from the selected resources. Accordingly, the derived list of search queries is tabulated below:

Table 3.4: Search Strings

String #	Search String
Attp#1	(((Requirement) AND volatility) AND “Software architecture”)
Attp# 2	(((Requirement) AND volatility) AND “Software design”)
Attp#3	(((Requirement) AND volatility) AND “Software structure”)
Attp#4	(((Requirement) AND volatility) AND “Software construction”)
Attp#5	(((Requirement) AND volatility) AND “Software building”)
. .	. .
Attp#56	(((Demand) AND inconsistent) AND “Software architecture”)
. .	. .
Attp#78	(((Condition) AND uncertainty) AND “Software structure”)
. .	. .
Attp#129	(((Need) AND change) AND “Software construction”)
. .	. .
Attp#141	(((Need) AND unstable) AND “Software architecture”)
. .	. .
Attp#150	(((Need) AND inconsistent) AND “Software building”)

Moreover, the table of complete search strings has been attached in the Appendix section shown as Appendix-A.

3.2.1.6 Selection Criteria

For the selection of the research studies, the quality papers are selected from the four different databases i.e. IEEE, Willey Online Library, ACM Digital Library, and Science Direct. Moreover, for the smooth conduct of a Systematic Literature Review (SLR), the published material between the years 2010 to 2020 timeframe are selected. Accordingly, the majority of the selected study's journal articles, accepted manuscripts, mature conferences, and special issue papers. In the same context, the complete detail of inclusion and exclusion criteria is tabulated below:

Table 3.5: Study Inclusion Exclusion Criteria

Inclusion Criteria	Exclusion Criteria
Study material related to the domain of Requirement Engineering selected. Whereby, the studies on the topic of requirement volatility and software architecture, published within the time frame of the last ten years i.e. 2010 to 2020 are considered.	The material published earlier or before the stated timeframe i.e. from the year 2010 is not included.
Published journal articles, mature conferences (started from the 15th onward), manuscripts, reviews, and special issue papers.	Books, tutorials, panel discussions, editorials, proceedings, unaccepted papers, and unauthentic material is not included.
The majority of the studies discussed the twin peaks of the software development life cycle i.e. Requirement volatility and software architecture and their factors. Moreover, those studies more curiously focused on their positive aspects.	The studies with the other domain, specific knowledge, and applications are not included.
Studies published in the English language.	Studies published in other than the English language.

3.2.1.7 Study Selection Procedure

The systematic literature review is a time taking activity. For the accomplishment of this, different three stages of the process and further series of steps have to implement. In

the same context, this study adopted the guidelines of Kitchenham [23], for the smooth conduct of the SLR. As far as the concern about the study selection criteria, firstly fixed the timeframe and the studies of the last ten years 2010 to 2020 are selected. During the selection of the studies, the most relevant and stickier titles, and keywords are retrieved. Moreover, considered the inclusion and exclusion criteria and applied filter. In the end, the final reviewed studies are selected as shown.

3.2.2 Review Conduction

This is the second phase of the Systematic Literature Review (SLR) and plays a vital role in refining the study material through performing the quality assessment. Besides this, indicates the complete vision of selected material in numbers and categorized all of them in terms of different attributes.

3.2.2.1 Study Quality Assessment checklist and procedure

To avoid biases, this step plays a major role in conducting the process of quality assessment and during conduct more refined the considered studies. For implementation, this study adopted the pre-defined questions of the guideline of kitchenham [23]. The respondents and participants have to answer these questions against each study for selection purposes. Accordingly, assigned them a certain scale against each question, as shown in the table. For this, studies are circulated among the respondents, participants, and candidates for the smooth conduct of the process of quality assessment. As a result, studies are selected on behalf of these respondents or participants.

Table 3.6: Quality Assessment Criteria

Sr No.	Quality Assessment Question
1.	Are the aims clearly stated?
2.	Is the finding Credible and important?
3.	Are the prediction techniques used clearly described and their selection is justified?
4.	Is the knowledge or understanding been extended by the research?
5.	Is the diversity of perspective and context been explored?
6.	Are the links between data, interpretation, and conclusions clear?
7.	Does the detail/ depth/ complexity/of the data is conveyed?

As mentioned earlier, the scale of quality assessment checklist adapted from the guidelines of the Kitchenham [23], is tabulated below:

Table 3.7: Scale of Quality Assessment Checklist

Answer	Score
Yes	1
Partially	0.5
No	0

Moreover, the completed detail of this conduct of quality assessment activity is attached herewith in Appendix-B. In forgoing this, the selected material is tabulated below, which comprises the different attributes i.e. databases, found studies, titles & keywords, Abstracts, repeated studies, quality assessment, and final reviewed selection. Wherein, the attributes of the database indicate the selected data sources and the very next attribute of found studies indicated the found studies against each respective database. Besides this, the attribute of title and keywords indicates all those studies which were stickier to the identified keywords and title of the study. In forgoing this, the attribute of the Abstract indicates about all of the studies have similarities to the research gap of this study. Besides this, for removal of redundancy, the attribute of repeated studies indicated all those studies that have already been considered and repeated here more than one time. Moreover, the attribute of Quality Assessment indicated the core activity of SLR, at which point, the founded studies are refined by the various respondents and participants, individually. In the end, the final reviewed and vetted studies are selected for smoothly conducting this research.

Table 3.8: Study Selection Criteria

Sr. No.	Database	Found studies	Title & Keywords	Repeated studies	Abstract + Conclusion	Quality Assessment	Final Selected study
1.	**IEEE**	10,765	220	20	142	54	23
2.	**Wiley Online Library**	21,058	524	363	132	35	19
3.	**Science Direct**	41,532	529	268	128	62	33
4.	**ACM Digital Library**	5,344	182	159	23	23	08
	Total	**78,699**	**1,455**	**810**	**425**	**174**	**83**

The search strings are applied to four different digital libraries i.e. IEEE, Wiley Online Library, science direct, and ACM digital libraries. At the first level of extraction, 1455 studies were found based on title and keywords. In the second step, 810 repeated studies are included only once. In the third step, 410 studies are selected in respect of the Abstract and conclusion. To consider the inclusion and exclusion criteria, the quality assessment is performed on 174 studies. As a result, in numbers of 83 studies are selected on account of the final selected studies. Accordingly, the following upcoming flow chart represents the complete vision of the adopted procedure for the selection of studies.

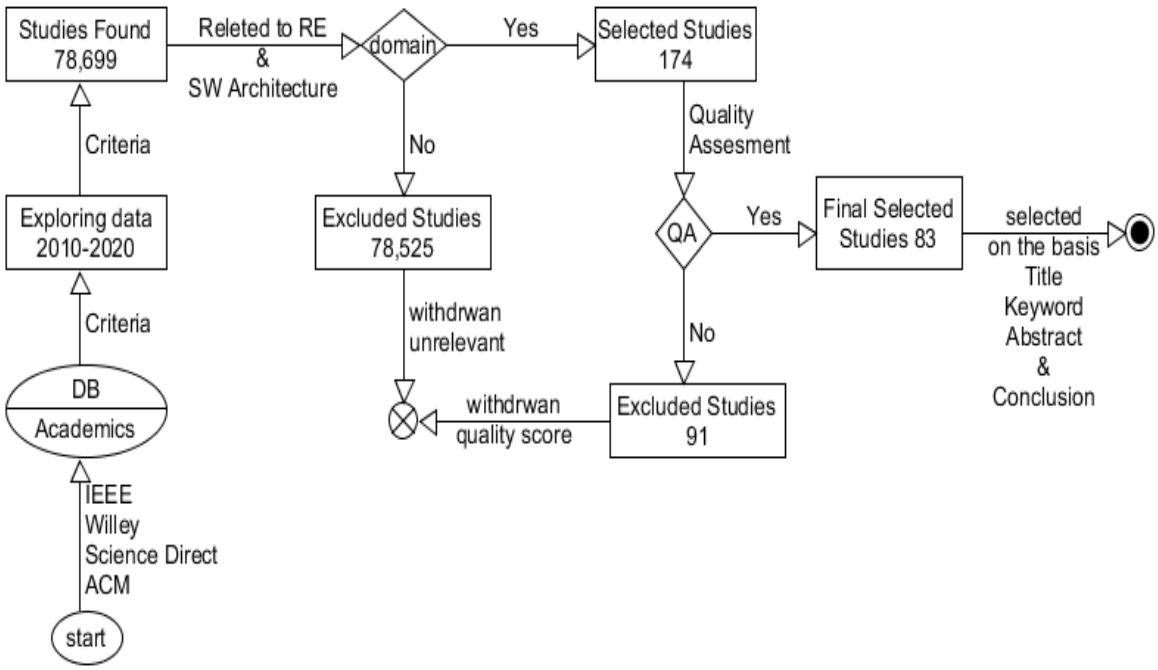


Figure 3.2 Selection strategies of studies

3.2.2.2 Data Extraction Strategy and Synthesis of Extracted Data

To extract and synthesize the data the extraction form was designed for presenting the complete detail of selected studies and retrieved information in a tabulated form as attached herewith, in Appendix-C. Wherein, the data extraction form is comprised of seven (07) different entities along with their respective information, as tabulated below:

Table 3.9: Data Extraction Form

Entities	Respective Data
Title	
Paper ID	
Type	
Publisher	
QA Score	
Answer to RQ1	
Status	

3.3 Grounded Theory

To establish a theory, an inductive and comparative technique was adopted namely 'Grounded Theory. This provides the complete platform for gathering qualitative data. Then, synthesize and analyze the data for making a meaningful theory. Besides this, it is highlighted

that Barney G. Glaser and Anslem L. Strauss developed the grounded theory for analyzing qualitative data. Wherein, the data encoding and general concept of (codes) are used for the extraction purpose of the data. In the same context, this technique aims to retrieve refined and unbiased data from the selected primary studies. Hence, for implementation of this, the data encoding technique of grounded theory is applied to the selected primary studies for the extraction purpose of the refine list of factors. Accordingly, the complete detail of implementation is placed in the forthcoming chapter 4. Moreover, to see the complete execution of the process, in terms of the data encoding technique, the explicit and implicit removal see Appendix-D and Appendix-E, respectively.

3.4 Expert Review

To consider the results of grounded theory, there is an essential need to conduct an expert review against the identified list of factors, for validation purposes. Accordingly, the expert review was conducted, whereas, the identified list of requirement volatility factors on SW architecture was evaluated by the experts in the domain of Requirement Engineering. As a result, the findings of this conduct of Systematic Literature review (SLR) were validated by the experts/scholars in the field. Accordingly, the following strategies were adopted to conduct the expert review.

3.4.1 Expert Identification

This is the very first and the most important step of the expert review. In which, the experts of the domain were selected for validation purposes of the work. For onward implementation, the basic competency of this step is to identify the field experts which have sound knowledge related to this study domain and have experience. In the same context, the identified experts validate that work, such as the evaluation of identified the list of factors in terms of different categories, for checking the purpose of their naming convention or classification. In forgoing this, here Figure No.3.3 represents the complete vision of the process of expert review.

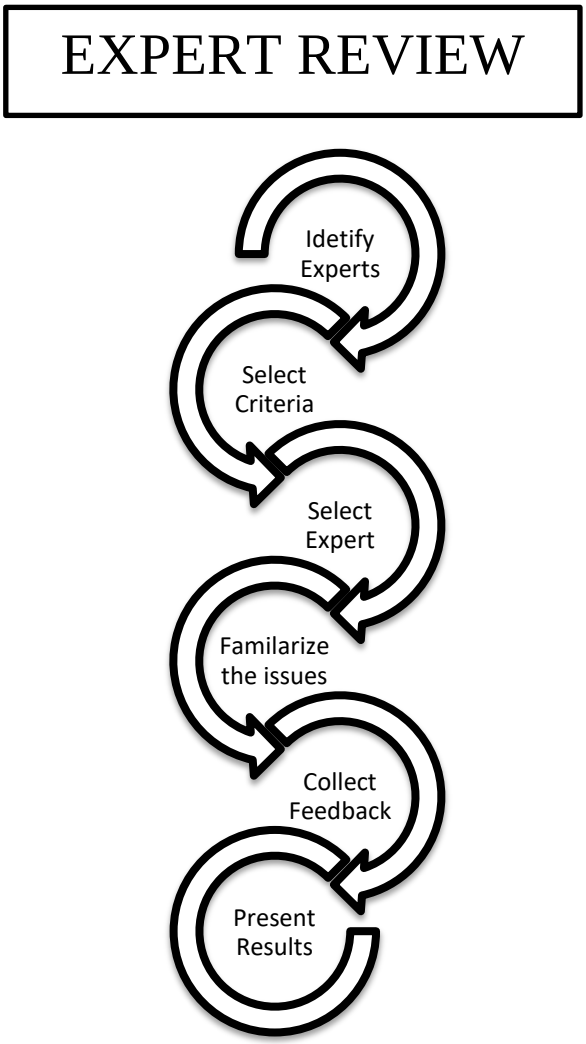


Figure 3.3 Process of Expert Review

3.4.2 Selection Criteria

To conduct the smooth exercise of expert review, the expert of the domain needed to select for the review purpose of the work. Hereby, the complete selection criteria in terms of their work experiences and expertise are tabulated below:

Table No. 3.10 Selection Criteria for Expert Review

Required Experience	Required Expertise
Having experience of a minimum of 10 years to a maximum of 20 years.	Having expertise in the domain of the Requirement Engineering or specialist.
Having experience of a minimum of 10 years to a maximum of 20 years.	Having expertise in the domain of Software Architecture or specialist.

Here, the table of the selection criteria for expert review indicated about the reviewer (s) expertise must be specialized in the domain of requirement engineering and software architecture. Accordingly, selected evaluators must have a maximum experience of 10-20 years at the ends of industry or academia.

3.4.3 Expert Selection

Here, the experts are selected as per the selection criteria, a total in numbers of six experts was approached during this part. As a result, this study got responses from four (04) different experts in the domain. Accordingly, to validate the research study, it is necessary to review the work by a minimum of 1 to 4 experts for validation purposes. Therefore, this study fulfills the basic criteria of expert selection.

3.4.4 Issue Familiarization

To proceed, there is necessary to familiarize the selected reviewer(s) with the research problem, research purpose, and data collection for the validation purpose of the study. Moreover, it is quite a challenging task, to sum up, stuff in the design expert evaluation form, as well. Here, the willing experts are familiar with this stuff related to the study.

3.4.5 Collection of responses

The core purpose of this step is to get responses from worthy experts in the domain. Accordingly, the collected responses are placed at the end of Appendix-F.

3.5 Presentation of Results

After getting the sound consent of the worthy reviewer (s) the results are presented in the tabulated form. Where the results are placed i.e. final list of identified factors of requirement volatility related to the SW architecture along with their categories placed here in the forthcoming chapter 4. The mentioned categories are categorized where these factors are being a lie, in terms of ‘Internal’, ‘External’, and ‘Both’. Moreover, the complete detail of the worthy experts in terms of their respective organizations and designations is also tabulated, here.

Table 3.11 Personal Detail of Evaluators

Expert No.	Organization’s Name	Designation
Evaluator No. 1	National University of Modern Languages (NUML), Islamabad.	Dean FE&CS/ Associate Professor
Evaluator No. 2	University of Technology, Malaysia.	Assistant Professor/IT Officer
Evaluator No. 3	University of Manchester, England.	Associate Professor
Evaluator No. 4	University of Vienna, Austria.	Assistant Professor

3.6 Industrial Survey

To meet the RQ2, here, the industrial survey was conducted. Whereby, the primary goal of this method is to get the sound consent of the practitioners about the identified list of requirement volatility factors and ensure their positive implications on the Software Architecture. For implementation, the most popular protocol is used hereby proposed by Mark Kasuinic [24]. This is the widely used guideline to survey the field of Software Engineering. Besides this, Figure 3.4 illustrates the complete vision of the steps carried out to conduct the survey.

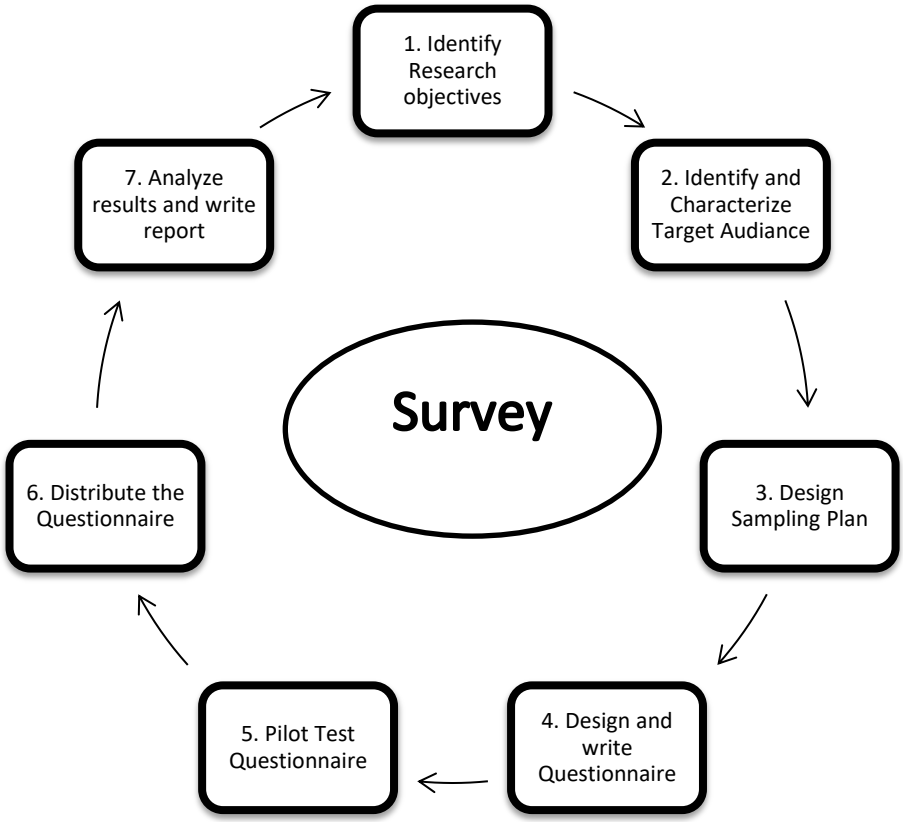


Figure 3.4 Steps for Survey Conduction

3.6.1 Research Question and Research Objective

Table 3.12 Research Question for Conduct of Industrial Survey

ID	Research Question	Rationales
RQ2	What are the positive implications of requirements volatility factors on the software architecture?	This research question aims to indicate the positive implications of an identified list of factors specifically regarding software architecture.

3.6.2 Identification of Research Objective

This is the very first step of the survey. Where the basic competency is to identify the research goals and objectives. In the same context, the purpose of this conduct of survey is to propose the positive implications of requirements volatility factors on the software architecture.

3.6.3 Identification & Characterization of Target Audience

This is the second step of a survey which is comprised of two activities. Whereas, in the very first activity is to identify the respondents. Moreover, this is a more challenging task to identify the target audience who could understand the research questions and acknowledge the research terminologies. Here, the audiences are selected based on their designations, geography, demography, and their experiences. Who has worked on the twin peaks of the software development life cycle (SDLC), specifically in the domain of Requirement Engineering and Software Architecture? After the audience is selected, the second step is to analyze the audience and characterize the intended respondent. To get better results, here, a web-based questionnaire was conducted. Accordingly, for the smooth conduct of the survey, the industry practitioners are selected having more than two (02) years of experience in the study domain.

3.6.4 Designing of Sampling Plan

This step indicates the sample size of the respondents participating in the survey. Besides this also intimates about requires responses and how much the sample size is enough for responses. Here, the industrial practitioners are selected on account of the selected sample populations who have worked in the domain of Requirement Engineering and Software Architecture. As a result, the random sampling strategy is selected, and the sample size of 91 people from the software industry. In forgoing this, the sample size is calculated by using Cochran‘s formula for sample collection.

3.6.5 Designing of Questionnaire

To facilitate the respondents, here, in this step questionnaire is designed. For smooth designing, the basic necessity is to keep in mind the research objectives. In the first instance, internally the survey questions are proposed and then transformed into the shape of a questionnaire. As a result, the questionnaire design is comprised of two sections. Whereas, section one contains the demographic information of the participants in terms of their designations, locations, industry types, and this study-related completed projects, etc. While

section two contains the complete detail regarding the list of identified factors to get the positive implications of requirements volatility on software architecture based on a pre-defined scale, as tabulated below:

Table 3.13 Scale defining the level of factors

Scale	Score
Strongly Agree	1
Agree	2
Neutral	3
Disagree	4
Strongly Disagree	5

In light of the above, the prepared questionnaire is placed herewith in Appendix-G.

3.6.6 Pilot Test Questionnaire

To get authentic responses from the targeted audience, here, in this step, the prepared questionnaire test through the conduct of a small-scale simulation. The core purpose of this exercise is to remove the bugs from the design questionnaire.

3.6.7 Distribution of Questionnaire

The designed and tested questionnaire was distributed to the selected audience for getting the responses. At the preliminary stage, the survey was distributed via email and LinkedIn Corporation. Moreover, to get more rapid responses, the IT parks of Islamabad/Rawalpindi are visited for getting the sound consent of the industrial practitioners.

3.6.8 Analyzing the Final Results & Writing a Report

Once the responses are collected, the findings are presented, here. Accordingly, the appropriate method was used to represent the results. Besides this, the findings of the conducted survey could also be reported in a written form, where, the solutions and recommendations are made based on the results. In the same context, the detail is represented in the forthcoming chapter 5. Accordingly, the complete survey form along with the identified factors is also placed herewith, in Appendix-G.

3.7 Phases of Research Study

This part represents the complete vision of this study in aspects of all activities that have been carried out to conduct this research. It is pertinent to mention here that the study is based on the two research questions. Where the first objective of this study is to identify a list of requirements volatility factors and their impact on the software architecture by the conduct of a Systematic Literature Review (SLR). On the other hand, the second objective of this

study is to represent the positive implications of requirements volatility factors on Software Architecture by the conduct of an industrial survey. Accordingly, Figure 3.5 represents the complete vision of this conduct of the study and their carried phases in the shape of the flow diagram.

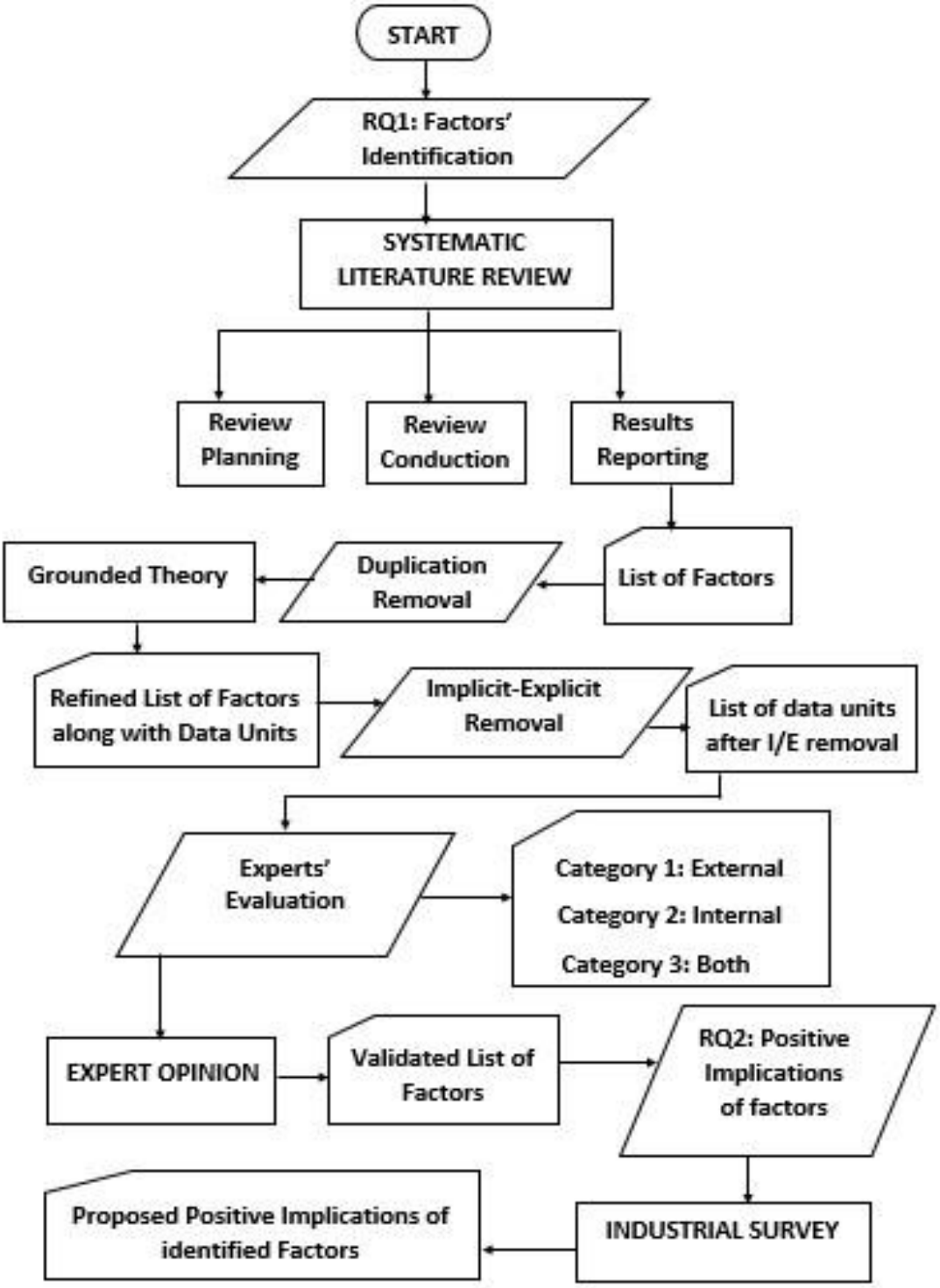


Figure 3.5 Flow Diagram of Research Study

The flow diagram of the research study represented the flow of this conduct of research study. Whereby, for the accomplishment of the RQ1 and to meet the first objective of this study, a Systematic Literature Review (SLR) [23] is conducted, which is based on three different phases ‘Review Planning’, ‘Review Conduction’ and ‘Results Reporting’, respectively. As a result, this study obtained the list of factors. Then, the duplication removal is performed on the list o factors. Moreover, to refine the list of identified factors, the Grounded Theory was implemented through a data encoding technique. During this phase, different extractions of constructs are generated to get the refined list of factors along with the data units, then, an activity of implicit explicit removal is conducted. After the successful execution of the implicit explicit removal, this study gets the final list of factors. As the result, Expert Review is conducted for the validation purpose of the identified list of factors. Where, the factor is defined into three different categories i.e. ‘external’, ‘internal’, and both. After the successful implementation of the Expert Review, this study find the validated list of factors and accomplished the first objective and RQ1 of this study. To meet the second objective of this study and accomplishment of RQ2, an industrial survey is conducted. The core purpose of this conduct of industrial survey empirically investigates the positive implications of requirement volatility factors on the software architecture, from the practitioners or industry people. As a result, this study proposed the positive implications of requirements volatility factors on the software architecture and contributed their part through this conduct of the study.

3.8 Summary of the Chapter

This chapter contains complete detail in terms of adopted methodologies during this conduct of research study. Where, the Systematic Literature Review (SLR), is conducted for the identification of the requirement volatility factors on the software architecture and validated through the conduct of an Expert Review. Accordingly, an industrial survey is conducted for getting the positive implications of these identified factors on the software architecture. In the end, the flow diagram of the research study is represented diagrammatically.

CHAPTER 4

REQUIREMENTS VOLATILITY FACTORS RELATED TO THE SOFTWARE ARCHITECTURE

4.1 Introduction

This chapter contains the complete information related to the findings of this research study design that has been carried out to achieve the research goals. This study is based on the two research questions, which depict the aim of this study. To meet the research objectives, the finding of the systematic literature review is reported here along with the identified list of factors. Then, identified factors are modified by the implementation of the grounded theory. Accordingly, the grounded theory generated meaningful constructs by adopting the data encoding technique. In the end, the results of these constructs are validated by the experts in the domain through the conduct of Expert Review. As a result, the refined list of factors is placed in the form of a final list of factors along with their sub-factors or data units. After the accomplishment of this final list of factors, an industrial survey is conducted to propose the positive implications of requirements volatility on the software architecture.

4.2 SLR Findings

This section reported the findings of the conducted Systematic Literature Review (SLR). The core purpose of this conduct of SLR is to accomplish the research question one i.e. RQ1. Where the objective of RQ1 is to identify the requirements volatility factors related to the software architecture. In the same context, the complete detail of the SLR findings in terms of the distribution of studies based on years, and distribution of the studies as per their nomenclature i.e. Journals and conferences are represented. Moreover, the details of the names of the journals and conferences are also represented.

4.2.1 Distribution of Studies based on years

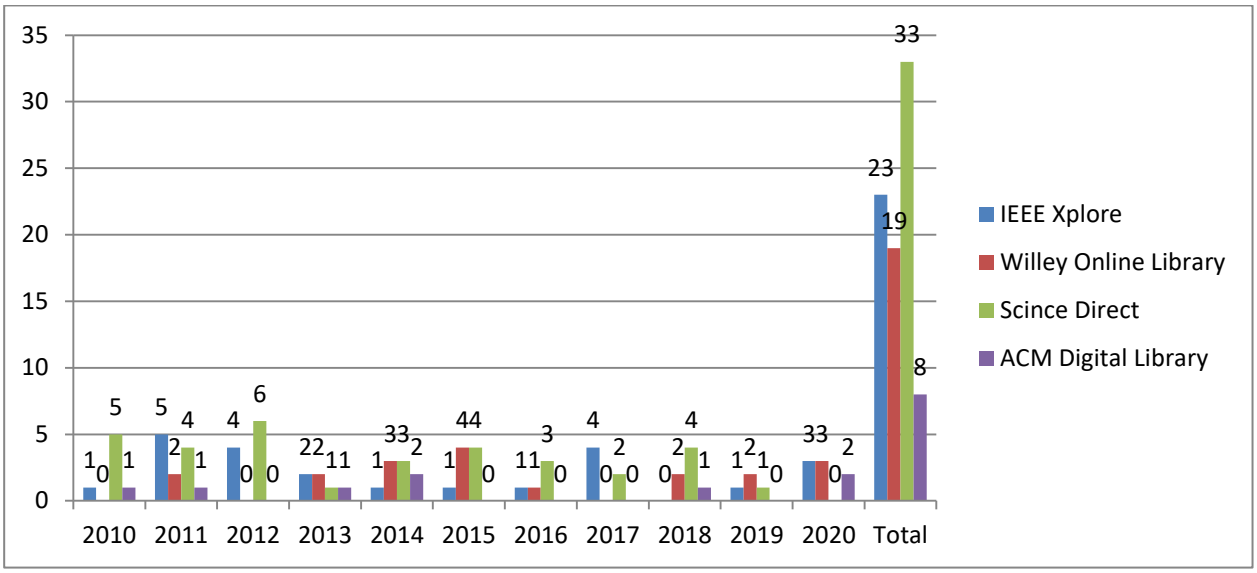


Figure 4.1 Graph of included studies as per the publications year

The above graph represents the included studies as per the publication years. In the same context, the inclusion criteria of this conduct of SLR are the time frame from the year 2010 to 2020. Here, the x-axis of the graph represents the publication years and the y-axis represents the number of included studies. This studies graph contains the data of four digital resources i.e. IEEE Xplore, Willey Online Library, Science Direct, and ACM digital library, respectively. Whereby, the blue color represents the studies of IEEE, the Red color represents the studies of Wiley studies, the green color represents the studies of science direct, and the purple color represents the studies of ACM. This graph also represents the individual rank of each data base on account of their selected studies. Therefore, it is highlighted that in the year 2010, one (01) study is published in IEEE, five (05) studies are published in Science direct, and one (1) was published in ACM. However, this year no relevant studies published in the Willey. In forgoing this, here the five (05) studies are published in IEEE, two (02) published in the Willey, four (04) published in Science Direct, and one (01) published in the ACM. Accordingly, a total number of twenty-three (23) studies were published in the IEEE, nineteen (19) published in the Willey Online Library, thirty-three (33) published in the science direct, and eight (08) published in the ACM. Therefore, this study collectively included eighty-three (83) numbers of studies on part of this thesis work.

4.2.2 Distribution on basis of Type of Research Studies

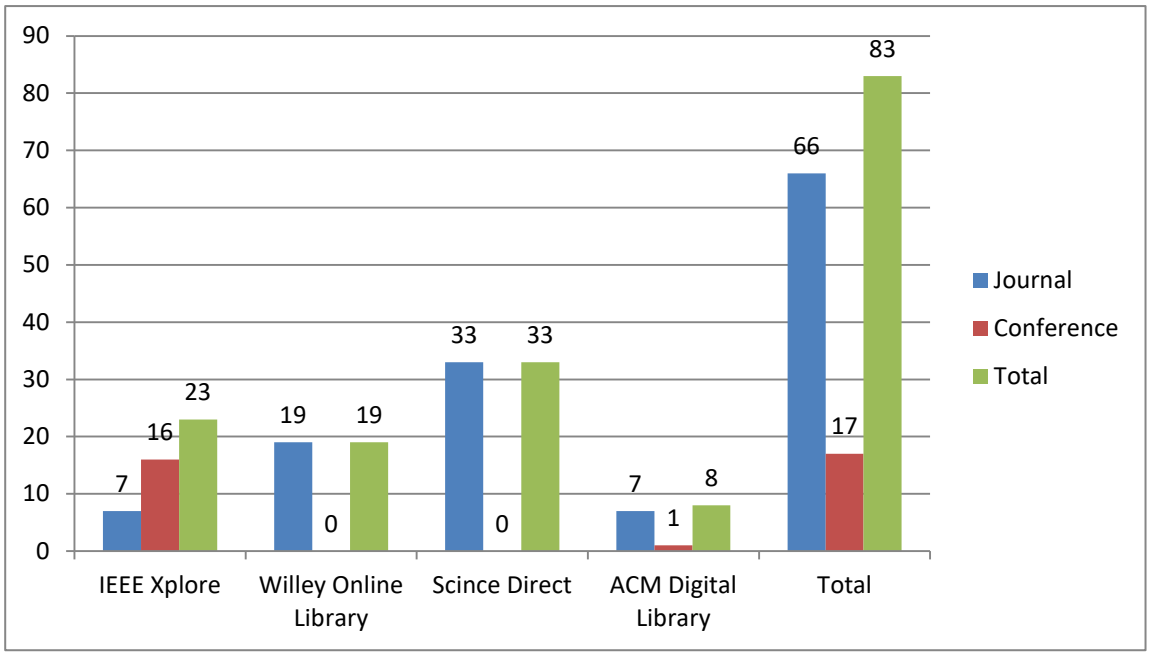


Figure 4.2: Graph of studies distribution based on the type

This graph represents the distribution of research studies based on the type of studies. At this point, the x-axis indicated the type of studies against all selected databases and the y-axis indicated the number of selected studies. Accordingly, the blue color represents the included journals, and the red color indicated the conferences. Besides this, the green color represents the total number of studies. Moreover, the selected data sources are presented in the sequence i.e. IEEE Xplore, Willey Online Library, Science Direct, and ACM digital library, respectively. In the end, the total number of studies is also represented in this graph. In forgoing this, it is highlighted that a total number of seven (07) journals and sixteen (16) conferences are included in the IEEE. However, nineteen (19) and thirty-three (32) journals are found in Wiley and Science Direct, respectively. It is pertinent to mention here that no conference was considered in both respective DBs. Besides this, seven (07) journals and one (01) conference were found at the ends of the ACM digital library. In the end, there are collectively Eighty-three (83) studies are selected to conduct this research, where, sixty-six (66) are published in journals and seventeen (17) are published in conferences.

4.2.3 Distribution of studies on the basis journal type

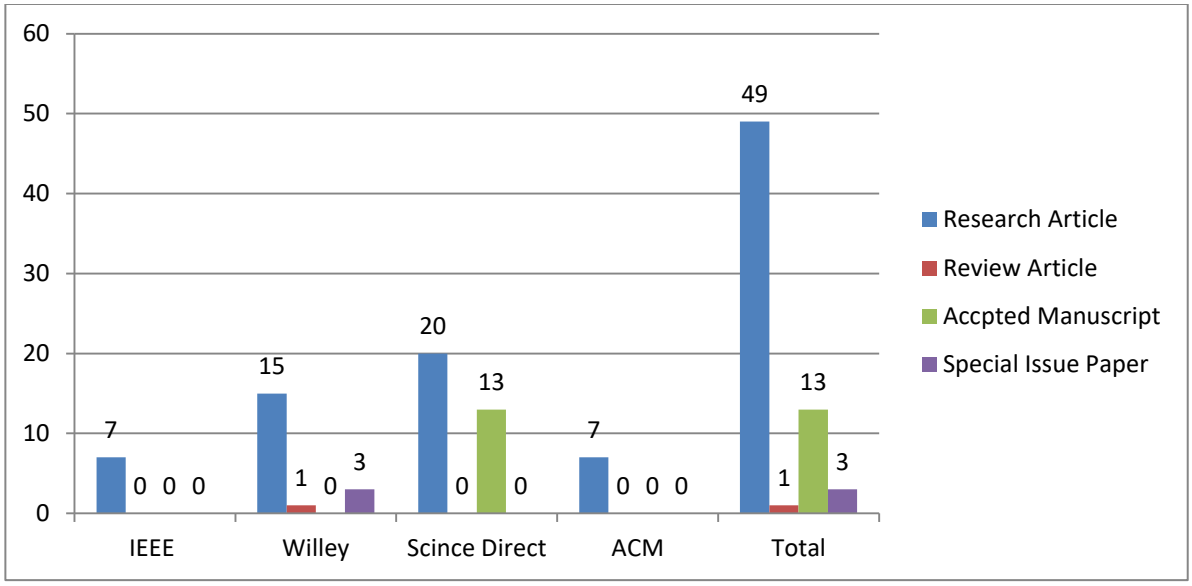


Figure 4.3: Graph of the distribution of studies based on the journal type

This graph represents the distribution of studies based on the type of journal i.e. 'Research Article', 'Review Article', 'Accepted Manuscript', and 'Special Issue Paper'. In the same context, at the ends of the x-axis, the blue color indicates the research article, the red color indicates the review article, the green color represents the accepted manuscript, and the purple indicates the special issue paper. While the y-axis intimated the number of studies considered against these types of journals. Accordingly, seven (07) research articles are considered in IEEE. In forgoing this, no review article or accepted manuscript is found in this DB. From Wiley, fifteen (15) research articles, one (1) review article, and three (3) special issue papers were selected. However, from the DB of Science Direct, twenty (20) research articles, and thirteen (13) accepted manuscripts are selected. In the end, seven (07) research articles are selected from the ACM. Hence, collectively, forty-nine (49) research articles, one (01) review article, thirteen (13) accepted manuscripts, and three (03) special issue papers were considered to conduct this SLR.

Keeping given the above, the complete detail related to the distribution of studies in terms of the Research articles, Review Paper, accepted manuscript, and special issue papers is mentioned hereby, in the next, in the shape of tabulated form i.e Table 4.1.

Table 4.1 Distribution of Research Studies Journals

DB's	Research Article	Review Article	Accepted Manuscript	Special Issue Paper
IEEE	I1, I2, I3, I4, I5, I6, I7	_	_	_
Willey	W2, W3, W4, W5, W6, W7, W8, W9, W10, W11, W12, W13, W15, W17, W18	W19	_	W1, W14, W16
Science Direct	SD2, SD4, SD6, SD8, SD10, SD11, SD12, SD13, SD14, SD15, SD16, SD17, SD20, SD21, SD22, SD23, SD25, SD27, SD29, SD33	_	SD1, SD3, SD5, SD7, SD9, SD18, SD19, SD24, SD26, SD28, SD30, SD31, SD32,	_
ACM	A1, A2, A3, A4, A5, A6, A7	_	_	_

Table 4.1 indicates that seven research articles (I1, I2, I3, I4, I5, I6, I7) published in IEEE, fifteen research articles (W2, W3, W4, W5, W6, W7, W8, W9, W10, W11, W12, W13, W15, W17, W18) published in Willey, twenty research articles (SD2, SD4, SD6, SD8, SD10, SD11, SD12, SD13, SD14, SD15, SD16, SD17, SD20, SD21, SD22, SD23, SD25, SD27, SD29, SD33) published in Science Direct and seven research articles (A1, A2, A3, A4, A5, A6, A7) published in ACM. Besides this one review article (W19) and three special issue papers (W1, W14, W16) were published in Willey. Moreover, thirteen accepted manuscripts were published in Science Direct.

4.2.4 Selected Conferences

Table 4.2 Distribution of Studies Conferences

Study ID	Particulars of Conference
I8	International Conference and Workshops on the Engineering of Computer-based Systems (ECBS)
I9	The Asia-Pacific Software Engineering Conference
I10	The Asia-Pacific Software Engineering Conference
I11	The International Conference on Computer Software and Applications.
I12	The International Requirements Engineering Conference (RE)
I13	The International Conference on Evaluation & Assessment in Software Engineering
I14	The Australian Conference on Software Engineering
I15	The International Conference on Automated Software Engineering
I16	The International Conference on Software Engineering (ICSE)
I17	The International Conference on Automated Software Engineering
I18	The International Requirements Engineering Conference
I19	The Latin American Computing Conference (CLEI)

I20	The Annual Computer and Applications Conference
I21	The International Conference on Model-Driven Engineering Languages and Systems (Models)
I22	The Hawaii International Conference on System Sciences
I23	The Australian Software Engineering Conference
A8	The International Conference on Software Engineering (ICSE)

4.2.5 List of Journals

Table 4.3 Distribution of Studies based on Journals

Study ID	Particulars of Journals
I1, I2,	IEEE Systems Journal
I3, I6	IEEE Access Journal
I4	IEEE Transactions on Software Engineering
I5	IEEE Transactions on Systems Man, and Cybernetics
I7	IEEE Transactions on Software Engineering
W1, W3, W4, W5, W8, W9, W11, W12, W13, W14, W18, W19	Journal of Software Evaluation and Process
W2, W10, W16	Journal of Software Practice and Experience
W6	Journal of IET software
W7	Journal of Software Testing, Verification, and Reliability
W15	Journal of System Engineering
W17	Journal of Institution of Engineering and Technology
SD1, SD2, SD3, SD4, SD5, SD8, SD11, SD12, SD13, SD16, SD17, SD18, SD20, SD21, SD22, SD22, SD24, SD26, SD29, SD30, SD31, SD33	The Journal of Systems & Software
SD6, SD7	The Journal of Information and Software
SD9, SD19, SD27	The Journal Science of Computer Programming
SD10, SD15, SD32,	The Journal of Information and Software Technology
SD14	The International Journal of Project Management
SD25	The Journal of Measurement
A1	Journal of ACM Transactions on Software Engineering and Methodology
A2	Journal of ACM Computing Surveys
A3, A4, A5	Journal of ACM Transactions on Software Engineering and Methodology
A6	Proceedings of the ACM on Human-Computer Interaction
A7	Journal of ACM Transactions on Autonomous and Adaptive Systems

4.2.6 Distribution of factors based on sub-factors/data units

Table 4.5 represents the distribution of factors based on their data-units or sub-factors. Which is comprised of five different attributes i.e. Factor number, Paper ID, name of Factor, Data units, and references, respectively.

Table 4.4 Distribution of Factors based on data units.

F #	Paper ID	Factor	Data Units	Reference
1.	W1, W7, A1, A4	SW Defects	1-6	[1] [25] [26] [27]
2.	I6, W1, W4, W8	Resource Management	7-12	[28] [1] [84] [85]
3.	I19, I21, W8, W14, W15, SD3, SD11, SD16, SD29, SD31, SD33	Knowledge	13-14	[29] [30] W8 [21] [86] [31] [20] [32] [33] [34]
4.	I5, I12, I14, W1, W16, SD33, A1	Communication Issues	15-22	[35] [35] [36] [1] [87] [34]
5.	I6, I10, I13, W1, W9, A1	Modules Dependencies	23-31	[28] [37] [38] [1] [17] [26]
6.	I6, I19, W14, W17, W19, SD4, SD17, SD24, A4	Traceability	32-37	[28] [29] [21] [39] [18] [10] [40] [41] [27]
7.	I14, A1	Dynamic Business Environment	38	[36] [26]
8.	I13, I15, I16, I17, W4, SD12, SD29, A6	Stakeholder	39-43	[36] [42] [43] [44] W4 [45] [32] [46]
9.	I1, I2, I3,I8, I9, I11, I16, I19, W1, W2, W5, W6, W15, W18, SD1, SD4, SD6, SD7, SD8, SD9, SD10, SD11, SD12, SD14, SD15, SD16, SD18, SD19, SD20, SD22, SD23, SD26, SD27, SD28, SD31, SD32, SD33, A2, A8	Architecture	44-52	[47] [48] [49] [50] [31] [51] [43] [29] [1] [88] [14] [52] [53] [54] [10] [55] [56] [11] [57] [58] [20] [45] [12] [59] [60] [61] [62] [16] [63] [64] [15] [65] [33] [66] [34] [67] [68]
10.	I1, I3, I4,I5, I8,I11, I13, I15, I18, I21, I22, W10, W12, W17, W5, SD5,	SW Design	53-55	[47] [49] [69] [93] [50] [51] [42] [70] [30] [71] [38] [72] [22] [39] [14] [73] [74] [16] [65] [26]

	SD13, SD20, SD28, A1, A2, A4, A5, A6			[67] [27] [75] [46]
11.	I10	Wrong Organizational Choice	56-57	[37]
12.	I11, W10, W16, SD6, SD23	Adaption strategies and Policies	58-61	[51] [72] [87] [55] [64]
13.	I1, W13, W19, A3	Maintenance	62-63	[47] [89] [18] [76]
14.	I7, A6, A8	SW Artefacts	64-68	[77] [46] [68]
15.	W14	Usage	69-70	[21]
16.	I12, I19, SD29, A2	Trade-off	71-72	[35] [29] [32] [67]
17.	I13, W9, W11, W12, SD2, SD26, A3	Code	73-75	[38] [17] [90] [22] [78] [79] [76]
18.	SD1	Technical Debt.	76-77	[54]
19.	SD7, SD9, A6	Human Behaviour	78-79	[56] [57] [46]
20.	W9, SD22, SD30, SD8	Lack of Verification	80-82	[17] [63] [13] [11]
21.	I2, I5, SD13	Team Cohesion	83-85	[48] [92] [74]
22.	I12	Lack of Explicit Linkage	86	[35]
23.	I6, W1, W4, A5	Documentation	87-89	[28] [1] [84] [75]
24.	I21, W4, W13, SD2, SD21, SD25, SD29, A1, A3	Complexity Concerns	90-91	[30] [84] [89][78] [80] [81] [32] [26] [76]
25.	I3, I8, I9, I13, I16, I17, I18, W1, W2, W3,W9, W11, SD10, SD11, SD14, SD22, SD30, SD8	Requirement Volatility	92-103	[49] [50] [31] [38] [43] [44] [70] [1] [88] [91] [17] [90] [58] [20] [12] [63] [13] [11]
26.	I3, I11	Quality Assurance Concerns	104-106	[49] [51]
27.	SD17	Security Concerns	107	[40]
28.	A7	Self-Healing Mechanism	108	[82]

4.3 Findings from Grounded Theory

As a result of this conduct of Systematic Literature Review (SLR), this study identified factors based on different retrieved data units from the selected studies. Whereas, the different authors represented different proposals. However, few authors represented the same ideas. To fix this redundancy and inconsistency, there is an essential need to consider all

those studies only once at a time which has the same idea or purpose. For this purpose, the grounded theory is implemented for the effective removal of redundancy or duplication of data. Moreover, the data encoding technique is applied as shown in Table 4.5. As a result, the sound full constructs originated as shown in Table 4.6 on account of implicit/explicit removal.

The following table contains the complete information related to the implementation of the data encoding technique. where, the attribute namely 'Paper ID' represents the included studies, and 'Paper statement' represents the data of respective studies for extraction purposes of constructs via encoding. As a result, the generated codes are placed in the attribute of 'Respective Code'. In the same context, the complete detail is tabulated below:

Table 4.5: Example of Data Encoding

Paper ID	Paper Statement	Respective Code	Data Encoding
I3	"Dynamic system who's Constituent Systems (CS) is not known precisely at design time, and the environment in which they operate is uncertain. Moreover, unknown conditions and volatility have significant effects on crucial Quality Attributes (QAs) such as performance, reliability, and security."	I3L3, I3L5, I3L6	SW Design AND Volatility AND Quality Assurance
W1	"Requirements volatility is a major issue in software development, causing problems such as higher defect density, project delays, and cost overruns. A software architecture that guides the overall vision of software product is one of the areas that is greatly affected by requirements volatility."	W1L3, W1L4, W1L6	Higher defect density AND Resource management AND Architecture
SD7	"Despite past empirical research in software architecture decision-making, we have not yet systematically studied how to perform such empirical research. Software architecture decision-making involves humans, their behavioral issues, and practice."	SD7L5, SD7L4	Human Behavior AND Architecture
A1	"In today's volatile business environments, the collaboration between information systems, both within and across company borders, has become essential to success. A key challenge is to manage the ever-growing design complexity. In this article, we argue that software architecture should play a more prominent role in the development of collaborative applications."	A1L1, A1L5, A1L7, A1L9	Dynamic Business Environment AND SW Design AND Architecture Complexity AND Communication Issues

The above example indicated the execution of the data encoding technique. Wherein, the second tuple of code is retrieved from the base paper of this study and discussed the

requirements volatility, and their major causes and further intimates about a few factors which become the main cause and show their implications in the context of software architecture.

Accordingly, the respective code was generated as W1L3, W1L4, and W1L6. As a result, the data encoding accomplishment with the refine sets of factors. Besides this, the complete detail in terms of data encoding is hereby placed in Appendix-D.

It is highlighted that different authors represented the same phenomenon with the same naming convention. However, some authors reported the same phenomenon but with different contexts or naming conventions. It is, therefore, for smooth representation of this data encoding technique in a refined shape, an excise of explicit and implicit removal conducted, as tabulated below:

Table 4.6 Example of Explicit/Implicit Removal

Paper ID	Constructs	Implicit & Explicit Removal
W1, W7, A1, A4	Higher defect density, fault proneness, defect proneness, SW failure logs, error handling, pre-release failure, and post-release failure.	SW Defects
I5, I12, I14, W1, W16, SD33, A1	Poor communication, user-centric communication, coordination mechanism, interaction mechanism, lack of communication, communication gap, message exchange, and information distortion.	Communication Issues
I13, I15, I16, I17, W4, W18, SD12, SD29, A6	Stakeholder Synchronization, Stakeholder Goal, Stakeholder Involvement, User Involvement, and Stakeholder Objectives.	Stakeholder
I3, I8, I9, I13, I16, I17, I18, W1, W2, W3,W9, W11, SD10, SD11, SD14, SD22, SD30, SD8	Ambiguous Requirement, Awareness of requirement volatility, High level of Evaluability, Changing user needs, Tracing the requirement, requirement specification, non-functional requirements, unnecessary changes, anticipated changes, changing to code, knowledge of initial changes.	Requirement Volatility

As a result, the duplication of data and any kind of redundancy and inconsistency are rectified by the run of explicit and implicit removal, and multiple constructs are generated against each selected study, to get the refine sets of factors. Accordingly, the implementations are as placed in Appendix-E.

4.4 Conduction of Expert Review

For smooth conduct of the Systematic Literature Review (SLR) and after the successful implementation of the Grounded Theory, an exercise of Expert review was conducted. Moreover, to meet the basic criteria of this conduct of the expert review, four (04) experts are selected hereby having sound knowledge about the domain of this study, for evaluation purposes of the identified list of factors. Where different authors suggested their sound consent and recommended some suggestions. As a result, the suggestions are considered and placed hereby in the suggestion table as tabulated below.

4.4.1 Expert Evaluation and Suggestion Table

After the successful execution of the Expert Review. The suggestions of all the Experts are adopted and implemented hereby in this study. Accordingly, Table 4.7 represents the complete details in terms of the Expert's suggestions and implementations.

Table 4.7 Expert Review Suggestions and Implementation

Expert Reviewers	Comments	Response	Action Taken	Reference
Evaluator 1	The suggestion is to re-consider the factors based on their categories. Where few factors belong to category three i.e. Both (Internal and External)	Thank you for your valuable comment. The factors are placed in category three i.e. both.	SW Defects, Dependencies, Architecture, SW Design, Adaption to change, Human Behaviour, Team, and Integration of Linkage are considered on account of category 3 i.e. Both.	I1, I2, I4, I5, I3, I6,I8, I9, I10, I11, I12, I13, I15, I16, I18, I19, I21, I22, W1, W2, W5, W6, W7, W9, W10,W12, W15, W16, W17, W5, W18, SD1, SD4, SD5, SD6, SD7, SD8, SD9, SD10, SD11, SD12, SD13, SD14, SD15, SD16, SD18, SD19, SD20, SD22, SD23, SD26, SD27, SD28, SD31, SD32, SD33, A1, A2, A4, A5, A6, A8
Evaluator 2	The suggestion is to re-consider the naming convention of identified factors.	Thank you for your valuable comment. The suggested naming convention is considered against the identified	Module Dependencies to Dependencies.	I6, I10, I13, W1, W9, A1
			Wrong Organizational Choice to Organizational Leadership.	I10
			Adaption Strategies	I11, W10, W16,

		factors.	and Policies to Adaption to Change.	SD6, SD23
			Code to Source Code	I13, W9, W11, W12, SD2, SD26, A3
			Lack of verification of Requirement	SD22, SD30, SD8
			Team Cohesion to Team	I2, I5, SD13
			Lack of Explicit Linkage to Integration of Linkage	I12
			Complexity Concerns to Architectural Complexity	I21, W4, W13, SD2, SD21, SD25, SD29, A1, A3
			Quality Assurance Concerns to Quality Assurance	I3, I11
			Maintenance to SQW Maintenance	I1, W13, W19, A3
			SW Artefacts to Artefacts	I7, A6, A8
			Security Concerns to Security	SD17
			Usage to Integration of Usage	W14
Evaluator 3	The suggestion is to merge the two identified factors. i.e. 'Requirement ' and 'Volatility' as Requirement Volatility.	Thank you for your valuable comments. The suggested factors have been merged.	The interlinked two factors namely 'Requirement' placed at serial No. 20 and 'Volatility' placed at serial No. 25 of the expert review have been merged and considered as a single factor i.e. 'Requirement Volatility'.	I3, I8, I9, I13, I16, I17, I18, W1, W2, W3,W9, W11, SD08, SD10, SD11, SD14, SD22, SD30
Evaluator 4	The suggestion is to combine a few sub-factors or data units. Besides this, suggested refining the	Thank you for your valuable comment. The suggested data units or sub-factors have been combined. Moreover, the	The data units namely stakeholder goal and stakeholder objective combined as a single sub-factor.	I13, I15, I16, I17, W4, W18, SD12, SD29, A6
			Poor communication and lack of	I5, I12, I14, W1, W16, SD33, A1

	data units or sub-factors in terms of their scope and meaning and re-consider as a single one.	highlighted data units or sub-factors have been fixed or re-considered in terms of their scope/meaning and treated as a single one.	communication are refined and treated as a single one.	
			Coordination and Interaction Mechanisms are refined and treated as a single one.	
			Dependency on modules and dependency b/w SW Components refined and treated as a single one.	I6, I10, I13, W1, W9, A1
			Safety artifacts and the structural relationship have been removed or replaced on account of scope.	I7, A6, A8

Evaluator 1 suggested that to re-consider the factors based on their defined categories. (i.e. Categor1: Internal, Category 2: External, and Category 3: Both) Where, a few factors belonging to category 3, 'Both' (i.e. Internal and External) to consider the valuable suggestions of the worthy experts of the domain the highlighted factors placed at the ends of category three i.e. both. Accordingly, the factors namely SW Defects, Dependencies, Architecture, SW Design, Adaption to change, Human Behavior, Team, and Integration of Linkage considered as both i.e. internal and external factors.

Evaluator 2 suggested that to re-consider the naming convention of identified factors. As per directions of the worthy experts of the domain the factors naming convention rectified with implementations as Module Dependencies to 'Dependency', Wrong Organizational Choice to 'Organizational Leadership, Adaption Strategies and Policies to 'Adaption to Change', Code to 'Source Code', Lack of Verification to 'Requirement', Team Cohesion to 'Team', Lack of Explicit Linkage to 'Integration of Linkage' Complexity Concerns to 'Architectural Complexity', Quality Assurance Concerns to 'Quality Assurance', Maintenance to 'SQW Maintenance', SW Artefacts to 'Artefacts', Security Concerns to 'Security' and Usage to 'Integration of Usage'.

Evaluator 3 suggested that to merge the two identified factors i.e. 'Requirement' and 'Volatility' as a single factor. As per directions of the worthy Expert of the domain the interlinked two factors namely 'Requirement' placed at serial No 20. and factor namely

‘Volatility’ placed at serial No. 25 of the Expert Evaluation Form has been merged and considered as single factors i.e. ‘Requirement Volatility’.

Evaluator 4 suggested that there is a need to refine the data units or sub-categories factors. In the same context, suggested combining a few data units or sub-factors in terms of their scope and meaning or re-consider them as a single one. As per directions of the worthy Expert of the domain, the data unit namely stakeholder goal and stakeholder objective combined as a single sub-factor, as ‘stakeholder goal and objectives’. Besides this, as poor communication and lack of communication have the same methodologies, therefore, this sub-factors or data units refined this level and treated it as ‘Poor Communication’, only. Accordingly, as the two sub-factors namely coordination and interaction mechanism have the same worth, therefore, as per directions of the expert of the domain refined and considered as a ‘Coordination Mechanism’. In the same context, Dependency on modules and dependency b/w components are treated as a single one. Moreover, as per observations of the worthy expert in the domain, the two data units have been removed.

In light of the above and to consider the given suggestions of worthy experts in the domain, the final list of identified factors is tabulated below.

Table 4.8: After implementation of the Expert's Suggestions the Final list of factors along with their category type and sub-factors

Sr No.	Paper ID	Sub Factors/Data Units	Factor (s)
1.	W1, W7, A1, A4	Higher Defect Density	SW Defects
		Defect Proneness	
		SW failure logs	
		Error Handling	
		Pre-release failure	
		Post-release failures	
2.	I6, W1, W4, W8, W18	Cost overrun	Resource Management
		Resource Estimation	
		Time and Resource Management	
		Budget Constraints	
		Schedule Issues	
		Project Size	
3.	I19, I21, W8, W14, W15, SD3, SD11, SD16, SD29, SD31, SD33	Decision Knowledge	Knowledge
		Decision Issues	
4.	I5, I12, I14, W1, W16, SD33, A1	Poor Communication	Communication Issues
		User-Centric Communication	
		Coordination Mechanism	
		Communication Gap	

		Message Exchange	
		Information Distortion	
5.	I6, I10, I13, W1, W9, A1	External Dependencies	Dependencies
		Change Dependencies	
		Dependency on modules	
		Requirement Dependencies	
		Data dependencies	
		Architectural Dependencies	
		Task Dependencies	
6.	I6, I19, W14, W17, W19, SD4, SD17, SD24, A4	Inability to trace a design	Traceability
		Tracing patterns	
		Design rationale Traceability	
		Traceability Links	
		Tracing Inconsistencies	
		Tracing the architectural Implementation	
7.	I14, A1	Dynamic Business Environment	Dynamic Business Environment
8.	I13, I15, I16, I17, W4, W18, SD12, SD29, A6	Stakeholder Synchronization	Stakeholder
		Stakeholder Goal & Objectives	
		Stakeholder Involvement	
9.	I1, I2, I3,I8, I9, I11, I16, I19, W1, W2, W5, W6, W15, W18, SD1, SD4, SD6, SD7, SD8, SD9, SD10, SD11, SD12, SD14, SD15, SD16, SD18, SD19, SD20, SD22, SD23, SD26, SD27, SD28, SD31, SD32, SD33, A2, A8	Architectural Knowledge	Architecture
		Architectural Decision	
		Architectural Integration	
		Architectural Assumptions	
		Architectural Erosion	
		Architectural Styles	
		Architectural Specification	
		Architectural crosscutting concern	
10.	I1, I3, I4,I5, I8,I11, I13, I15, I18, I21, I22, W10, W12, W17, W5, SD5, SD13, SD20, SD28, A1, A2, A4, A5, A6	Design Decision	SW Design & Design Implementation
		Design Patterns	
		Design Issues	
11.	I10	Wrong Organizational Choice	Organizational Leadership
		Basic Competency	
12.	I11, W10, W16, SD6, SD23	Adaption strategies and policies	Adaption to Change
		Adaption Flexibility	
		Strategy Change	
		On-Demand Adaption	
13.	I1, W13, W19, A3	Maintenance Prediction	SQW Maintenance
		SW Maintenance	
14.	I7, A6, A8	SW artifacts	Artifacts
		Design Artefacts	

		Architecture Req. Artifacts	
		Artifacts documents management	
15.	W14	Integration of Usage	Integration of Usage
		Utilization of Usage	
16.	I12, I19, SD29, A2	Trade-off Analysis	Trade-off
		Architectural Trade-off	
17.	I13, W9, W11, W12, SD2, SD26, A3	Code Smell	Code
		Coherent sets of code	
		Code Issues	
18.	SD1	Architectural Technical Debt	Technical Debt.
		Technical Debt Design	
19.	SD7, SD9, A6	Human Behavior	Human Behaviour
		Human Cognitive Constraints	
20.	I2, I5, SD13	Team Cohesion	Team
		Developer Focus	
		Effective Collaboration	
21.	I12	Lack of Explicit Linkage	Integration of Linkage
22.	I6, W1, W4, W18,SD27, A5	Architectural Documentation	Documentation
		Poor Documentation	
		Low-Quality Documentation	
23.	I21, W4, W13, SD2, SD21, SD25, SD29, A1,	Increased Complexity	Architectural Complexity
		Architectural Complexity	
24.	I3, I8, I9, I13, I16, I17, I18, W1, W2, W3,W9, W11, SD08, SD10, SD11, SD14, SD22, SD30	Ambiguous Requirement	Requirement Volatility
		Awareness of Requirement Volatility	
		High Level of Evaluability	
		Changing User Needs	
		Tracing the requirements	
		Requirement Specification	
		Non-Functional Requirements	
		Unnecessary changes	
		Anticipated changes	
		Changing to code	
		Knowledge of Initial Changes	
		Scope Change	
		Lack of Verification	
		Emotional and Relational Problems	
		Lack of Clarity in Business Objectives	
25.	I3, I11	Quality Assurance Concerns	Quality Assurance
		Maintaining Quality Attributes	
		Quality Attributes	
26.	SD17	Security	Security
27.	A7	Self-Healing Mechanism	Self-Healing Mechanism

The factor ‘SW Defects’ is based on the higher defect density, Defect proneness, SW failure logs, Error Handling, pre-release failure, and post-release failure sub-factors from the studies of W1, W7, A1, and A4. While, the factor ‘Resource Management’ is based on the

Cost overrun, Resource Estimation, Time and Resource Management, Budget Constraints, Schedule issues, and Project Size sub-factors from the studies of I6, W1, W4, W8, and W18. Moreover, the factor namely 'Knowledge' based on decision knowledge and decision issues sub-factors from the studies of I19, I21, W8, W14, W15, SD3, SD11, SD16, SD29, SD31, SD33. Accordingly, the factor 'Communication Issues' is based on poor communication, user-centric communication, coordination mechanism, communication gap, message exchange, and information distortion sub-factors from the studies of I5, I12, I14, W1, W16, SD33, and A1. In the same context, factor 'Dependencies' based on external dependencies, change dependencies, dependency on modules, requirement dependencies, data dependencies, architectural dependencies, and task dependencies sub-factors from the studies of I6, I10, I13, W1, W9, A1.

The factor 'Traceability' is based on the inability to trace design, tracing patterns, design rationale traceability, traceability links, tracing inconsistencies, and tracing the architectural implementations sub-factors from the studies of I6, I19, W14, W17, W19, SD4, SD17, SD24, A4. Besides this the factor namely 'Dynamic Business Environment generated from the studies I14, A1. The factor namely 'Stakeholder' based on stakeholder synchronization, stakeholder goals, objectives, and stakeholder involvement sub-factors from the studies I13, I15, I16, I17, W4, W18, SD12, SD29, and A6. On the hand, the most important factor namely 'Architecture' based on Architectural knowledge, architectural decision, architectural integration, architectural assumptions, Architectural erosion, architectural styles, architectural specification, and architectural crosscutting concern sub-factors from the studies I1, I2, I3, I8, I9, I11, I16, I19, W1, W2, W5, W6, W15, W18, SD1, SD4, SD6, SD7, SD8, SD9, SD10, SD11, SD12, SD14, SD15, SD16, SD18, SD19, SD20, SD22, SD23, SD26, SD27, SD28, SD31, SD32, SD33, A2, A8. In foregoing of this, the factor namely 'SW Design and Design Implementations' based on a design decision, design patterns, and design issues from the studies I1, I3, I4, I5, I8, I11, I13, I15, I18, I21, I22, W10, W12, W17, W5, SD5, SD13, SD20, SD28, A1, A2, A4, A5, A6.

Moreover, the factor 'Organizational Leadership' is based on wrong organizational choice and basic competency sub-factors from the study I10. The factor namely 'Adaption to Change' based on adaption strategies and policies, adaption flexibility, strategy change, and on-demand adaption sub-factors from the studies I11, W10, W16, SD6, SD23. While, the factor namely 'SQW Maintenance' based on maintenance prediction and SW maintenance from the studies I1, W13, W19, and A3. The factor 'Artefacts' is based on SW artifacts, Design artifacts, architecture requirement artifacts, and artifacts documents management sub-factors from the studies I7, A6, and A8. The factor 'Integration of Usage' is based on the

Integration of usage and utilization of usage from the study W14. The factor ‘Trade-Off’ is based on trade-off analysis and architectural trade-off from the studies I12, I19, SD29, and A2. The factor namely ‘code’ based on code smell, coherent sets of code, and code issues sub-factors from the studies I13, W9, W11, W12, SD2, SD26, and A3. The factor ‘Technical Debt.' is based on architectural technical debt. and technical debt design from the study SD1.

The factor ‘Human Behavior’ is based on human cognitive constraints and behavior sub-factors from the studies SD7, SD9, and A6. The factor namely ‘Team’ based on team cohesion and developer focus from the studies I2, I5, and SD13. The factor ‘Integration of Linkage’ is based on the Lack of explicit linkage sub-factor from the study I12. The factor namely ‘Documentation’ architectural documentation, poor documentation, and low-quality documentation sub-factors from the studies I6, W1, W4, W18, SD27, and A5. The factor ‘Architectural Complexity’ is based on increased complexity and architectural complexity from the studies I21, W4, W13, SD2, SD21, SD25, SD29, A1, and A3. Besides this, another important factor namely ‘Requirement Volatility’ based on the ambiguous requirement, awareness of requirement volatility, high level of evaluability, changing user needs, tracing the requirements, requirement specification, non-functional requirements, unnecessary changes, anticipated changes, changing to code, knowledge of initial changes, scope change, lack of verification, emotional and relational problem and lack of clarity in business objective sub-factors from the studies I3, I8, I9, I13, I16, I17, I18, W1, W2, W3, W9, W11, SD08, SD10, SD11, SD14, SD22, SD30. Moreover, the factor namely, ‘Quality Assurance based on the quality assurance concerns, maintaining quality attributes or quality attributes sub-factors from the studies I3, I11. Accordingly, the factors namely ‘Security’ and ‘Self-Healing Mechanism derived from the studies SD17 and A7, respectively.

To answer the research question RQ1, Table 4.9 tabulated below indicated all possible factors of the requirements volatility on software architecture. Wherein it, the table is comprised of twenty-seven attributes horizontally which contain the complete information in terms of factors, and eighty-three tuples vertically which contain the complete data in terms of the studies or paper IDs tilted as I1-I22, SD1- SD33, W1-W19, and A1-A8.

Table 4.9 Tabulated representation of RV Factors related to the SW Architecture.

Paper ID's / Factors	Software Defects	Resource Management	Knowledge	Communication Issue	Dependencies	Traceability	Dynamic Business Environment	Stakeholder	Architecture	SW Design	Organizational Leadership	Adaptation to Change	SQW Maintenance	Artifacts	Integration of Usage	Trade-off	Code	Technical Debt	Human Behavior	Team	Integration of Linkage	Documentation	Architectural Complexity	Requirement Volatility	Quality Assurance	Security	Self-Healing Mechanism
I1	-	-	-	-	-	-	-	-	✓	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-
I2	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-
I3	-	-	-	-	-	-	-	-	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	✓	-	-
I4	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
I5	-	-	-	✓	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-
I6	-	✓	-	-	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-
I7	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-
I8	-	-	-	-	-	-	-	-	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-
I9	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-
I10	-	-	-	-	✓	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
I11	-	-	-	-	-	-	-	-	✓	✓	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-
I12	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	✓	-	-	-	-	-	-
I13	-	-	-	-	✓	-	-	✓	-	✓	-	-	-	-	-	-	✓	-	-	-	-	-	-	✓	-	-	-
I14	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

I15	-	-	-	-	-	-	-	✓	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
I16	-	-	-	-	-	-	-	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-
I17	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-
I18	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-
I19	-	-	✓	-	-	✓	-	-	✓	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-
I20	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
I21	-	-	✓	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-
I22	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SD1	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-
SD2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	✓	-	-	-	-
SD3	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SD4	-	-	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SD5	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SD6	-	-	-	-	-	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SD7	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-
SD8	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-
SD9	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-
SD10	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-
SD11	-	✓	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-
SD12	-	-	-	-	-	-	-	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SD13	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-
SD14	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-
SD15	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SD16	-	-	✓	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SD17	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	
SD18	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SD19	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SD20	-	-	-	-	-	-	-	-	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

SD21	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-
SD22	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-
SD23	-	-	-	-	-	-	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SD24	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SD25	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-
SD26	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-
SD27	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-
SD28	-	-	-	-	-	-	-	-	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SD29	-	-	✓	-	-	-	-	✓	-	-	-	-	-	-	-	✓	-	-	-	-	-	✓	-	-	-	-
SD30	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-
SD31	-	-	✓	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SD32	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SD33	-	-	✓	✓	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
W1	✓	✓	-	✓	✓	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	✓	-	✓	-	-	-
W2	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-
W3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-
W4	-	✓	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	✓	✓	-	-	-	-
W5	-	-	-	-	-	-	-	-	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
W6	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
W7	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
W8	-	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
W9	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	✓	-	-	-
W10	-	-	-	-	-	-	-	-	-	✓	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-
W11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	✓	-	-	-
W12	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-
W13	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	✓	-	-	-	-
W14	-	-	✓	-	-	✓	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-
W15	-	-	✓	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

W16	-	-	-	✓	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
W17	-	-	-	-	-	✓	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
W18	-	✓	-	-	-	-	-	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-
W19	-	-	-	-	-	✓	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A1	✓	-	-	✓	✓	-	✓	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-
A2	-	-	-	-	-	-	-	-	✓	✓	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-
A3	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	✓	-	-	-	-	-	✓	-	-	-	-
A4	✓	-	-	-	-	✓	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A5	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	✓	-	-	-	-	-
A6	-	-	-	-	-	-	-	✓	-	✓	-	-	-	✓	-	-	-	-	✓	-	-	-	-	-	-	-	-
A7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓
A8	-	-	-	-	-	-	-	-	✓	-	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-

The symbol "✓" indicated the occurrence of the factor included in the respective studies while the symbol "-" indicated the non-occurrence of the factor against each study. Accordingly, the factor 'SW Defects' occurred in W1, W7 studies of the Willey online Library and A1, A4 studies from the ACM digital library. The factor namely, 'Resource Management' occurred in the I6 study of IEEE Explorer, W1, W4, W8, and W18 studies of the Wiley online library. The factor namely 'Knowledge' occurred in the I19, I21 studies from the IEEE, W8, W14, and W15 studies from the Wiley online library and SD3, SD11, SD16, SD29, SD31, SD33 studies from the Science Direct. Factor 'Communication Issues' occurred in I5, I12, and I14 studies from the IEEE, SD33 from the Science Direct and W1, W16 studies from the Willey, and A1 from the ACM Digital Library. The factor namely 'Dependencies' I6, I10, and I13 from the studies IEEE, W1, and W9 studies from the Wiley online library and A1 from the ACM digital library. "Traceability' occurred in I6, I19 studies from the IEEE, SD4, SD17, SD24 studies from the Science Direct, W14, W17, and W19 studies from Willey, and A4 studies from the ACM. The factor namely 'Dynamic Business Environment' occurred in the I14 from the studies IEEE and A1 study of the ACM.

The factor 'Stakeholder' occurred in the I13, I15, I16, and I17 studies from the IEEE, SD12, SD29 studies of the science direct, W4, W18 studies from the Willey and in A6 study of the ACM Digital Library and A6 study of the ACM Digital library. The Factor namely 'Architecture' occurred in the I1, I2, I3,I8, I9, I11, I16, I19 studies form the IEEE, SD1, SD4, SD6, SD7, SD8, SD9, SD10, SD11, SD12, SD14, SD15, SD16, SD18, SD19, SD20, SD22, SD23, SD26, SD27, SD28, SD31, SD32, SD33 studies from the Science Direct, W1, W2, W5, W6, W15, W18 studies from the Willey and A2, A8 studies from the ACM Digital Library. The factor namely 'SW Design' found in the I1, I3, I4,I5, I8,I11, I13, I15, I18, I21, I22 studies from the IEEE, SD5, SD13, SD20, SD28 studies from the Science Direct, W10, W12, W17 studies from the Willey and A1, A2, A4, A5, A6 studies from the ACM digital library. The factor 'Organizational Leadership' occurred in the I10 study from the IEEE. The Factor 'Adaption to Change' was found in the I11 study from the IEEE, SD6, SD23 studies from Science Direct, and W10, and W16 studies from the Willey. The factor 'SQW Maintenance' is found in the I1 from the IEEE, W13, and W19 studies from the Willey and A3 study from the ACM Digital Library. The factor 'Artefacts' is found in I7 from the study IEEE and in the A6, and A8 studies from the ACM. The factor 'Integration of Usage' in the W14 study from Willey. The factor namely 'Trade-off' found in I12, I19 studies from IEEE, SD29 study from Science Direct, and A2 study from the ACM. The factor 'Code' found in the I13 study from the IEEE, SD2, SD26 studies from the Science Direct, W9, W11,

and W12 studies from the Willey and A3 study from the ACM. The factor namely ‘Technical Debt’ found in study SD1.

The factor namely ‘Human Behavior’ occurred in the SD7 and SD9 studies from the Science Direct and A6 study from the ACM Digital Library. The factor ‘Team’ was found in I2 and I5 studies from the IEEE and SD13 studies from Science Direct. The Factor ‘Integration of Linkage’ was found in the W14 study from Willey Online Library. The factor ‘Documentation’ found in the I6 study from IEEE, the SD27 study from Science Direct, W1, W4, and W18 studies from the Willey Online Library, and the A5 study from ACM Digital Library. The factor ‘Architectural Complexity’ found in the I21 study from IEEE, SD2, SD21, SD25, and SD29 studies from Science Direct, W4, W13 from the studies of Willey, and A3 from the ACM. The factor ‘Requirement Volatility’ found in the I3, I8, I9, I13, I16, I17, I18 studies from IEEE, SD08, SD10, SD11, SD14, SD22, SD30 studies from the Science Direct and W1, W2, W3,W9, W11 studies from the Willey. The factor namely ‘Quality Assurance’ found in I3, and I11 studies from the IEEE. The factor ‘Security’ is found in the SD17 from Science Direct. In the last, the factor namely ‘Self-Healing Mechanism’ found in the A7 study from the ACM Digital Library.

4.5 Description of the Identified Factors

4.5.1 Software Defects

The software team members and stakeholders perceived the ‘SW Defect’ as a fault and a wrong attempt. However, this is an important factor of the software development process which is most specifically used by the testers or quality-concerned ones. Whereby, the testers considered it by reading the ‘requirement document’, which is the core element of this factor, to dig out defects related to incorrect behavior or anything that is not good. The core purpose of this exercise is to determine the meeting of project requirements to fulfill customer satisfaction. Hence, through the use of this factor, the testers reported the upcoming changes or occurrence of requirement volatility to the developers. In the same context, when requirement volatility occurs it changes the previous architecture decision redundant. As a result, the architects have to put effort to make a new round of architectural prototypes. Therefore, it is important to consider the software defect timely; it will concurrently help out the developers and engineers to adopt the changes throughout the development process. Hence, there is a necessity to consider this factor, to avoid the pre-release and post-release failures of the project [1][82].

4.5.2 Resource Management

As, the requirements volatility could occur throughout the software development process, as a result, architects take time to update the architecture decision to the re-designing purpose of the prototype. This causes challenges, in terms of the project size, time and schedule issues, and budget constraints matters. Therefore, there is necessary to consider resource management effectively. To cope with these challenges, the management has to prepare development roaster plans with the quarterly releases of the project. As a result, the developers and architects could analyze the project resources and could be able to meet the upcoming changes in the document and architects will be able to handle the re-planning and architectural decision issues and will also be able to update the backlog list for smooth development [1][28].

4.5.3 Knowledge

Meeting with the upcoming changes or volatility thought out software development is a challenging activity. But, it could be achieved through the phenomenon of knowledge. In the same context, the problem of the existing software architecture that could adversely affect the architecture decision is an important element to consider with the knowledge. This factor plays a vital role in development if developers or engineers have sound integration of knowledge that from where to start the changes and how it could impact the software architecture. Hence, this is an important factor that needed to consider on time with experience and by generating fruitful development strategies [20][21].

4.5.4 Communication Issues

Communication issues may originate during the elicitation phase, where the customer elaborates on the upcoming desired product. This is normally occurring due to the wrong use of terminologies and semantics between the customer and the developers. Moreover, language barriers and cultural differences are other issues that lead to communication issues. There is a dire need to cope with factors by using supporting tools and well-developed approaches to improve these factors for successful development or to meet the desired end product [1]. To deal with this phenomenon of twin peaks of the SLDC, there is an essential need to consider this factor more precisely, to cope with the upcoming changes [34].

4.5.5 Dependencies

The developers are working in a dynamic platform along with different business lines and geographic locations. As a result, they have to deal with some external as well as internal dependencies. For example, for giving the solution of the end product, developers may require collaboration among team members from different business lines. Where one may deal with the coding and another may deal with the service-providing related matters. In this context, if requirement volatility occurs, they are dependent on each other which are known as external dependencies and different business lines engineers are dependent on each other to adopt the changes. Moreover, no doubt, requirement volatility is a challenging activity but developers could easily achieve it if they are familiar with the used modules along with their dependencies and their strong relationship that changes one how much effect on another. As a result, developers easily adopt upcoming changes, which are known as internal dependencies. To deal with the upcoming changes, there is an essential need to consider these dependencies, effectively [1] [21].

4.5.6 Traceability

To implement the explicit or implicit volatility the architects have to deal with the architectural decisions for re-designing the purpose of the prototype. It could be achieved by tracing back the requirement changes throughout the development process. This phenomenon of traceback is known as traceability. Moreover, it is another important factor to consider by the architects or developers to deal with the upcoming changes. To consider traceability, the architects record the requirements or design while making an architectural decision. As a result, they can easily trace back the decisions in case of re-designing the architecture. But, Requirement volatility could adversely affect this recording method or may cause redundancy or muddle the information that is already recorded. To cope with this issue, there is an essential need to deal with this factor of traceability to trace back the architectural decisions more, effectively [1] [40].

4.5.7 Dynamic Business Environment

Now a day, IT parks are operating in a dynamic business domain and adjusting new strategies to accommodate updated or new development techniques for being part of the market competition. As a result, the market leads the situation being more challenging to adopt the upcoming changes or requirement volatility. There is an essential need to consider this factor, where, as most of the customers are working on android-based applications, in a

result, their operating system must be enough dynamic to consider upcoming changes, frequently [1].

4.5.8 Stakeholder

Stakeholder concern is one of the major causes of requirement volatility. It is a fundamental activity that occurs throughout the software development process. As a result, architects also have to adopt the upcoming changes by revisiting their architectural decisions. Moreover, the architects are not only the core person to deal with the architectural changes the different stakeholder concerns are also involved in this process. Therefore, for successful deployment, there is an essential need to consider stakeholder synchronization. This problem occurs due to interunit issues, where the team is in the same business unit but at different development sites. To cope with this challenge the architects needed to use effective tools to overcome the raised inconsistency or to meet the requirement volatility concerning stakeholder involvement [1] [45].

4.5.9 Architecture

Software architecture represents the complete vision of the software system that could adversely affect by the requirement volatility. Whenever it occurs, the architects have to reconsider the architectural decision to reshape the architectural design according to the customer's need or want. As a result, architects also have to reconsider the product backlog. This event could raise redundancy and inconsistency at the architectural implementation ends. There is an essential need to consider this factor more precisely and the developers must get their requirements in all aspects during the elicitation phase, first. Moreover, as the modern iterative models such as agile consider it throughout the SDLC, therefore, firms needed to make a consolidated plan in respect of the upcoming changes along with the project release matters and timeframe issues. As a result, the architects will be able to manage the architectural decisions to cope with the upcoming changes [34].

4.5.10 SW Design and Design Implementation

Software designs are becoming more revolutionary during the software development process (SDP), due to the adoption of frequent changes in stakeholder requirements. Therefore, the developers have to consider different platforms at once such as handling the errors or bugs, adding up the new functionalities or features in design implementation, or deletion of some previous ones. Handlings of these changes prevent the

degradation of SE design or code. This challenge could be detected through ‘software smells’ or ‘code smells. Usually, it occurs during the code or design levels. However, the existence of code smell also produces low-quality attributes such as changeability. To fix this, developers have to use the ‘refactoring’ process during SW design. This process alters the SW implementations and design without changing its external behavior. As a result, it will enhance the design implementations or SW design along with the well-structured programs. Therefore, there is an essential need to consider this factor to cope with the upcoming changes in the SW Design [22]

4.5.11 Organizational Leadership

Organizational leadership is an important factor, where, the firms are planning, controlling, managing support, informing, and evaluating the team members for the successful completion of the desired product. The leadership and management of the firms are overlapping each other to meet the desired goals through effective communication and teamwork. The strong relationship between these two pillars of the organizations or firms indicates their competency level. To meet the stakeholder requirements there is an essential need to consider this factor of organizational leadership, whereby, the management of the organization needed to play a vital role to train the staff along with the performance essentials, get familiarize them with the user's culture and motivate them to consider emotional intelligence during dealing with them [37].

4.5.12 Adaption to Change

Modern iterative systems are dealing with changes throughout the software development life cycle along with the customer collaboration or interaction mechanism. Therefore, system designers are becoming more experienced in system adaptability. Where an adaptive system can modify its system design during run time through adaption to change. As a result, the system designer requires knowledge about the adaption to change e.g. whether these changes are system-driven or user-driven, and analyze their impact on the system. To cope with the requirement volatility, there is an essential need to consider this factor and train SW designers along with experience that how to handle the adaption to change mechanism during a run time [72].

4.5.13 SQW Maintenance

The primary objective of SQW maintenance is to make the software system operates according to the stakeholder's needs and fix the bugs in the system. Software maintenance usually occurs when it has been delivered to the end user. Besides this, SW maintenance may involve where the new functionality is needed to be added e.g. using the latest technology. Moreover, it also may require maintenance on the software code segments, whereby, a few software patches are needed to execute to fix the bugs in the document. Accordingly, it could also lead to affected by cost constraint issues. To deal with the requirement volatility and software architecture there is an essential need to consider this factor and train the engineers to predict the software maintenance timely to fix the errors in the code segment and try to meet the user level satisfaction before the deployment.

4.5.14 Artifacts

Software artifacts are the key features of the software development process which are used for manufacturing the software architecture and system design. In the same context, artifacts included diagrams, architecture designs or images, meeting notes, documentation papers, source code, and prototypes. Therefore, artifacts act like a roadmap that helps the developers to trace the design implementations. Moreover, the software artifacts are usually created during the development phases and restored in the repository document. Whenever the changes occur engineers re-visit the repository to re-designing the purpose of the prototype and architectural decisions to adopt the changes. Besides this, many of the artifacts are sued as safety evidence. As a result, the upcoming changes could be easily managed and it could also help to better assist in the impact analysis related to the consequences of changes in the end product.

4.5.15 Integration of Usage

The integration of usage is the prominent factor that is used by the practitioners and which is reported about the software tools that are available in the company. In the same context, this factor deal with four different usages, whereby, the first one is about ‘support of external usage knowledge system’ which is used for getting feedback, the second one is about ‘integrated development’, which is used to ensure the functionality of the system developed by the developers in the code, the third one is about ‘support for interfaces’ which are used to ensure the hardware related changes and the fourth one is about, ‘full support for any kind of project management tool’, which is used to ensure the stakeholder's acceptance by focusing

on their usages. To cope with the upcoming changes or requirement volatility, there is an essential need to consider this factor as a priority [21].

4.5.16 Trade-off

The trade-off is another visible factor that is used for risk mitigation purposes during the software development life cycle. As a result, the engineers meet the quality attributes of the development process and enable the upcoming changes in the software structure, design, or code. To achieve the goals of the end product there is a dire need to consider trade-off analysis and sensitivity along the way so the desired product can be built correctly [32].

4.5.17 Code

The need of managing the requirements volatility in the software development process has been raised as one of the challenging tasks of software development. Whereby, the developers have to adopt the changes frequently throughout the development process. Besides this, during this process architecture appears as an integral unit that could also adversely affect by the volatility. Moreover, the architecture must have to translate in the shape of ‘code'. Therefore, one wrong attempt between architecture and code could lead to software failures in terms of the project's financial implications as well as customer expectations. To cope with upcoming changes there is an essential need to consider code implementations as a priority [79].

4.5.18 Technical Debt

The engineers usually prioritize the requirements in respect of customer needs over architectural considerations. However, the architects have to consider the requirements with architectural aspects; this phenomenon is known as, ‘Technical Debt’. As architects are also flooded with requirement volatility, in a result, they lost their vision to find out optimal architecture design implementations. Moreover, prioritizing the functional requirement over the non-functional requirement is a major cause of technical debt. Because most of the non-functional requirements are architectural significant requirements and they are also needed to consider for smooth architectural building. Therefore, there is a dire need to consider technical debt. on priority to cope with the upcoming changes along with accurate architectural implementations [1].

4.5.19 Human Behavior

Software development activity is human role activity and behavior of involved humans having great importance for the software development community. The involved human behavior is usually associated at three different levels i.e. industrial behavior, team behavior, and organizational behavior. Whereby, the industrial and team behavior rectifies the failures of the existing tools in the system. While, in software architecture decisions humans also play a vital role, in terms of architects or stakeholders. As a result, human behavior factors have a huge impact on software development. Therefore, there is a dire need to consider this factor and the top-level management has to use authentic strategies to make their developers interconnected, on the same dashboard of the development along with the refined set of goals and objectives among the stakeholders and team members [56].

4.5.20 Team

The software development team works together for developing the end products. During development, every person contributed their work for the software development. However, the individual team members of the development team have specialized skills and domains for development but the accountability of work measures as a whole ’team’. To consider the upcoming change or volatility there is a dire need that the team members to be closely interconnected with each other through an exchange of messages or meetings. As a result, this factor contributes a significant positive impact on software development and change management [83].

4.5.21 Integration of Linkage

The adaptive system aims to consider the requirements volatility at runtime. As a result, the systems require re-configuration and the resources may require trade-off analysis between the functional and non-functional requirements. It is penitent to mention here that the architecture development may have interlinked with the non-functional requirement. However, developers usually prioritize the functional requirements over the non-functional. As a result, the lack of explicit linkage originates raise heavy surge in the failure rate of software development. Therefore, to cope with the upcoming change and architectural management there is an essential need to consider the integration of linkage which consider or prioritized the non-functional requirements to meet the user level satisfaction with confirmative architecture development [35].

4.5.22 Documentation

This is the most prominent factor of software development and it helps to preserve and share the knowledge about the system that is going to build. In the initial phase freeze requirements are placed in this document for the development purpose of the product. It acts like a repository of customer requirements in a documented form, which is used by the developers for developing the system. But, it becomes difficult to maintain when the requirement is required frequently changed in the document. To cope with the upcoming changes and architectural re-designing there is a dire need to consider this factor as a priority and try to keep refreshing this document with the updated changes in the requirements till the end of the product [1] [15].

4.5.23 Architectural Complexity

Architectural complexity increase when the future requirements are not clear. To reduce the complexity there are normally two different ways used to design the architecture. Whereby, the first option is to build a full-scale initial architecture that is flexible enough to consider future requirements. During this phase, the frequency of architectural complexity seems to be high to manage and it also requires intensive time and other resources to implement the architecture. As a result, it helps to consider future changes easily. The second option is to consider the architecture simple and evolve it as time progress and new requirements whenever received. In the same context, most of the developers used this option because this method is aligned with the agile implementations which are most widely used by the developers to meet the requirements volatility. There is a dire need to consider this factor more wisely because new systems are working on iterative development or dealing with the changes throughout the software development process [1] [80].

4.5.24 Requirement Volatility

Requirement volatility is a fundamental activity that is required throughout the software development life cycle. As a result, it could adversely affect the software architecture which intimates the complete vision of the desired product. Moreover, it is a challenging activity, but it could be achieved through the integration of usage and decision knowledge [21]. This study revealed that communication issues and dependencies are the core factor which is becoming the cause of requirements volatility and the factors related to the architecture i.e. traceability, SW design, design implementations, and architectural complexity

are the main factors that have the major implications of requirements volatility on the software architecture. Therefore, practitioners have to consider this factor as a key priority for the smooth adaption of future changes during development phases [1] [5].

4.5.25 Quality Assurance

There are a lot of variations that exist to consider quality assurance during the software development process. The practitioner faces many challenges related to integrating quality attributes. Besides this, during the development process, developers have to consider the implicit as well as explicit requirements. Which may cause inconsistency in the management of the quality attributes of the end product. However, most of the time software team is considering the requirements on their own and defines priorities, to ensure and integrate quality assurance. This phenomenon leads to an increase in the complexity at the quality ends, especially, when requirements volatility existence is higher. Therefore, upcoming changes could potentially impact quality assurance. The practitioners have to consider this factor more precisely and try to meet the quality attribute rather than typically assigning the priorities to specific retirements. As a result, the development team will be able to achieve the quality attributes of the end product [1] [49].

4.5.26 Security

Software security is a visible factor in the software development process. Whereby, different techniques or applications are used to protect the end product from vulnerabilities. Which also ensures the smooth working of the system and prevents it from attacks. The major goal of this factor is to prevent the product from failure or defects. Moreover, this factor also measures the extra-functional properties of the product along with the security element i.e. performance and reliability. To cope with the upcoming changes there is an essential need to consider the security functions along with the updated requirements and their implications on the product [40].

4.5.27 Self-Healing Mechanism

Self-healing mechanism indicated bout the self-adaption which can be achieved through different ways. For this, rule-based approaches considered the phenomenon of self-adaption if the system and environment meet certain conditions. However, utility-driven approaches are used to adapt the optimal decisions through cost optimization that could run on large-scale problems. In the same context, the self-healing mechanism adopts both rule-based

approaches and utility-driven approaches, through expensive optimization. Moreover, Architects used this scheme for the construction of architecture-based self-healing problems, whereby, the patterns are designed to accommodate the mechanism of self-healing for the architecture. To cope with the upcoming change and construction of the dynamic architectural environment, there is an essential need to consider this factor, precisely [82].

4.6 Summary of the Chapter

This chapter intimated the findings of the conducted, SLR, grounded theory, and Expert Review. Whereby, a list of factors identified by the conduct of SLR and in numbers refined list of twenty-seven (27) factors are identified through the conduct of grounded theory. In the end, the suggestions and recommendations are collected by worthy experts in the domain to rectify the proposed factors. After the implementation of the expert's recommendations and for conducting the empirical investigation, the final list of factors is rotated among the competent practitioners, engineers, and developers to get the positive implications of requirements volatility on the software architecture through the conduct of an industrial survey. In the same context, the forthcoming chapter will intimate the findings or results of the survey.

CHAPTER 5

INDUSTRIAL SURVEY

5.1 Introduction

The previous chapter contains the complete information related to the conduct of a Systematic Literature Review (SLR) along with their findings. Moreover, contains complete information related to the implementation of Grounded Theory and valuable conduct of the Expert Review. Preliminarily, in numbers twenty-eight (28) factors are identified via smooth conduct of SLR, during this phase the factors are also refined through the adoption of the data encoding technique of grounded theory. After the successful completion of this process, the refined list of factors was forwarded to the different experts in the domain for validation purposes. Whereby, the worthy experts of the domains rectified the identified list of factors and recommended their suggestions. To meet the core purpose of this chapter, the validated list of factors is shared with the practitioners and industrial people, for the smooth conduct of the empirical investigation through the conduct of an industrial survey about the positive implications of these identified factors on the software architecture.

5.2 Industrial Survey Findings

An industrial survey was conducted to get the positive implications of the identified list of factors on the software architecture. For the implementation of this, the guidelines of Mark Kasunic [24] were used for the smooth conduct of the industrial survey. However, the complete detail in terms of this conduct survey has already been elaborated on in chapter 3. Whereby, each step under the guidelines is reported towards implementation. As this study is most specifically about the twin peaks of the SDLC i.e. ‘Requirement Volatility’ and ‘Software Architecture’, therefore, to address the matter efficiently the industrial practitioners or developers of this domain selected for participation in the survey on account of the target audiences. In the very first step, the questionnaire was designed online through Google forms and shared with the targeted audiences in January 2022. The survey is based on three different sections, whereby section one contains the introductory part. While Section 2 is based on the demographic section, the core purpose of this section is to get the demographic details of each respondent in terms of their Name, Designation, Qualification, software development experience, Organizational information, email id, etc.

Besides this, this section also obtained some additional information related to their companies in terms of considered projects and names of projects most specifically to this study domain. Moreover, this part also obtained the respondent's company demographics information including company location, scope or level, staff, SPI certifications, and type of companies. In the end, this section also gets the information related to the respondent (s) companies care about the twin peaks of the SDLC i.e. 'Requirement volatility' and 'software architecture'. Then, section three of this survey is designed which contains the complete details of the identified final list of factors, which are already been passed out through the experts of the domain during the conduct of SLR. To achieve the goal, this list of factors is placed in this section of the survey along with their descriptions for getting the positive implications of each factor in the form of a 5 Likert Scale i.e. strongly agree to strongly disagree. In the same context, the complete designed questionnaire is placed herewith at the ends of the appendixes in Appendix-G.

5.2.1 Distribution of Respondent's Experiences based on Software Development

The graph shown below is representing the respondent's software development experience (in years) in current/ previous organizations. Whereby, 29.7% of respondents have 2 years of experience, and 20.9% of respondents have experience 3 years. While 19.8% of respondents have experience of 5 years. 9.9% of respondents have experience of 6 years. 8.8% of respondents have experience of 4 years. 3.3% of respondents have experience of 7 to 8 years. However, 2.2% of the respondent have experience of 10 years.

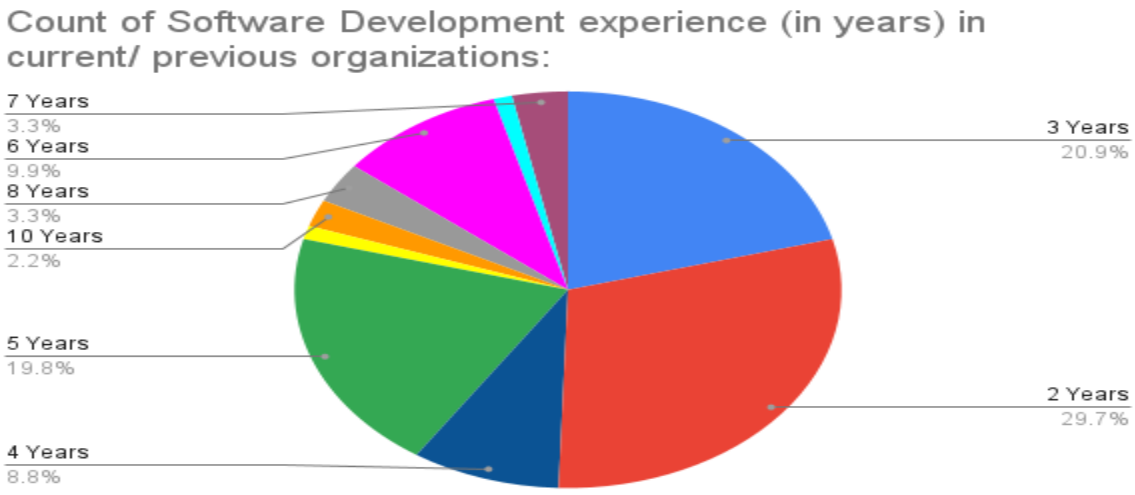


Figure 5.1 Distribution of responders based on Experiences

5.2.2 Distribution of companies based on the study domain

The below-drawn graph represented the distribution of companies based on the study domain i.e. about twin peaks of the SDLC. Whereby, 80.2 % of companies responded as 'Yes' in terms of considerations of the positive implications of requirements volatility on Software Architecture. However, 19.8% of companies responded as 'No'.

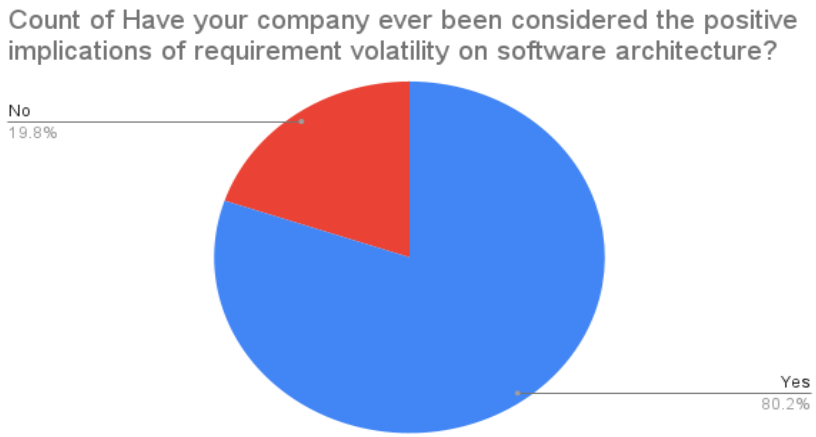


Figure 5.2 Distribution of Companies based on the study domain

5.2.3 Distribution of responses based on the Scope of the Company

The graph shown below represented the distribution of responses based on the scope of the companies. Whereby, 53.8 % of responders companies are working on the scope of 'National' level and 46.2% of companies are working on the level of 'Multi-National'.

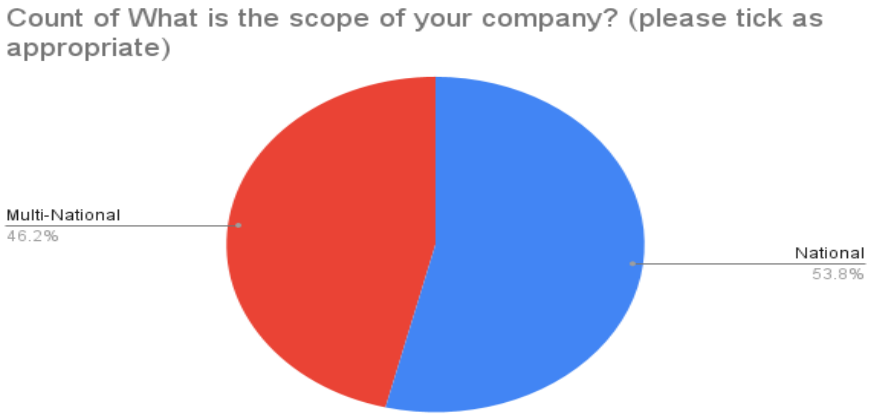


Figure 5.3 Distribution of the company's responses based on the scope

5.2.4 Distribution of companies based on the working strength

The graph shown below represented the company's or organization's strength related to the employed or working staff. Whereby, 44.0% of companies have a working strength of 20-100. While 31.9% of organizations have a working strength greater than 100. However, 24.2% of companies have less than 20 working strength on account of employed staff.

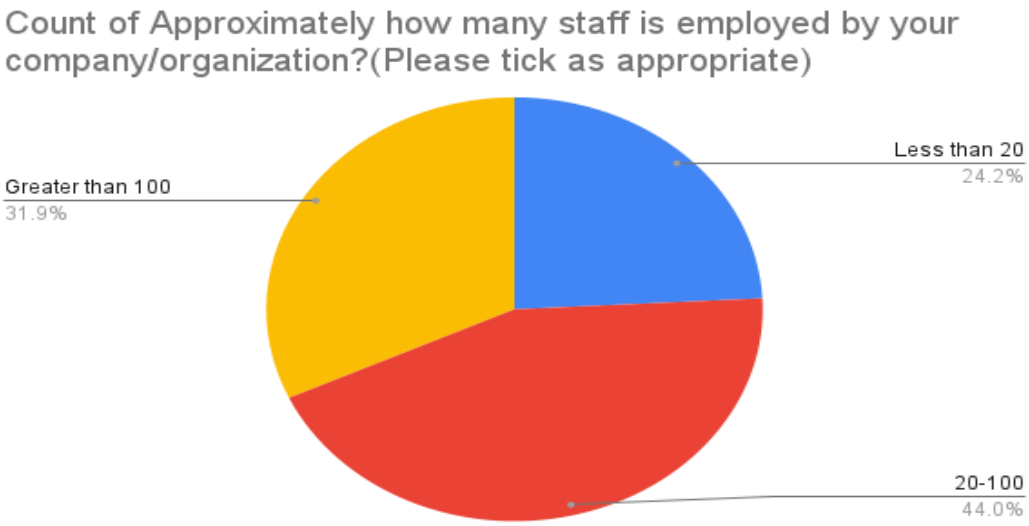


Figure 5.4 Distribution of companies based on the working strength**.**

5.2.5 Distribution of Companies based on SPI Certifications

The graph shown below represented the complete detail in terms of the respondent company/firm achieved SPI Certifications. Whereby, 59. 3% of respondent companies have CMMI certifications. While 40.7% of respondent companies have ISO Certifications.

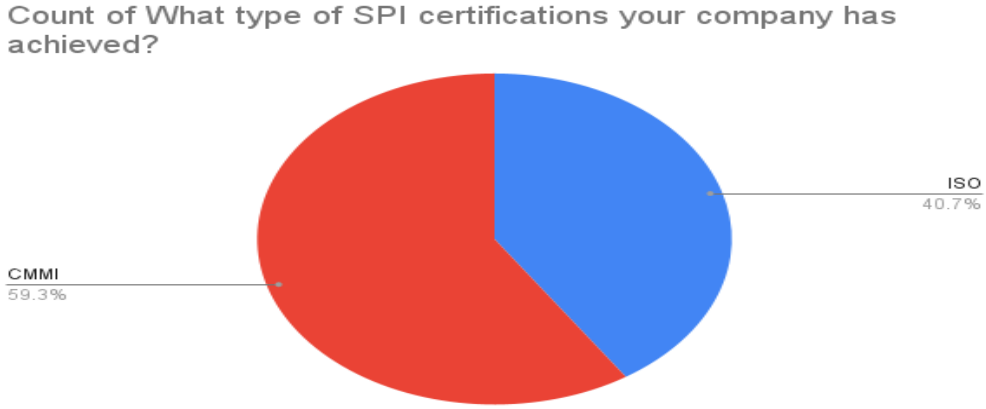


Figure 5.5 Distribution of company based on SPI Certifications

5.2.6 Distribution of respondent firms based on the type of development

The graph shown below represented the total count of respondent firms about their type of software development or major concern. Accordingly, 11 companies are concerned with related 'Database development', and 15 companies participating in 'real-time system' development. The 19 companies are working on the 'Web Application' development. While 6 firms have major concerns about the 'Games development'. Moreover, 10 firms have major concerns with 'Multimedia Software. Besides this, 5 firms are working on the 'Desktop application'. In the same context, 7 firms are working on the development of 'Graphic Designing' and another 7 on 'System Software. However, the 6 firms have major concerns with 'Mobile Development'. However, 2 firms are playing their major roles in the back-end development. While another 2 firms have major concerns with the 'Data Analysis'.

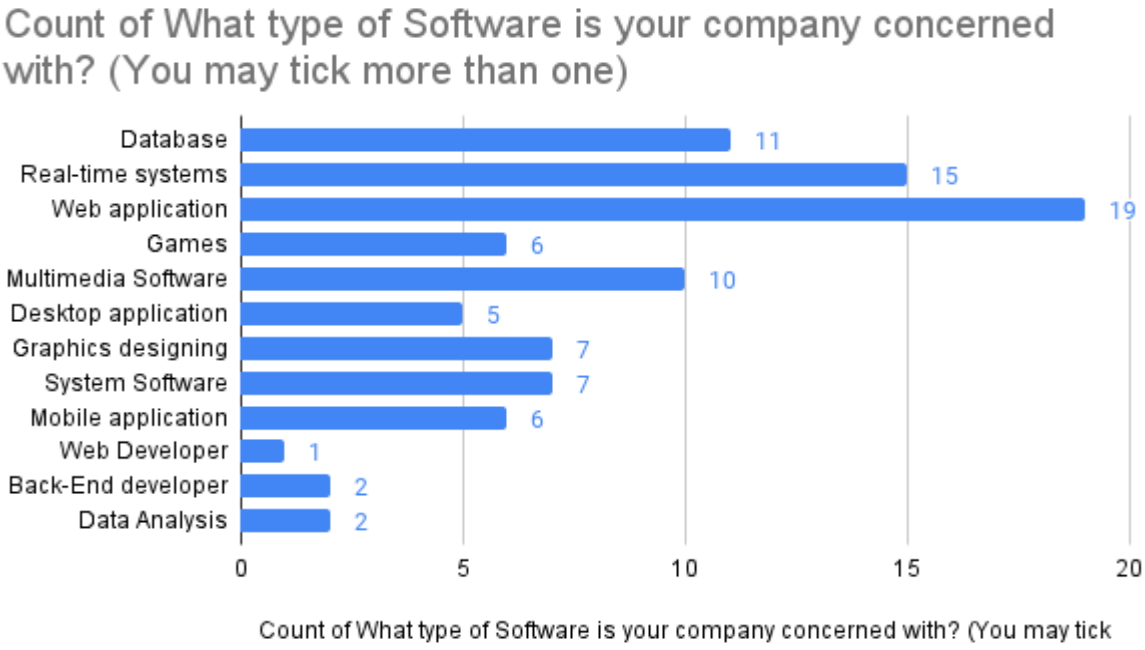


Figure 5.6 Distribution of respondent firms based on the type of development

5.3 Result Analysis and Reporting

To achieve the goal and proposed the solutions. This section of the study reported the results of the positive implications of Requirement Volatility factors on the Software Architecture. In the same context, the proposed results are based on the key points that how much this identified list of factors contributes to their major roles during the development process, more specifically in terms of their positive implications on the software architecture.

5.4 Positive Implications of Requirement Volatility Factors on the SW Architecture

As mentioned earlier, the core purpose of this conduct of industrial survey is to propose the positive implications of requirements volatility factors on the Software Architecture. For this results are generated on basis of the Likert scale score from 1 to 5 i.e. Strongly Agree 1, Agree 2, Neutral 3, Disagree 4, and Strongly Disagree 5.

5.4.1 Testing Results/Statistics

For analysis purposes, this study used JMP statistical tool. Whereby, the results are generated against the factors. Accordingly, this study focused to get information from practitioners and industrial people regarding the positive implications of requirements volatility factor on the Software Architecture. For implementation, the 5 Likert scales were used (i.e. from Strongly Agree to Strongly Disagree). This study also tries to get the most relevant practitioners to this study domain to get the most precise information from the industry. As a result, 91 participants participated and submitted their valuable responses. In forgoing this, the objective is achieved and this study contributes to the Software Engineering Body of Knowledge. In the same context, practitioners and academicians can get the platform for ongoing factors and their solutions in the field of Software Engineering. Moreover, the suggestions of the practitioners or industrial people in the form of results generated via an analysis tool along with the charts and frequency distribution tables, against each identified factor are mentioned below.

5.4.1.1 Identified Frequency distribution of the factor ‘Software Defects’

The practitioners gave their consent about this factor namely, ‘Software Defects’. Accordingly, the results indicating the practitioner consents that 47% agreed and accepted that positive implications are found against this factor. While 34% strongly agree and 13% are neutral. However, the results against the rest of the two scales i.e. disagree and strongly disagree are not accepted.

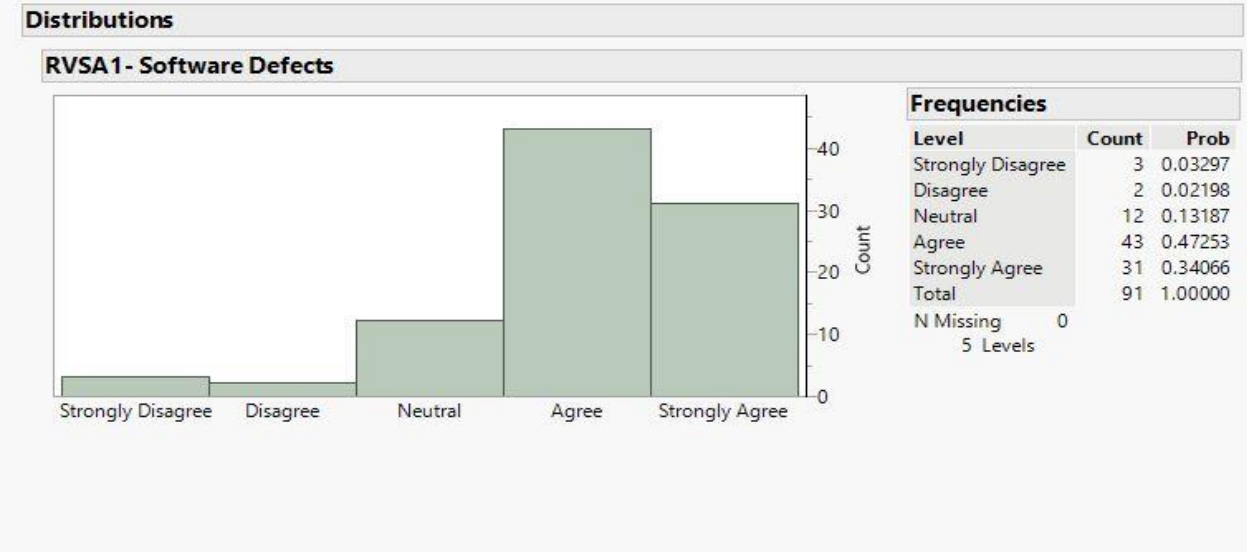


Figure 5.7 Frequency distribution against the factor Software Defects

5.4.1.2 Identified frequency of the factor 'Resource Management'.

The practitioners gave their consent about this factor namely, 'Resource Management. Accordingly, the results indicating the practitioner consents that 43% agreed and accepted that positive implications are found against this factor. While 17% strongly agree and 35% are neutral. However, the results against the rest of the two scales i.e. disagree and strongly disagree are not accepted.

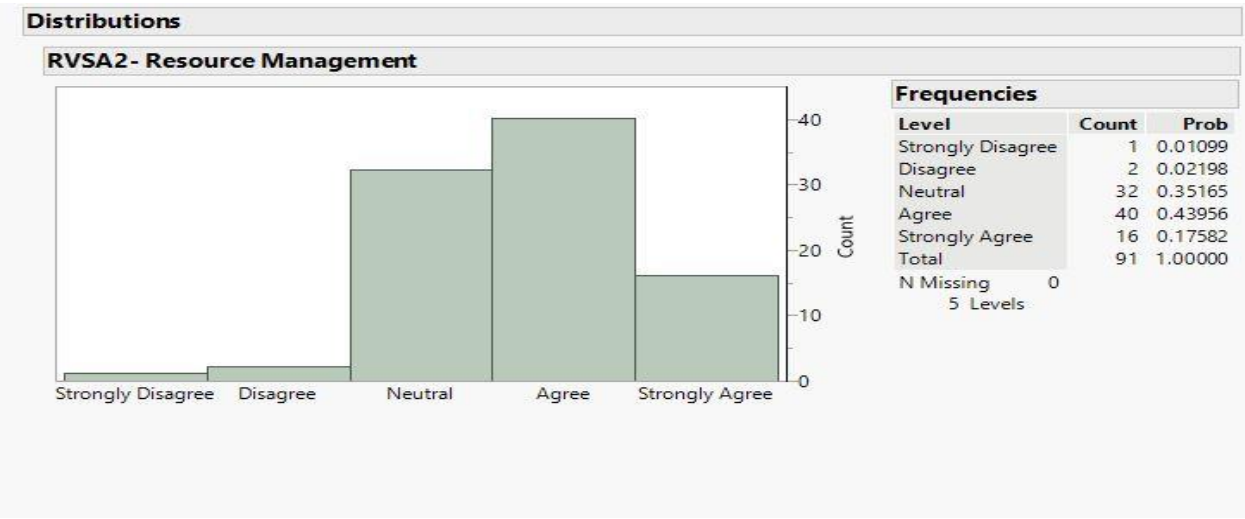


Figure 5.8 Frequency distribution against the factor Resource Management

5.4.1.3 Identified Frequency distribution of the factor 'Knowledge'

The practitioners gave their consent about this factor namely, 'Knowledge'. Accordingly, the results indicating the practitioner consents that 42% agreed and accepted that

positive implications are found against this factor. While 40% strongly agree and 16% are neutral. However, no results were found against the rest of the two scales and treated as 'NIL'.

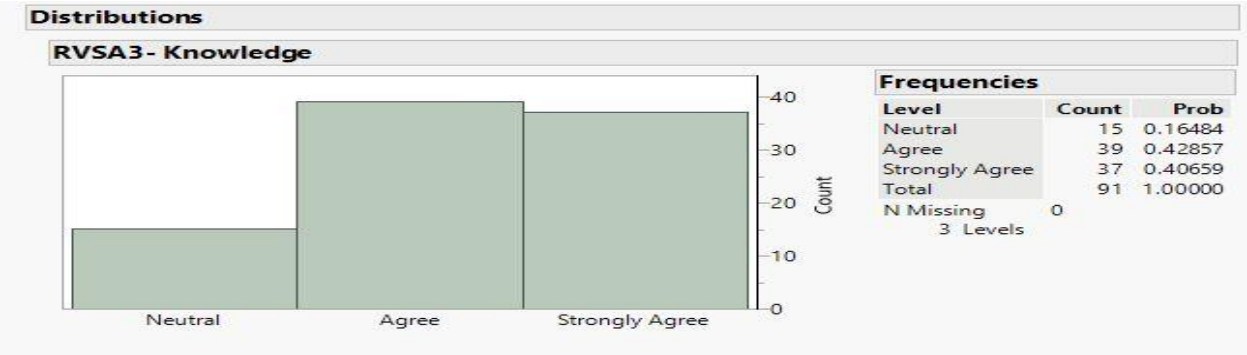


Figure 5.9 Frequency distribution against the factor Knowledge

5.4.1.4 Identified frequency distribution of the factor 'Communication Issues'

The practitioners gave their responses about this factor namely, 'Communication Issues'. Accordingly, the results indicating the practitioner consents that 40% agreed and accepted that positive implications are found against this factor. While 29% strongly agree and 24% are neutral. Besides this, results against the scale strongly disagree are not accepted. However, no results were found against the scale of disagree and treated as 'NIL'.

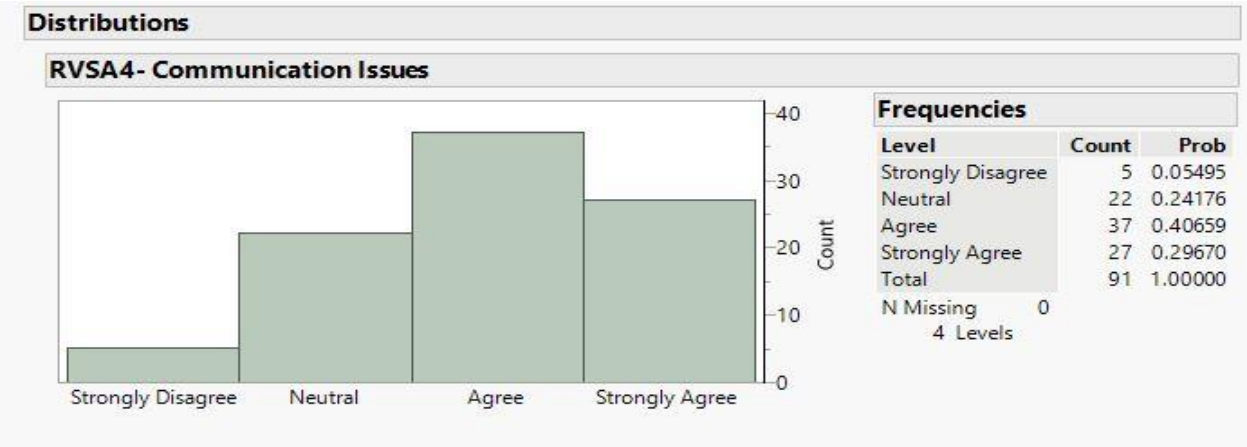


Figure 5.10 Frequency distribution against the factor Communication Issues

5.4.1.5 Identified frequency distribution of the factor 'Dependencies'

The practitioners gave their responses about this factor namely, 'Dependencies'. Accordingly, the results indicate the practitioner consents that 14% agreed and accepted that positive implications are found against this factor. While 30% strongly

agree and 48% are neutral. However, results against the rest of the two scales are not accepted.

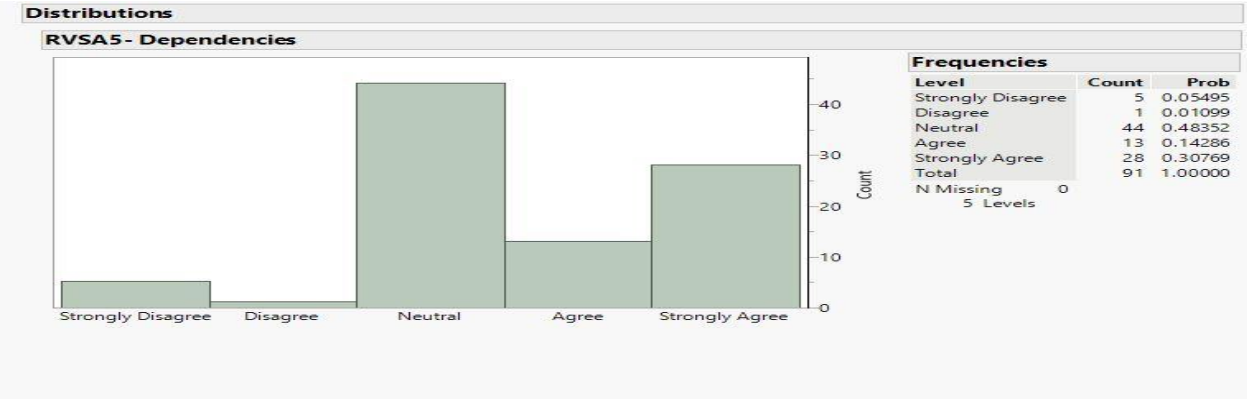


Figure 5.11 Frequency distribution against the factor Dependencies

5.1.4.6 Identified frequency distribution of the factor 'Traceability'

The practitioners gave their responses about this factor namely, 'Traceability'. Accordingly, the results indicating the practitioner consents that 37% agreed and accepted that positive implications are found against this factor. While 28% strongly agree and 32% are neutral. Besides this, results against the scale of Disagree are not accepted. However, no results were found against the scale of strongly disagree and treated as 'NIL'.

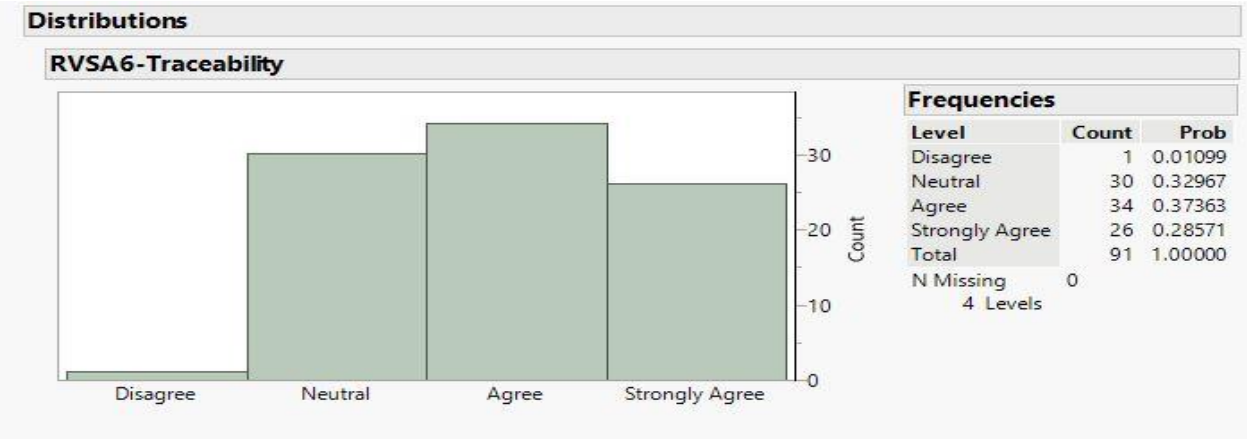


Figure 5.12 Frequency distribution against the factor Traceability

5.1.4.7 Identified frequency distribution of the factor 'Dynamic Business Environment'

The practitioners gave their responses about this factor namely, 'Dynamic Business Environment'. Accordingly, the results indicating the practitioner consents that 36% agreed and accepted that positive implications are found against this factor. While 30%

strongly agree and 26% are neutral. However, results against the rest of the two scales are not accepted.

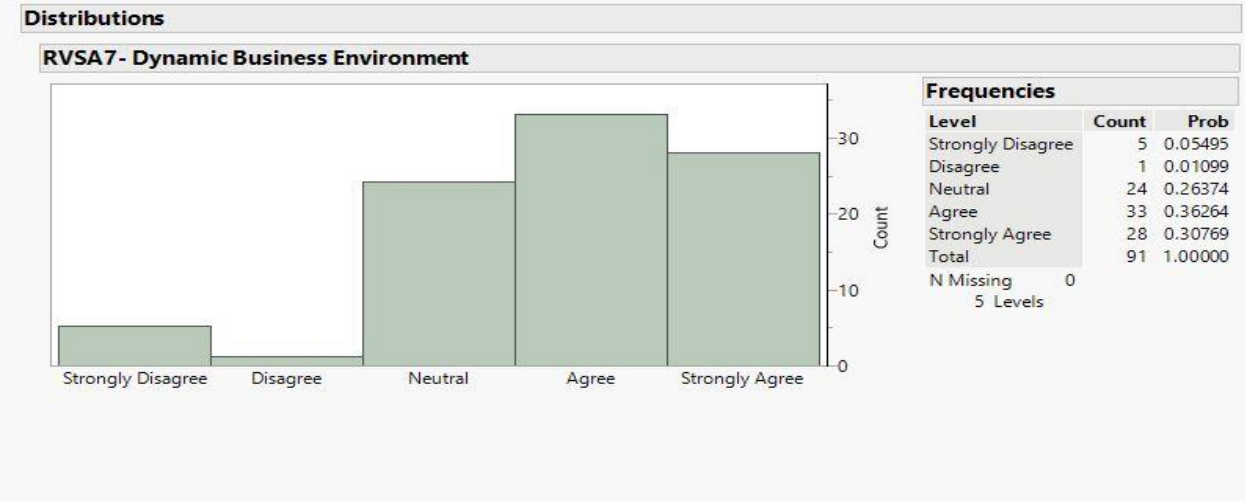


Figure 5.13 Frequency distribution against the factor Dynamic Business Environment

5.1.4.8 Identified frequency distribution of the factor 'Stakeholder Synchronization'

The practitioners gave their responses about this factor namely, 'Stakeholder Synchronization'. Accordingly, the results indicating the practitioner consents that 54% agreed and accepted that positive implications are found against this factor. While 16% strongly agree and 27% are neutral. Besides this, results against the scale disagree are not accepted. However, no results found against the scale strongly disagree and are treated as 'NIL'.

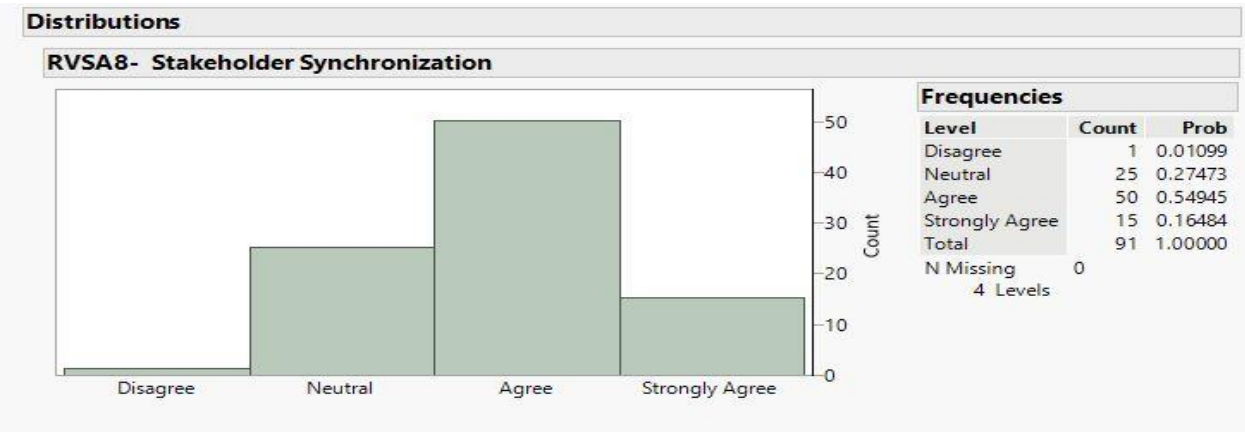


Figure 5.14 Frequency distribution against the factor Stakeholder Synchronization

5.1.4.9 Identified frequency distribution of the factor 'Architecture'

The practitioners gave their responses about this factor namely, 'Architecture'. Accordingly, the results indicating the practitioner consents that 43% agreed and accepted that positive implications are found against this factor. While 28% strongly agree and 25% are neutral. Besides this, results against the scale of disagreement are not accepted. However, no results were found against the strongly disagree and treated as 'NIL'.

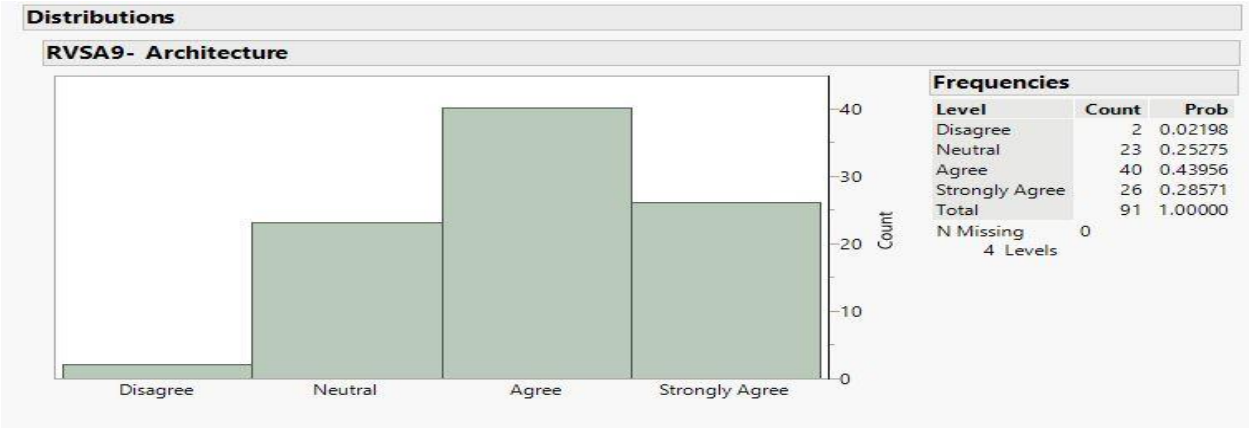


Figure 5.15 Frequency distribution against the factor Architecture

5.1.4.10 Identified frequency distribution of the factor 'Design Implementation'

The practitioners gave their responses about this factor namely, 'Design Implementation'. Accordingly, the results indicating the practitioner consents that 32% agreed and accepted that positive implications are found against this factor. While 24% strongly agree, 31% are neutral and 10% disagree. However, no results were found against the scale of strongly disagree and treated as 'NIL'.

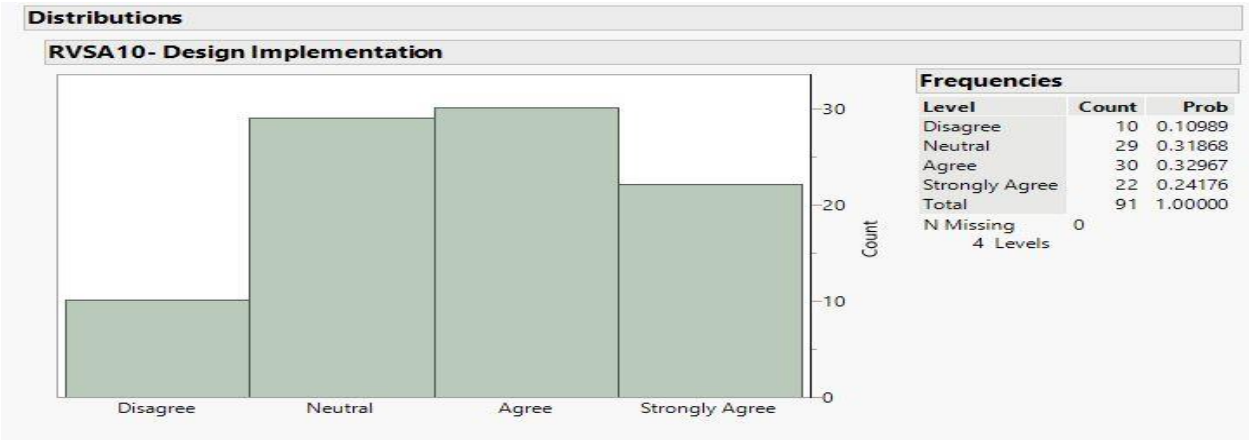


Figure 5.16 Frequency distribution against the factor Design Implementation

5.1.4.11 Identified frequency distribution of the factor 'Organizational Leadership'

The practitioners gave their responses about this factor namely, 'Organization Leadership'. Accordingly, the results indicate the practitioner's consent, whereby, 49% agreed and accepted that positive implications are found against this factor. While 17% strongly agree 19% are neutral and 13% disagree. However, no results were found against the scale of strongly disagree and treated as 'NIL'.

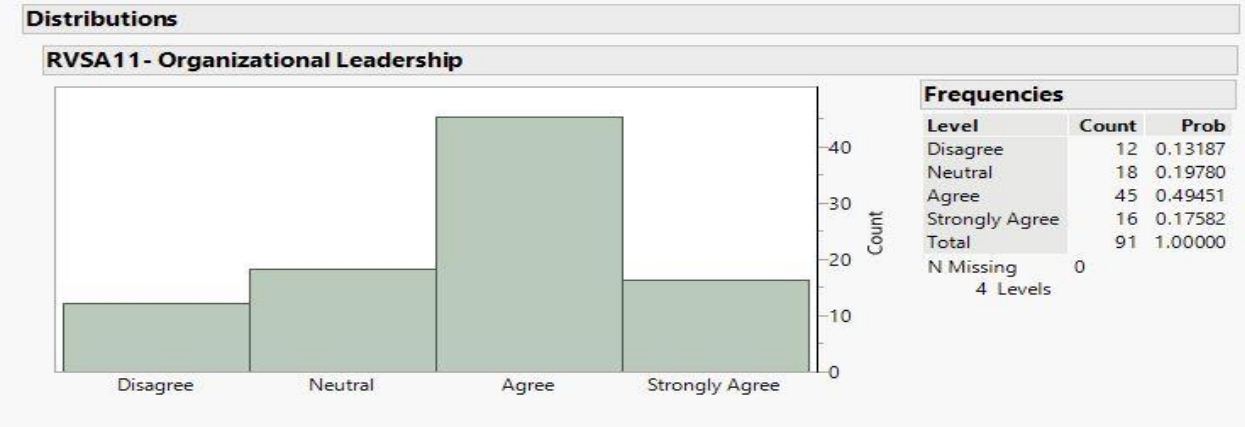


Figure 5.17 Frequency distribution against the factor of Organizational Leadership

5.1.4.12 Identified frequency distribution of the factor 'Adaption to Change'

The practitioners gave their responses about this factor namely, 'Adaption to Change'. Accordingly, the results indicating the practitioner consents that 56% agreed and accepted that positive implications are found against this factor. While 24% strongly agree and 19% are neutral. However, no results were found against the rest of the two scales and treated as 'NIL'.

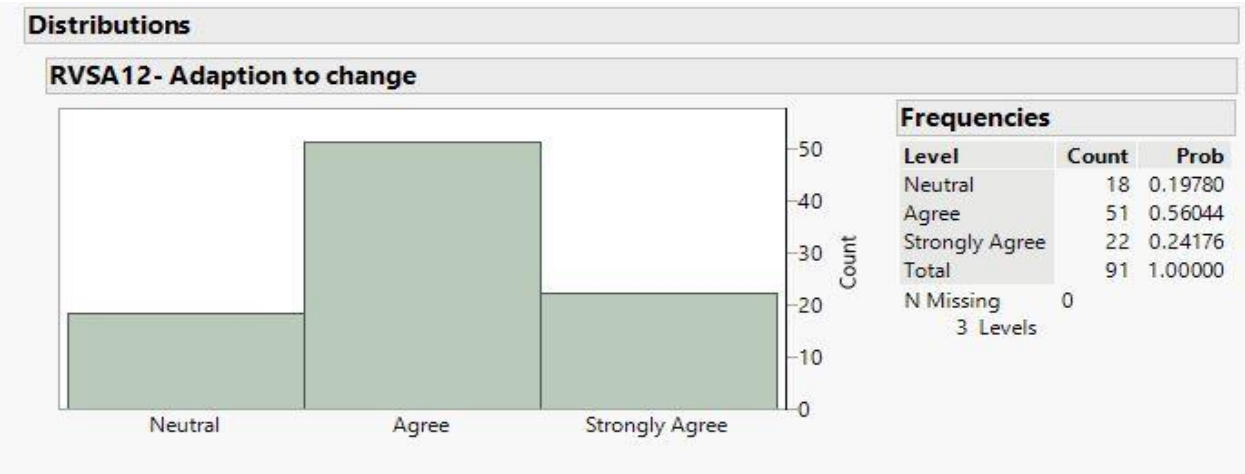


Figure 5.18 Frequency distribution against the factor Adaption to Change

5.1.4.13 Identified frequency distribution of the factor 'SQW Maintenance'

The practitioners gave their responses about this factor namely, 'SQW Maintenance'. Accordingly, the results indicating the practitioner consents that 34% agreed and accepted that positive implications are found against this factor. The same, 34% strongly agree and 31% neutral. However, no results were found against the rest of the two scales and treated as 'NIL'

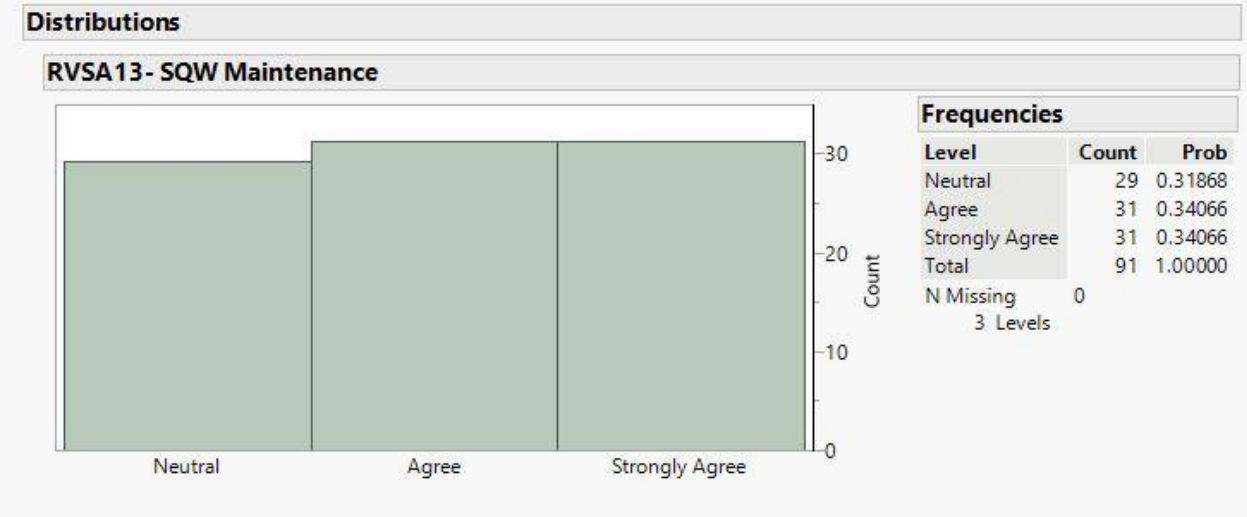


Figure 5.19 Frequency distribution against the factor SQW Maintenance

5.1.4.14 Identified frequency distribution of the factor 'Artefacts'

The practitioners gave their responses about this factor namely, 'Artefacts'. Accordingly, the results indicating the practitioner consents that 47% agreed and accepted that positive implications are found against this factor. While 34% strongly agree and 18% are neutral. However, no results were found against the rest of the two scales and treated as 'NIL'.

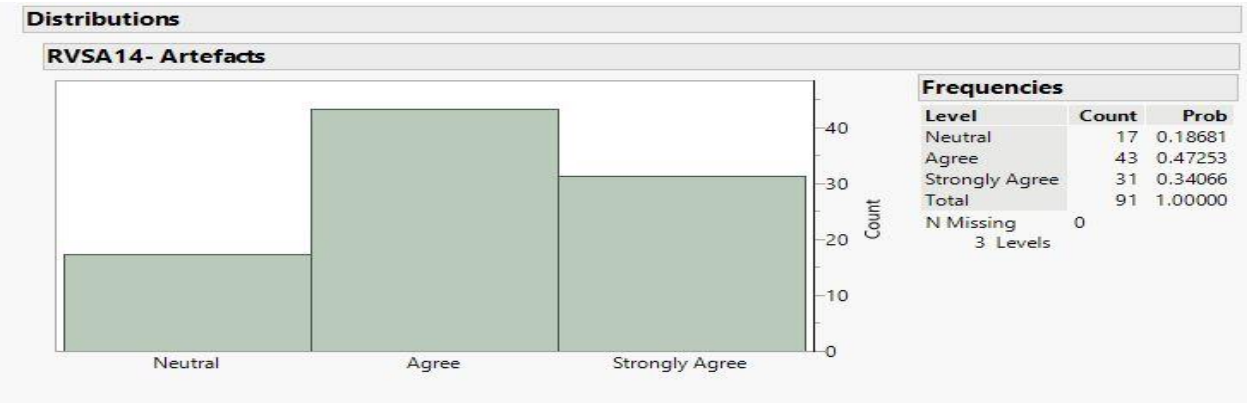


Figure 5.20 Frequency distribution against the factor Artefacts

5.1.4.15 Identified frequency distribution of the factor 'Integration of Usage'

The practitioners gave their responses about this factor namely, 'Integration of Usage'. Accordingly, the results indicate the practitioner consents that 64% agreed and accepted that positive implications are found against this factor. While 8% strongly agree and 26% are neutral. However, no results were found against the rest of the two scales and treated as 'NIL'.

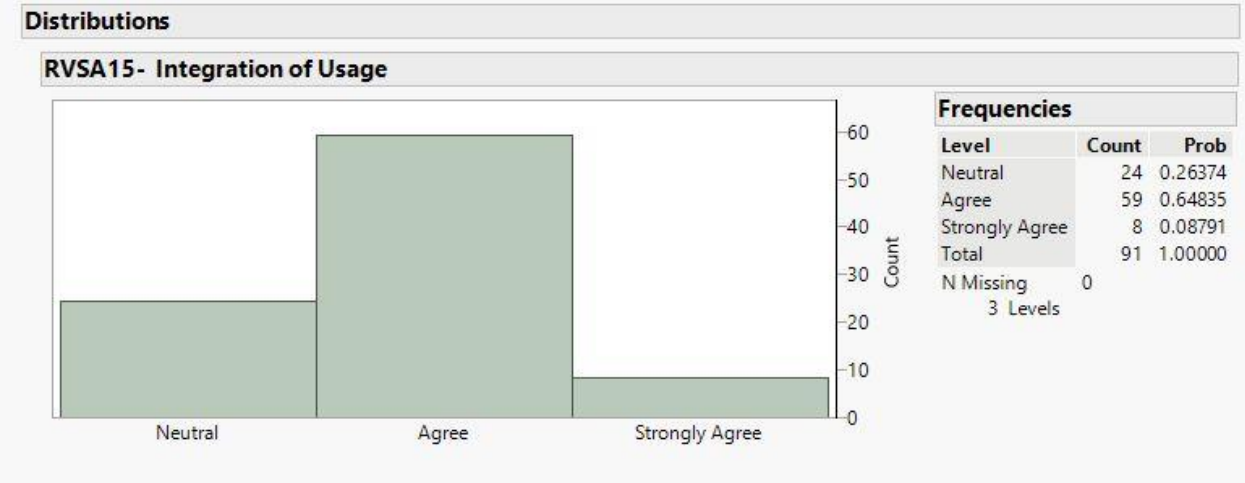


Figure 5.21 Frequency distribution against the factor Integration of Usage

5.1.4.16 Identified frequency distribution of the factor 'Trade-off'

The practitioners gave their responses about this factor namely, 'Trade-off'. Accordingly, the results indicating the practitioner consents that 39% agreed and accepted that positive implications are found against this factor. While 25% strongly agree and 34% are neutral. Besides this, results against the scale disagree are not accepted. However, no results found against the scale strongly disagree and are treated as 'NIL'.

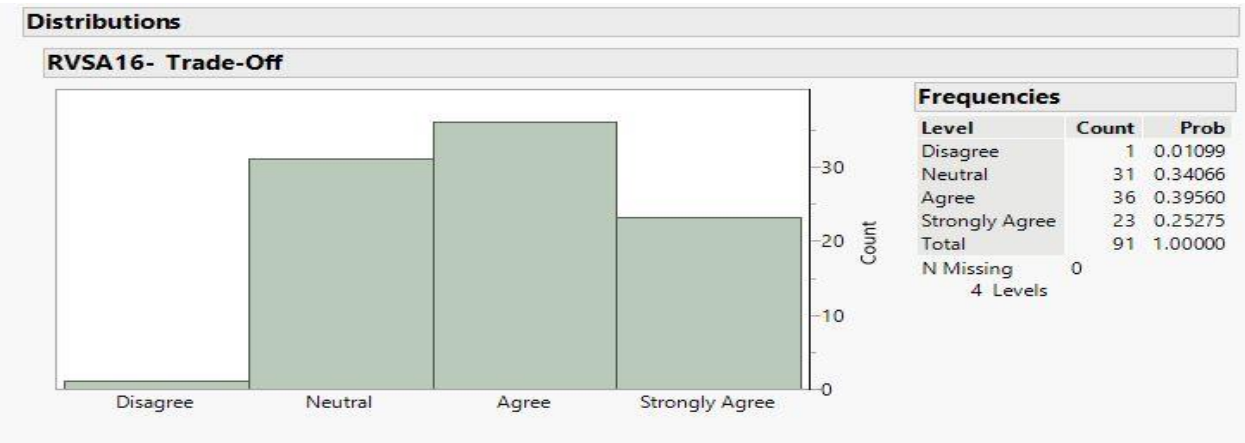


Figure 5.22 Frequency distribution against the factor Trade-off

5.1.4.17 Identified frequency distribution of the factor 'Code'

The practitioners gave their responses about this factor namely, 'Code'. Accordingly, the results indicating the practitioner consents that 38% agreed and accepted that positive implications are found against this factor. While 39% strongly agree and 19% are neutral. Besides this, results against the scale disagree are not accepted. However, no results found against the scale strongly disagree and are treated as 'NIL'.

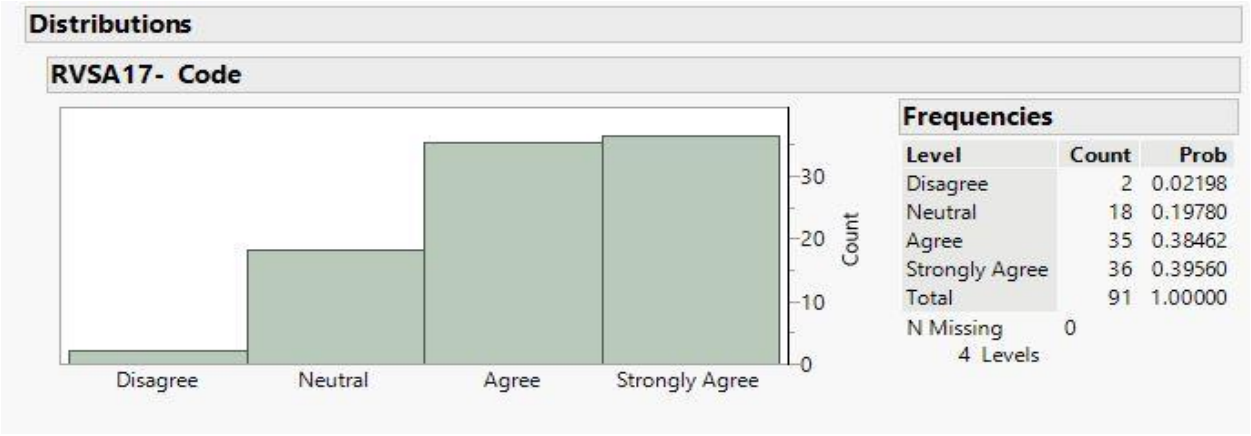


Figure 5.23 Frequency distribution against the factor Code

5.1.4.18 Identified frequency distribution of the factor 'Technical Debt.'

The practitioners gave their responses about this factor namely, 'Technical Debt'. Accordingly, the results indicate the practitioner consents that 41% agreed and accepted that positive implications are found against this factor. While 23% strongly agree and 27% are neutral. Besides this, results against the scale disagree are not accepted. However, no results found against the scale strongly disagree and are treated as 'NIL'.

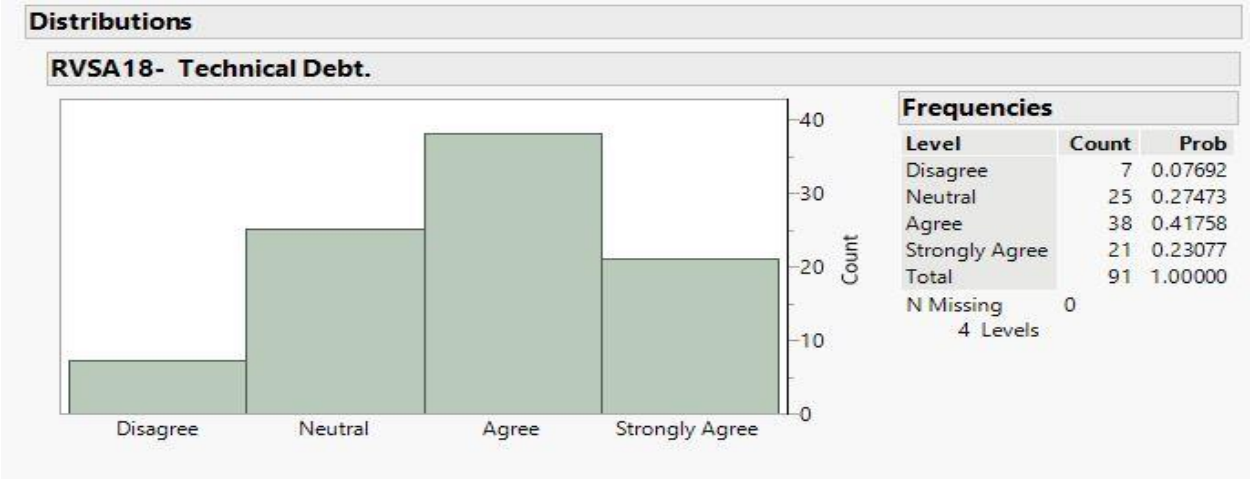


Figure 5.24 Frequency distribution against the factor Technical Debt.

5.1.4.19 Identified frequency distribution of the factor 'Human Behavior'

The practitioners gave their responses about this factor namely, 'Human Behavior'. Accordingly, the results indicating the practitioner consents that 43% agreed and accepted that positive implications are found against this factor. While 26% strongly agree and 23% are neutral. However, results against the rest of the scales are not accepted.

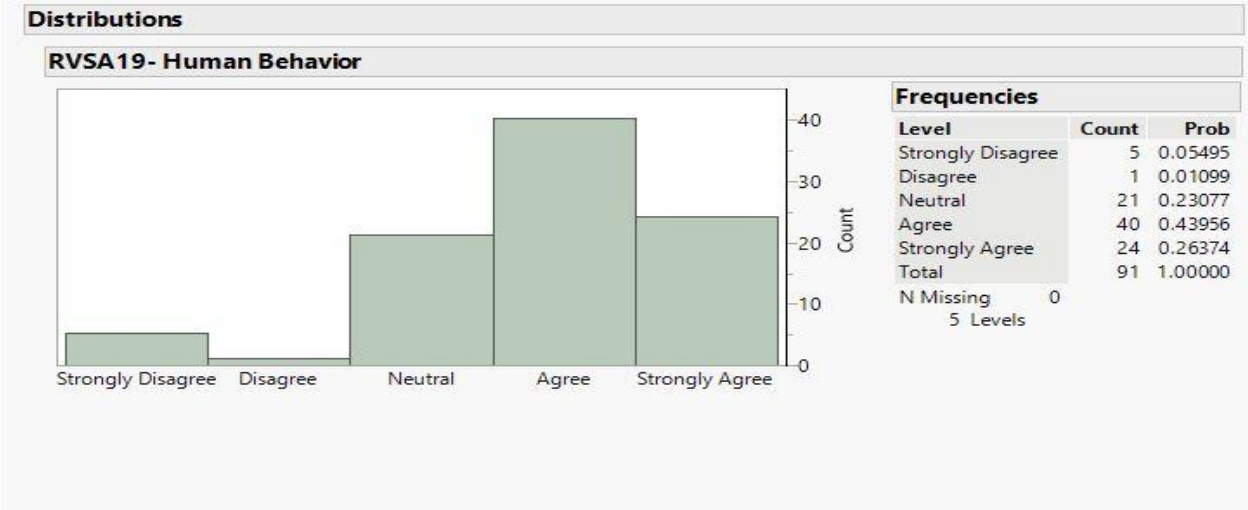


Figure 5.25 Frequency distribution against the factor Human Behavior

5.1.4.20 Identified frequency distribution of the factor 'Team'

The practitioners gave their responses about this factor namely, 'Team'. Accordingly, the results indicating the practitioner consents that 49% agreed and accepted that positive implications are found against this factor. While 31% strongly agree and 12% are neutral. Besides this, results against the scale of disagree are not accepted. However, no results found against the scale strongly disagree and are treated as 'NIL'.

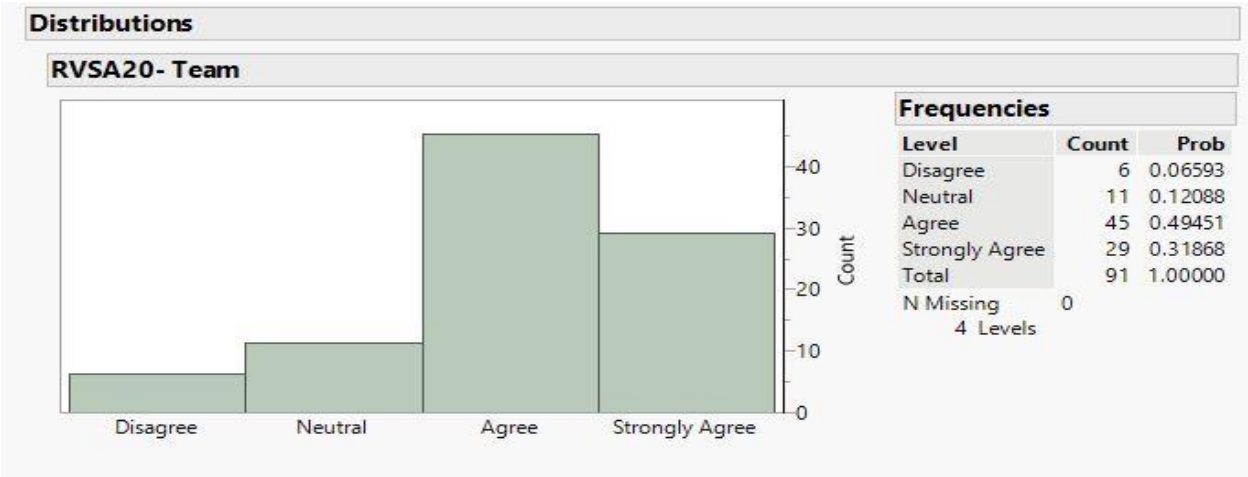


Figure 5.26 Frequency distribution against the factor Team

5.1.4.21 Identified frequency distribution of the factor 'Integration of Linkage'

The practitioners gave their responses about this factor namely, 'Integration of Linkage'. Accordingly, the results indicating the practitioner consents that 36% agreed and accepted that positive implications are found against this factor. While 34% strongly agree, 17% are neutral and 12% disagree. However, no results were found against the strongly disagree and treated as 'NIL'.

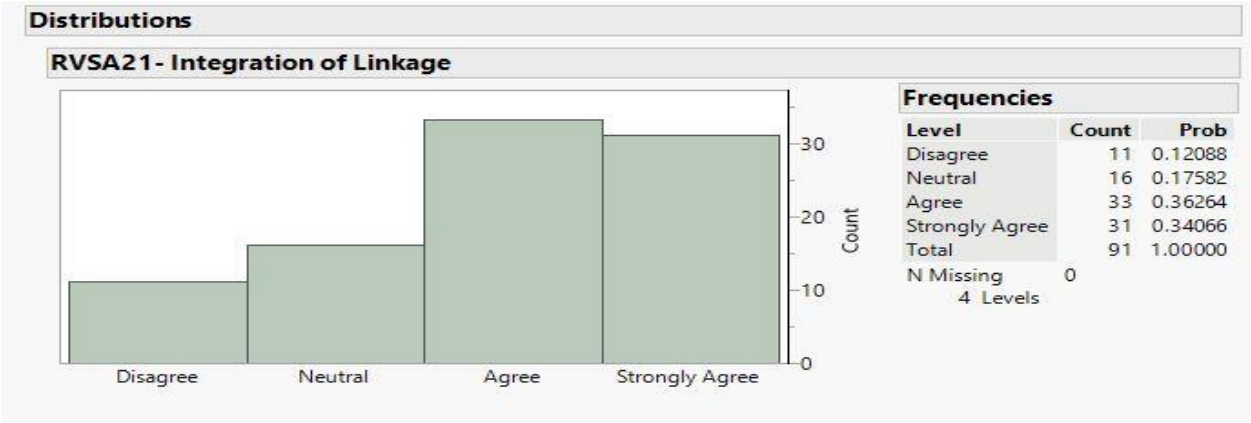


Figure 5.27 Frequency distribution against the factor Integration of Linkage

5.1.4.22 Identified frequency distribution of the factor 'Documentation'

The practitioners gave their responses about this factor namely, 'Documentation'. Accordingly, the results indicating the practitioner consents that 58% agreed and accepted that positive implications are found against this factor. While 17% strongly agree and the same 17% are neutral. Besides this, results against the disagree are not accepted. However, no results found against the scale strongly disagree and are treated as 'NIL'.

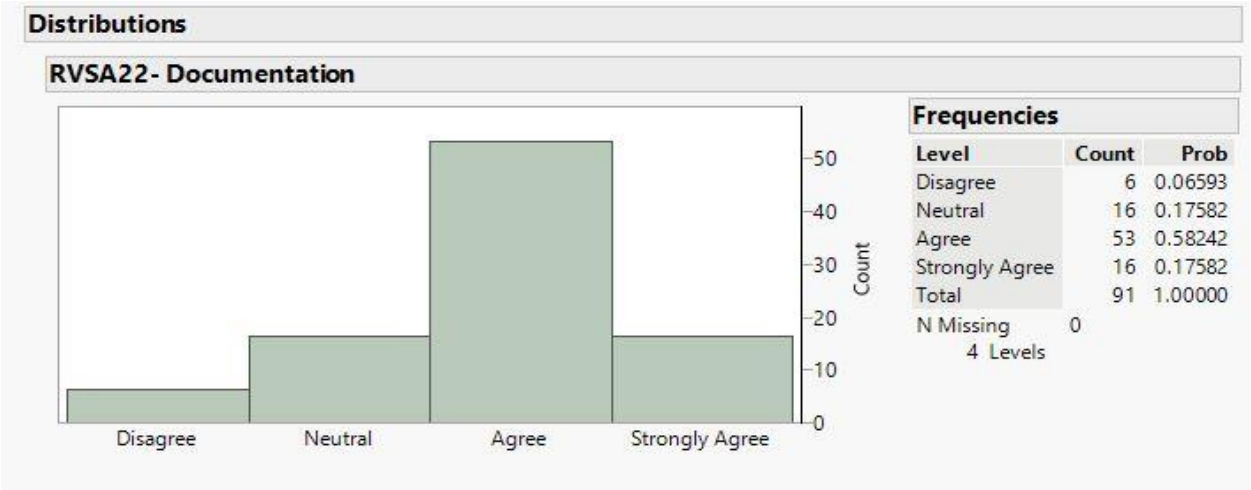


Figure 5.28 Frequency distribution against the factor Documentation

5.1.4.23 Identified frequency distribution of the factor 'Architectural Complexity'

The practitioners gave their responses about this factor namely, 'Architectural Complexity'. Accordingly, the results indicating the practitioner consents that 47% agreed and accepted that positive implications are found against this factor. While 16% strongly agree and 36% are neutral. However, no results were found against the rest of the scales and treated as 'NIL'.

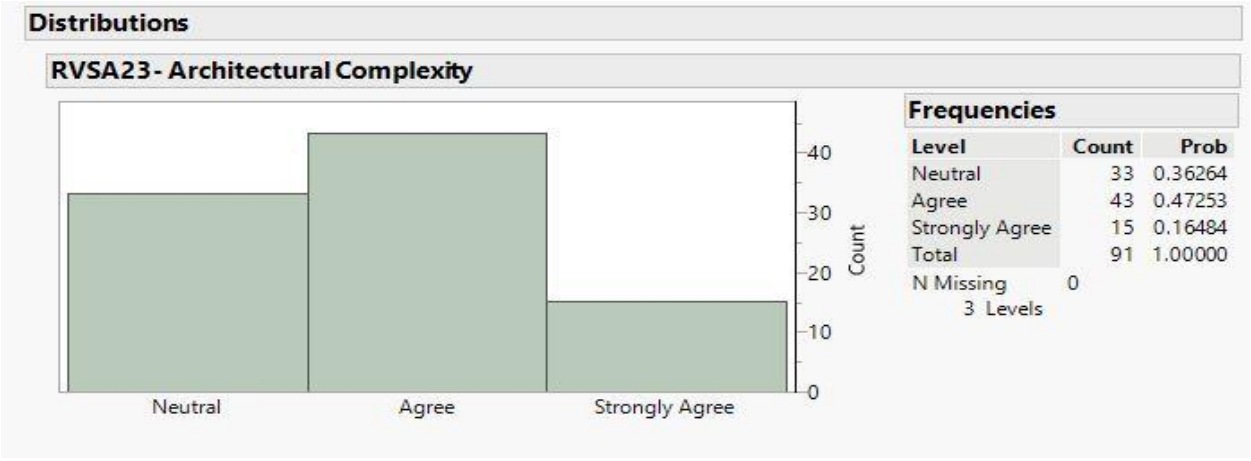


Figure 5.29 Frequency distribution against the factor Architectural Complexity

5.1.4.24 Identified frequency distribution of the factor 'Requirement Volatility'

The practitioners gave their responses about this factor namely, 'Requirement Volatility'. Accordingly, the results indicating the practitioner consents that 38% agreed and accepted that positive implications are found against this factor. While 20% strongly agree and 39% are neutral. Besides this, results against the scale disagree are not accepted. However, no results were found against the strongly disagree and treated as 'NIL'.

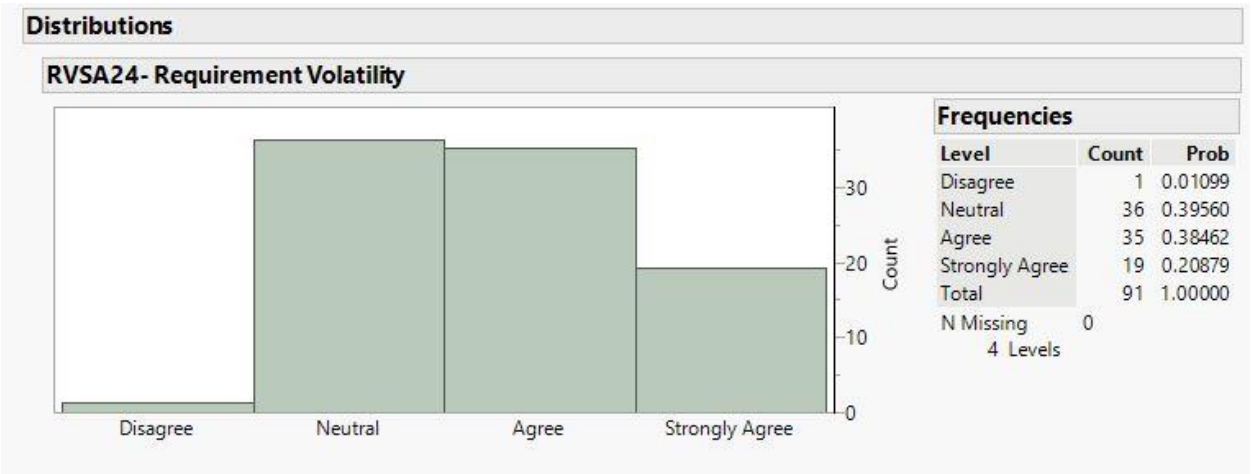


Figure 5.30 Frequency distribution against the factor Requirement Volatility

5.1.4.25 Identified frequency distribution of the factor 'Quality Assurance'

The practitioners gave their responses about this factor namely, 'Quality Assurance'. Accordingly, the results indicating the practitioner consents that 42% agreed and accepted that positive implications are found against this factor. While 35% strongly agree and 20% are neutral. Besides this, results against the scale disagree are not accepted. However, no results found against the scales strongly disagree and are treated as 'NIL'.

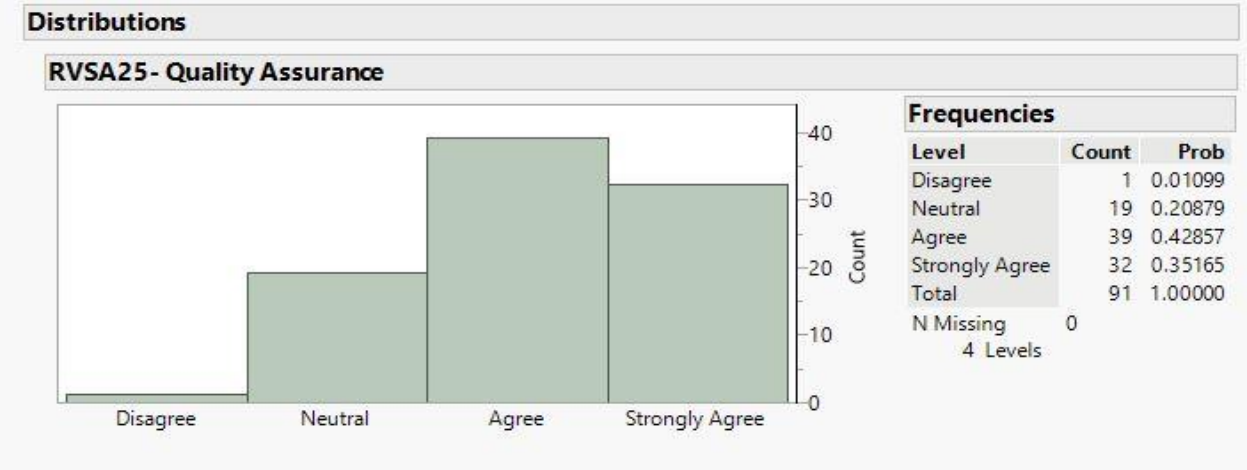


Figure 5.31 Frequency distribution against the factor Quality Assurance

5.1.4.26 Identified frequency distribution of the factor 'Security'

The practitioners gave their responses about this factor namely, 'Security'. Accordingly, the results indicating the practitioner consents that 48% agreed and accepted that positive implications are found against this factor. While 37% strongly agree and 14% are neutral. However, no results were found against the rest of the two scales and treated as 'NIL'.

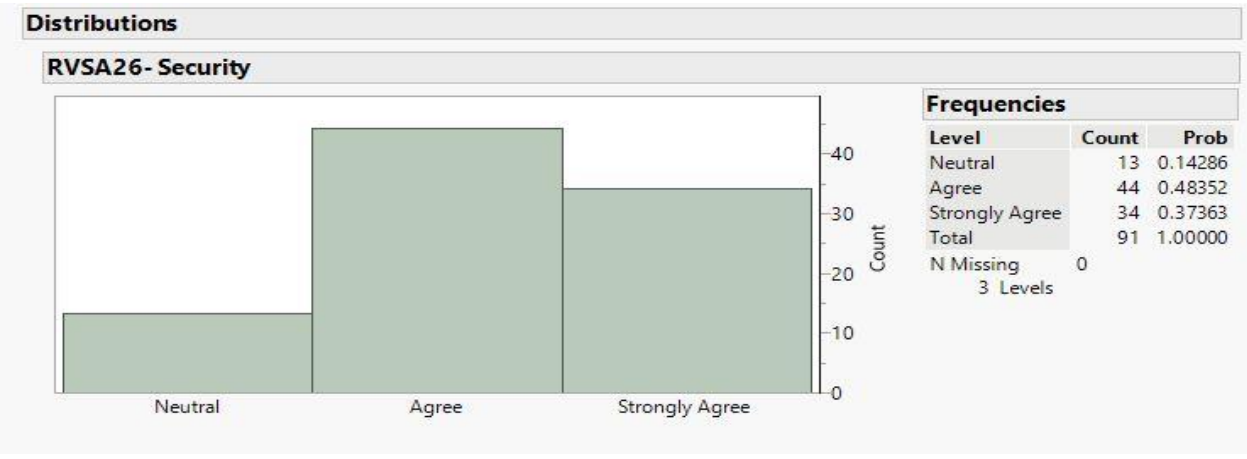


Figure 5.32 Frequency distribution against the factor Security

5.1.4.27 Identified frequency distribution of the factor 'Self-Healing Mechanism'.

The practitioners gave their responses about this factor namely, 'Self-Healing Mechanism'. Accordingly, the results indicating the practitioner consents that 51% agreed and accepted that positive implications are found against this factor. While 21% strongly agree and 26% are neutral. However, no results were found against the rest of the two scales and treated as 'NIL'.

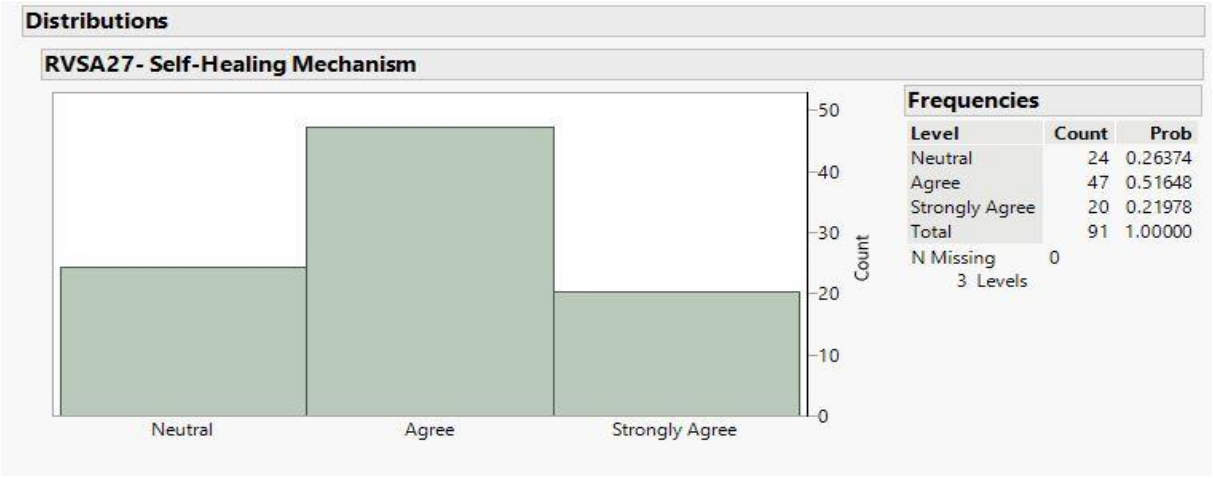


Figure 5.33 Frequency distribution against the factor Self-Healing Mechanism

5.4.2 Analysis of Survey

The results of the conducted survey in terms of collected responses are tabulated below:

Table 5.1 Findings of the Survey

No.	Factor	Strongly Agree (1)	Agree (2)	Neutral (3)	Disagree (4)	Strongly Disagree (5)	Total Responses (91)
1.	Software Defects	31*1=31	43*2=86	12*3=36	2*4=8	3*5=15	176
2.	Resource Management	16*1=16	40*2=80	32*3=96	2*4=8	1*5=5	205
3.	Knowledge	37*1=37	39*2=78	15*3=45	0*4=0	0*5=0	160
4.	Communication Issues	27*1=27	37*2=74	22*3=66	0*4=0	5*5=25	192
5.	Dependencies	28*1=28	13*2=26	44*3=132	1*4=4	5*5=25	215
6.	Traceability	26*1=26	34*2=68	30*3=90	1*4=4	0*5=0	188
7.	Dynamic Business Environment	28*1=28	33*2=64	24*3=72	1*4=4	5*5=25	193
8.	Stakeholder Synchronization	15*1=15	50*2=100	25*3=75	1*4=4	0*5=0	194
9.	Architecture	26*1=26	40*2=80	23*3=69	2*4=8	0*5=0	183
10.	Design Implementation	22*1=22	30*2=60	29*3=87	10*4=40	0*5=0	209
11.	Organizational Leadership	16*1=16	45*2=90	18*3=54	12*4=48	0*5=0	208
12.	Adaption to Change	22*1=22	51*2=102	18*3=54	0*4=0	0*5=0	178
13.	SQW Maintenance	31*1=31	31*2=62	29*3=87	0*4=0	0*5=0	180
14.	Artefacts	31*1=31	43*2=86	17*3=51	0*4=0	0*5=0	168
15.	Integration of Usage	8*1=8	59*2=118	24*3=72	0*4=0	0*5=0	198

No.	Factor	Strongly Agree (1)	Agree (2)	Neutral (3)	Disagree (4)	Strongly Disagree (5)	Total Responses (91)
16.	Trade-Off	23*1=23	36*2=72	31*3=93	1*4=4	0*5=0	192
17.	Code	36*1=36	35*2=70	18*3=54	2*4=8	0*5=0	168
18.	Technical Debt.	21*1=21	38*2=76	25*3=75	7*4=28	0*5=0	200
19.	Human Behavior	24*1=24	40*2=80	21*3=63	1*4=4	5*5=25	196
20.	Team	29*1=29	45*2=90	11*3=33	6*4=24	0*5=0	179
21.	Integration of Linkage	31*1=31	33*2=64	16*3=48	11*4=44	0*5=0	187
22.	Documentation	16*1=16	53*2=106	16*3=48	6*4=24	0*5=0	194
23.	Architectural Complexity	15*1=15	43*2=86	33*3=99	0*4=0	0*5=0	200
24.	Requirement Volatility	19*1=19	35*2=70	36*3=108	1*4=4	0*5=0	201
25.	Quality Assurance	32*1=32	39*2=78	19*3=57	1*4=4	0*5=0	171
26.	Security	34*1=34	44*2=88	13*3=39	0*4=0	0*5=0	161
27.	Self-Healing Mechanism	20*1=20	47*2=94	24*3=72	0*4=0	0*5=0	186

5.4.3 Survey Results from Weightage Values

To consider the responses, the weightage value of the survey results based on the acceptance validity is tabulated below:

Table 5.2 Accepted/Rejected Values

Sr No.	Factor	Weightage Values	AvgWeightage Responses	Status
1.	Software Defects	176	1.9340	Accepted
2.	Resource Management	205	2.2528	Accepted
3.	Knowledge	160	1.7582	Accepted
4.	Communication Issues	192	2.1098	Accepted
5.	Dependencies	215	2.3627	Accepted
6.	Traceability	188	2.0659	Accepted
7.	Dynamic Business Environment	193	2.1209	Accepted
8.	Stakeholder Synchronization	194	2.1319	Accepted
9.	Architecture	183	2.0109	Accepted
10.	Design Implementation	209	2.2970	Accepted
11.	Organizational Leadership	208	2.2857	Accepted
12.	Adaption to Change	178	1.9560	Accepted
13.	SQW Maintenance	180	1.978	Accepted
14.	Artifacts	168	1.8461	Accepted
15.	Integration of Usage	198	2.1758	Accepted
16.	Trade-Off	192	2.1098	Accepted
17.	Code	168	1.8462	Accepted
18.	Technical Debt.	200	2.1978	Accepted
19.	Human Behavior	196	2.1538	Accepted
20.	Team	179	1.9670	Accepted
21.	Integration of Linkage	187	2.0549	Accepted
22.	Documentation	194	2.1318	Accepted
23.	Architectural Complexity	200	2.1978	Accepted

Sr No.	Factor	Weightage Values	AvgWeightage Responses	Status
24.	Requirement Volatility	201	2.2088	Accepted
25.	Quality Assurance	171	1.8791	Accepted
26.	Security	161	1.7692	Accepted
27.	Self-Healing Mechanism	186	2.044	Accepted

5.5 Positive Implications of identified factors (from the top level to bottom)

The survey results intimated the positive implications of the requirements volatility factors. The core purpose of this section is to represent the identified factors in ascending order from very higher to lower levels based on the frequency of the factor distribution analysis, respectively.

Table 5.3 Positive Implications of factors from the top level to bottom

Sr No.	Factor	Freq. Weightage	Implications Level (Percentage)
1.	Adaption to Change	73	68%
2.	Integration of Usage	67	68%
3.	Documentation	69	67%
4.	Security	78	67%
5.	Software Defects	74	64%
6.	Artifacts	74	64%
7.	Team	74	64%
8.	Stakeholder Synchronization	65	62%
9.	Self-Healing Mechanism	67	62%
10.	Quality Assurance	71	60%
11.	Knowledge	76	59%
12.	Organizational Leadership	61	58%
13.	Code	53	58%
14.	Architecture	66	57%
15.	Human Behavior	64	56%
16.	Architectural Complexity	58	55%
17.	Technical Debt.	59	53%
18.	Integration of Linkage	64	53%
19.	Resource Management	56	52%
20.	Communication Issues	64	52%
21.	Trade-Off	59	52%
22.	Traceability	60	51%
23.	Dynamic Business Environment	61	51%
24.	SQW Maintenance	62	51%
25.	Requirement Volatility	54	48%
26.	Design Implementation	52	44%
27.	Dependencies	41	29%

This table indicated the positive implications of identified factors. Where, ten (10) factors are founded in the range of 60%- 68%, fourteen (14) factors are founded in the range of 51%-59%, and three (03) factors are founded in the range of 29%- 48%.

5.6 Summary of the chapter

This chapter discussed the complete details related to the findings or results of the conducted industrial survey. As a result, the targeted objective has been achieved and the task of the RQ2 accomplished. Whereby, the core purpose of this chapter is to identify the positive implications of requirement vitality on the software architecture. Accordingly, the positive implications of the identified factor from the top level to the bottom are reported in this study.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Introduction

This chapter reported the results and findings of this research. Where, this study intimated the accomplishments, especially in the context of their research questions (i.e. RQ1 and RQ2) and research objectives. Moreover, the major role of this study was to contribute to the existing Software Engineering Body of Knowledge (SEBOK) and especially the Requirement Engineering Body of Knowledge (REBOK). Accordingly, proposed the positive implications of requirement volatility factors on the software architecture for practitioners, software engineers, and developers. In the same context, this study contains complete information in terms of their contributions, limitations, and future work.

6.2 Contributions of the Study

As mentioned above, to meet the basic objectives and goals of this conduct of research, this study carried out the two research questions i.e. RQ1 and RQ2. To meet the objective of the RQ1, this study conducted the Systematic Literature Review (SLR) and for smooth conduction used the most popular guidelines of the Kitchenam [23] on account of the used protocol. Accordingly, at the initial phase, this study selected the eighty-three (83) primary studies from the different four digital libraries/resources or electronic databases i.e. IEEE Xplorer, Willey Online Digital Library, Science Direct, and ACM Digital Library. As a result, this study identified a list of twenty-seven (27) requirement volatility factors related to the software architecture. However, to better shape the identified list of factors, the identified information passed through the different stages during this phase of SLR conduction. For implementation, this study conducted the ‘Grounded Theory’, during this the identified list of factors was refined through the further multiple stages, whereby, the constructs are generated through the ‘data encoding technique’ and after execution of the ‘implicit/explicit removal’, this study gets the refined list of factors. Moreover, the identified list of factors is validated through the conduct of the ‘Expert Review’. In which, the Expert of the domain evaluated the work and produced their suggestions and recommendations. After the successful conduct of the SLR, this research accomplished the first objective of this study.

To meet the objective of the RQ2, this study conducted an industrial survey. The core purpose of this conduct of industrial survey is to empirically investigate the positive implications of the requirement volatility factors on the software architecture. In the same context, this study received a total number of 91 responses from practitioners and industrial people. As a result, this study proposed the positive implications of factors from its top level to the down level along with their positive implications in the shape of hierarchy.

6.3 Threats Validity

To streamline the proposed solutions of this research study, there is an essential need to focus on its threat validity, as well. In the same context, at the initial step, this study most carefully focused on the primary selected studies in terms of their publication related to the published and unpublished material from the time frame 2010-2020. In this regard, this study only selected that published material. However, forthcoming publications or accepted manuscripts are also considered. Moreover, premature conferences, proceedings, or newsletter material are not considered.

6.4 Future Work

This study was stickier to find out the positive implications of requirement volatility factors on the software architecture. In the future, we may conduct a comparison-based study related to this study domain, whereas, we may get the complete analysis in terms of their positive as well as negative implications (if any).

6.5 Conclusion

Requirements volatility is a vital part of the Requirement Engineering process. Here, the term volatility indicates its fragile nature and also intimate about its positive worth, to proceed with upcoming changes. Because, requirements are needed to be added, deleted, and modified throughout the SW development process. On the other hand, software architecture provides a complete vision of the system that is going to build. This phenomenon indicates the close connection of these twin peaks of SDLC. However, prior studies intimated about requirements volatility factors and their causes but none of them intimated about its positive implications on SW architecture. This study fulfills this gap in three different phases, whereby, phase 1 is conducted to identify all possible requirements volatility factors related to the software architecture through SLR conduction. In phase 2, all the identified factors are validated by the experts of the domain through the conduct of an expert review. In phase 3,

this study proposed the positive implications of these identified factors on software architecture through the conduct of an industrial survey. This study explored the 83 primary studies, as a result, proposed the 27 factors and acknowledges their positive implications on software architecture. In addition, the identified factors are also categorized into three different categories i.e. ‘Internal’, ‘External’, and ‘both’. Whereas, only 13 factors were found as internal factors. However, most of the factors are based on internal as well as external, both. Accordingly, this study also contributes to the existing Software Engineering Body of Knowledge and Requirement Engineering Body of Knowledge.

REFERENCES

[1] S. Dasanayake, S. Aaramaa, J. Markkula, and M. Oivo, “Impact of requirements volatility on software architecture: How do software teams keep up with ever-changing requirements?,” *J. Softw. Evol. Process*, vol. 31, no. 6, pp. 1–19, 2019.

[2] A. A. Khan, “Systematic literature review and empirical investigation of motivators for requirements change management process in global software development,” no. April, 2019.

[3] M. Zanoni, F. Perin, F. A. Fontana, and G. Viscusi, “Pattern detection for conceptual schema recovery in data-intensive systems,” *J. Softw. Evol. Process*, vol. 26, no. 12, pp. 1172–1192, 2014.

[4] M. A. Akbar, J. Sang, A. A. Khan, and S. Hussain, “Investigation of the requirements change management challenges in the domain of global software development,” *J. Softw. Evol. Process*, vol. 31, no. 10, pp. 1–22, 2019.

[5] S. Aaramaa, S. Dasanayake, M. Oivo, J. Markkula, and S. Saukkonen, “Requirements volatility in software architecture design: An exploratory case study,” *ACM Int. Conf. Proceeding Ser.*, vol. Part F1287, pp. 40–49, 2017.

[6] H. Sadia, S. Q. Abbas, and M. Faisal, “Volatile requirement prioritization: A fuzzy based approach,” *Int. J. Eng. Adv. Technol.*, vol. 8, no. 5, pp. 2467–2472, 2019.

[7] R. Valerdi and M. Pe, “Characterizing the Impact of Requirements Volatility on Systems Engineering Effort,” 2014.

[8] M. W. Grenn, S. Sarkani, and T. Mazzuchi, “A Theory of Information Quality and its Implementation in Systems Engineering,” *IEEE Syst. J.*, vol. 9, no. 4, pp. 1129–1138, 2015.

[9] R. V. M.P.Singh and Abstract-, “Requirements Volatility in Software Development Process,” *Int. J. Soft Comput. Eng.*, vol. 2, no. 4, pp. 259–264, 2012.

[10] A. Goknil, I. Kurtev, and K. Van Den Berg, “Generation and validation of traces between requirements and architecture based on formal trace semantics,” *J. Syst. Softw.*, vol. 88, no. 1, pp. 112–137, 2014.

[11] H. P. Breivold, I. Crnkovic, and M. Larsson, “Software architecture evolution through evolvability analysis,” *J. Syst. Softw.*, vol. 85, no. 11, pp. 2574–2592, 2012.

[12] Y. Fu, M. Li, and F. Chen, “Impact propagation and risk assessment of requirement changes for software development projects based on design structure matrix,” *Int. J. Proj. Manag.*, vol. 30, no. 3, pp. 363–373, 2012.

[13] Y. C. Cavalcanti, I. D. C. MacHado, P. A. D. M. S. Neto, and E. S. De Almeida, “Towards semi-automated assignment of software change requests,” *J. Syst. Softw.*, vol. 115, pp. 82–101, 2016.

[14] A. Ahmad, P. Jamshidi, and C. Pahl, “Classification and comparison of architecture evolution reuse knowledge - A systematic review,” *J. Softw. Evol. Process*, vol. 26, no. 7, pp. 654–691, 2014.

[15] C. López, V. Codocedo, H. Astudillo, and L. M. Cysneiros, “Bridging the gap between software architecture rationale formalisms and actual architecture documents: An ontology-driven approach,” *Sci. Comput. Program.*, vol. 77, no. 1, pp. 66–80, 2012.

[16] R. Weinreich and G. Buchgeher, “Towards supporting the software architecture life cycle,” *J. Syst. Softw.*, vol. 85, no. 3, pp. 546–561, 2012.

[17] X. Liu, L. G. Huang, A. Egyed, and J. Ge, “Do code data sharing dependencies support an early prediction of software actual change impact set?,” *J. Softw. Evol. Process*, vol. 30, no. 11, pp. 16–26, 2018.

[18] F. Tian, T. Wang, P. Liang, C. Wang, A. A. Khan, and M. A. Babar, “The impact of traceability on software maintenance and evolution: A mapping study,” *J. Softw. Evol. Process*, no. July, pp. 1–31, 2021.

[19] M. P. Singh and R. Vyas, “Requirements Volatility in Software Development Process,” *Int. J. Soft Comput. Eng.*, vol. 2, no. 4, pp. 259–264, 2012.

[20] J. A. Miller, R. Ferrari, and N. H. Madhavji, “An exploratory study of architectural effects on requirements decisions,” *J. Syst. Softw.*, vol. 83, no. 12, pp. 2441–2455, 2010.

[21] J. O. Johanssen, A. Kleebaum, B. Paech, and B. Bruegge, “Continuous software engineering and its support by usage and decision knowledge: An interview study with practitioners,” *J. Softw. Evol. Process*, vol. 31, no. 5, pp. 1–25, 2019.

[22] A. AbuHassan, M. Alshayeb, and L. Ghouti, “Software smell detection techniques: A systematic literature review,” *J. Softw. Evol. Process*, vol. 33, no. 3, pp. 1–48, 2021.

[23] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, “Systematic literature reviews in software engineering – A systematic literature review,” *Inf. Softw. Technol.*, vol. 51, no. 1, pp. 7–15, 2009.

[24] M. Kasunic, “Designing an Effective Survey,” no. September, 2005.

[25] M. Hamill and K. Goseva-Popstojanova, “Exploring the missing link: An empirical study of software fixes,” *Softw. Test. Verif. Reliab.*, vol. 24, no. 8, pp. 684–705, 2014.

[26] R. Haesevoets, D. Weyns, and T. Holvoet, “Architecture-centric support for adaptive service collaborations,” *ACM Trans. Softw. Eng. Methodol.*, vol. 23, no. 1, pp. 1–40, 2014.

[27] L. Briand, D. Falessi, S. Nejati, M. Sabetzadeh, and T. Yue, “Traceability and sysml design slices to support safety inspections: A controlled experiment,” *ACM Trans. Softw. Eng. Methodol.*, vol. 23, no. 1, 2014.

[28] 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,” *IEEE Access*, vol. 7, no. 2017, pp. 116585–116611, 2019.

[29] C. Orellana, M. M. Villegas, and H. Astudillo, “Assessing architectural patterns trade-offs using moment-based pattern taxonomies,” *Proc. - 2019 45th Lat. Am. Comput. Conf. CLEI 2019*, 2019.

[30] M. Famelis and M. Chechik, “Managing Design-Time Uncertainty,” no. March, pp. 179–179, 2017.

[31] L. Chen, L. Huang, C. Li, and W. Luo, “Software architecture matching by meta-model extension and refinement,” *Proc. - Asia-Pacific Softw. Eng. Conf. APSEC*, vol. 1, pp. 422–427, 2012.

[32] P. Gaubatz, I. Lytra, and U. Zdun, “Automatic enforcement of constraints in real-time collaborative architectural decision making,” *J. Syst. Softw.*, vol. 103, pp. 128–149, 2015.

[33] G. Borrego, A. L. Morán, R. R. Palacio, A. Vizcaíno, and F. O. García, “Towards a reduction in architectural knowledge vaporization during agile global software development,” *Inf. Softw. Technol.*, vol. 112, pp. 68–82, 2019.

[34] H. Unphon and Y. Dittrich, “Software architecture awareness in long-term software product evolution,” *J. Syst. Softw.*, vol. 83, no. 11, pp. 2211–2226, 2010.

[35] H. Samin, “Priority-Awareness of Non-Functional Requirements under Uncertainty,” *Proc. IEEE Int. Conf. Requir. Eng.*, vol. 2020-Augus, pp. 416–421, 2020.

[36] S. Jayatilleke and R. Lai, “A method of specifying and classifying requirements change,” *Proc. Aust. Softw. Eng. Conf. ASWEC*, pp. 175–180, 2013.

[37] M. Mannion and H. Kaindl, “Product line requirements reuse based on variability management,” *Proc. - Asia-Pacific Softw. Eng. Conf. APSEC*, vol. 2, pp. 148–149, 2012.

[38] J. Li *et al.*, “An initial evaluation of requirements dependency types in change propagation analysis,” *IET Semin. Dig.*, vol. 2012, no. 1, pp. 62–71, 2012.

[39] L. Aladib and S. P. Lee, “Pattern detection and design rationale traceability: An integrated approach to software design quality,” *IET Softw.*, vol. 13, no. 4, pp. 249–259, 2019.

[40] C. Trubiani, A. Ghabi, and A. Egyed, “Exploiting traceability uncertainty between software architectural models and extra-functional results,” *J. Syst. Softw.*, vol. 125, pp. 15–34, 2017.

[41] B. Wang, R. Peng, Y. Li, H. Lai, and Z. Wang, “Requirements traceability technologies and technology transfer decision support: A systematic review,” *J. Syst. Softw.*, vol. 146, pp. 59–79, 2018.

[42] K. Welsh, P. Sawyer, and N. Bencomo, “Towards requirements aware systems: Run-time resolution of design-time assumptions,” *2011 26th IEEE/ACM Int. Conf. Autom. Softw. Eng. ASE 2011, Proc.*, pp. 560–563, 2011.

[43] S. A. Busari and E. Letier, “RADAR: A Lightweight Tool for Requirements and Architecture Decision Analysis,” *Proc. - 2017 IEEE/ACM 39th Int. Conf. Softw. Eng. ICSE 2017*, pp. 552–562, 2017.

[44] S. A. Busari, “Towards search-based modelling and analysis of requirements and architecture decisions,” *ASE 2017 - Proc. 32nd IEEE/ACM Int. Conf. Autom. Softw. Eng.*, pp. 1026–1029, 2017.

[45] U. Van Heesch, P. Avgeriou, and R. Hilliard, “A documentation framework for architecture decisions,” *J. Syst. Softw.*, vol. 85, no. 4, pp. 795–820, 2012.

[46] T. Rocha Silva, M. Winckler, and H. Trætteberg, “Ensuring the Consistency between User Requirements and Task Models: A Behavior-Based Automated Approach,” *Proc. ACM Human-Computer Interact.*, vol. 4, no. EICS, 2020.

[47] C. J. Neill, R. S. Sangwan, and N. H. Kilicay-Ergin, “A Prescriptive Approach to Quality-Focused System Architecture,” *IEEE Syst. J.*, vol. 11, no. 4, pp. 1994–2005, 2015.

[48] J. B. Corbets, C. J. Willy, and J. E. Bischoff, “Evaluating System Architecture Quality and Architecting Team Performance Using Information Quality Theory,” *IEEE Syst. J.*, vol. 12, no. 2, pp. 1139–1147, 2018.

[49] A. Mohsin, N. K. Janjua, S. M. S. Islam, and M. A. Babar, “SAM-SoS: A stochastic software architecture modeling and verification approach for complex system-of-systems,” *IEEE Access*, vol. 8, pp. 177580–177603, 2020.

[50] M. Galster, A. Eberlein, and L. Jiang, “Structuring software requirements for architecture design,” *Proc. Int. Symp. Work. Eng. Comput. Based Syst.*, pp. 119–128, 2013.

[51] L. Shen, X. Peng, and W. Zhao, “Quality-driven self-adaptation: Bridging the gap between requirements and runtime architecture by design decision,” *Proc. - Int. Comput. Softw. Appl. Conf.*, pp. 185–194, 2012.

[52] S. A. Ebad and M. A. Ahmed, “Measuring stability of object-oriented software architectures,” *IET Softw.*, vol. 9, no. 3, pp. 76–82, 2015.

[53] D. Selva, B. Cameron, and E. Crawley, “Patterns in System Architecture Decisions,” *Syst. Eng.*, vol. 19, no. 6, pp. 477–497, 2016.

[54] C. C. Venters *et al.*, “Software sustainability: Research and practice from a software architecture viewpoint,” *J. Syst. Softw.*, vol. 138, pp. 174–188, 2018.

[55] P. Potena, “Optimization of adaptation plans for a service-oriented architecture with cost, reliability, availability and performance tradeoff,” *J. Syst. Softw.*, vol. 86, no. 3, pp. 624–648, 2013.

[56] M. Razavian, B. Paech, and A. Tang, “Empirical research for software architecture decision making: An analysis,” *J. Syst. Softw.*, vol. 149, pp. 360–381, 2019.

[57] V. Cortellessa, R. Mirandola, and P. Potena, “Managing the evolution of a software architecture at minimal cost under performance and reliability constraints,” *Sci. Comput. Program.*, vol. 98, no. P4, pp. 439–463, 2015.

[58] B. J. Williams and J. C. Carver, “Characterizing software architecture changes: A systematic review,” *Inf. Softw. Technol.*, vol. 52, no. 1, pp. 31–51, 2010.

[59] H. P. Breivold, I. Crnkovic, and M. Larsson, “A systematic review of software architecture evolution research,” *Inf. Softw. Technol.*, vol. 54, no. 1, pp. 16–40, 2012.

[60] P. Liang, A. Jansen, P. Avgeriou, A. Tang, and L. Xu, “Advanced quality prediction

model for software architectural knowledge sharing," *J. Syst. Softw.*, vol. 84, no. 5, pp. 786–802, 2011.

[61] C. Yang, P. Liang, and P. Avgeriou, "A survey on software architectural assumptions," *J. Syst. Softw.*, vol. 113, pp. 362–380, 2016.

[62] C. Yang, P. Liang, and P. Avgeriou, "Evaluation of a process for architectural assumption management in software development," *Sci. Comput. Program.*, vol. 168, no. August, pp. 38–70, 2018.

[63] H. Song *et al.*, "Supporting runtime software architecture: A bidirectional-transformation- based approach," *J. Syst. Softw.*, vol. 84, no. 5, pp. 711–723, 2011.

[64] L. De Silva and D. Balasubramaniam, "Controlling software architecture erosion: A survey," *J. Syst. Softw.*, vol. 85, no. 1, pp. 132–151, 2012.

[65] P. Y. Reyes-Delgado, M. Mora, H. A. Duran-Limon, L. C. Rodríguez-Martínez, R. V. O'Connor, and R. Mendoza-Gonzalez, "The strengths and weaknesses of software architecture design in the RUP, MSF, MBASE and RUP-SOA methodologies: A conceptual review," *Comput. Stand. Interfaces*, vol. 47, pp. 24–41, 2016.

[66] J. Gonzalez-Huerta, E. Insfran, S. Abrahão, and G. Scanniello, "Validating a model-driven software architecture evaluation and improvement method: A family of experiments," *Inf. Softw. Technol.*, vol. 57, no. 1, pp. 405–429, 2015.

[67] D. Falessi, G. Cantone, R. Kazman, and P. Kruchten, "Decision-making techniques for software architecture design: A comparative survey," *ACM Comput. Surv.*, vol. 43, no. 4, pp. 1–28, 2011.

[68] N. Bencomo, J. Whittle, P. Sawyer, A. Finkelstein, and E. Letier, "Requirements reflection: Requirements as runtime entities," *Proc. - Int. Conf. Softw. Eng.*, vol. 2, pp. 199–202, 2010.

[69] A. Egyed, "Automatically detecting and tracking inconsistencies in software design models," *IEEE Trans. Softw. Eng.*, vol. 37, no. 2, pp. 188–203, 2011.

[70] K. Welsh, P. Sawyer, and N. Bencomo, "Run-time resolution of uncertainty," *Proc. 2011 IEEE 19th Int. Requir. Eng. Conf. RE 2011*, pp. 355–356, 2011.

[71] K. D. Evensen, "Reducing uncertainty in architectural decisions with AADL," *Proc. Annu. Hawaii Int. Conf. Syst. Sci.*, pp. 1–9, 2011.

[72] P. Araújo-de-Oliveira, F. Durán, and E. Pimentel, "A procedural and flexible approach for specification, modeling, definition, and analysis for self-adaptive systems," *Softw. - Pract. Exp.*, vol. 51, no. 6, pp. 1387–1415, 2021.

[73] E. Stachtiari, A. Mavridou, P. Katsaros, S. Bliudze, and J. Sifakis, "Early validation of system requirements and design through correctness-by-construction," *J. Syst. Softw.*, vol. 145, pp. 52–78, 2018.

[74] H. Christiaans and R. A. Almendra, "Accessing decision-making in software design," *Des. Stud.*, vol. 31, no. 6, pp. 641–662, 2010.

[75] D. Falessi, L. C. Briand, G. Cantone, R. Capilla, and P. Kruchten, "The value of design

rationale information," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 3, pp. 1–32, 2013.

[76] Y. Zheng, C. Cu, and R. N. Taylor, "Maintaining architecture-implementation conformance to support architecture centrality: From single system to product line development," *ACM Trans. Softw. Eng. Methodol.*, vol. 27, no. 2, 2018.

[77] J. L. De La Vara, M. Borg, K. Wnuk, and L. Moonen, "An Industrial Survey of Safety Evidence Change Impact Analysis Practice," *IEEE Trans. Softw. Eng.*, vol. 42, no. 12, pp. 1095–1117, 2016.

[78] A. Molesini, A. Garcia, C. von Flach Garcia Chavez, and T. V. Batista, "Stability assessment of aspect-oriented software architectures: A quantitative study," *J. Syst. Softw.*, vol. 83, no. 5, pp. 711–722, 2010.

[79] T. Haitzer, E. Navarro, and U. Zdun, *Reconciling software architecture and source code in support of software evolution*, vol. 123. Elsevier Inc., 2017.

[80] R. Kazman, M. Gagliardi, and W. Wood, "Scaling up software architecture analysis," *J. Syst. Softw.*, vol. 85, no. 7, pp. 1511–1519, 2012.

[81] E. Eshraghian and V. Rafe, "Performance measurement of models specified through component-based software architectural styles," *Meas. J. Int. Meas. Confed.*, vol. 73, pp. 372–383, 2015.

[82] S. Ghahremani, H. Giese, and T. Vogel, "Improving Scalability and Reward of Utility-Driven Self-Healing for Large Dynamic Architectures," *ACM Trans. Auton. Adapt. Syst.*, vol. 14, no. 3, 2020.

[83] D. Crawford, "Technical correspondence," *Commun. ACM*, vol. 36, no. 11, p. 18, 1993.

[84] Nicoletti, Matias, Silvia Schiaffino, and J. Andres Diaz-Pace. "An optimization-based tool to support the cost-effective production of software architecture documentation." *Journal of Software: Evolution and Process* 27, no. 9 (2015).

[85] Wnuk, Krzysztof, Jaap Kabbedijk, Sjaak Brinkkemper, Björn Regnell, and David Callele. "Exploring factors affecting decision outcome and lead time in large-scale requirements engineering." *Journal of software: Evolution and Process* 27, no. 9 (2015).

[86] Selva, Daniel, Bruce Cameron, and Ed Crawley. "Patterns in system architecture decisions." *Systems Engineering* 19, no. 6 (2016).

[87] Dorn, Christoph, and Richard N. Taylor. "Analyzing runtime adaptability of collaboration patterns." *Concurrency and Computation: Practice and Experience* 27, no. 11 (2015).

[88] Mannaert, Herwig, Jan Verelst, and Kris Ven. "Towards evolvable software architectures based on systems theoretic stability." *Software: Practice and Experience* 42, no. 1 (2012).

[89] Lagerström, Robert, Ulf Sporrong, and Anders Wall. "Increasing software development efficiency and maintainability for complex industrial systems–A case study." *Journal of Software: Evolution and Process* 25, no. 3 (2013).

[90] Ben Charrada, Eya, Anne Koziolek, and Martin Glinz. "Supporting requirements update during software evolution." *Journal of Software: Evolution and Process* 27, no. 3 (2015).

[91] Janes, Andrea, Tadas Remencius, Alberto Sillitti, and Giancarlo Succi. "Managing changes in requirements: an empirical investigation." *Journal of software: evolution and process* 25, no. 12 (2013).

[92] Habhouba, Dounia, Soumaya Cherkaoui, and Alain Desrochers. "Decision-making assistance in engineering-change management process." *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 41, no. 3 (2010).

APPENDIX-A

Table A: THE LIST OF DESIGNED SEARCH STRINGS FOR CONDUCT OF SLR.

Sr No.	Search Strings
1.	(((Requirement) AND volatility) AND “Software architecture”)
2.	(((Requirement) AND volatility) AND “Software design”)
3.	(((Requirement) AND volatility) AND “Software structure”)
4.	(((Requirement) AND volatility) AND “Software construction”)
5.	(((Requirement) AND volatility) AND “Software building”)
6.	(((Requirement) AND change) AND “Software architecture”)
7.	(((Requirement) AND change) AND “Software design”)
8.	(((Requirement) AND change) AND “Software structure”)
9.	(((Requirement) AND change) AND “Software construction”)
10.	(((Requirement) AND change) AND “Software building”)
11.	(((Requirement) AND changeability) AND “Software architecture”)
12.	(((Requirement) AND changeability) AND “Software design”)
13.	(((Requirement) AND changeability) AND “Software structure”)
14.	(((Requirement) AND changeability) AND “Software construction”)
15.	(((Requirement) AND changeability) AND “Software building”)
16.	(((Requirement) AND uncertainty) AND “Software architecture”)
17.	(((Requirement) AND uncertainty) AND “Software design”)
18.	(((Requirement) AND uncertainty) AND “Software structure”)
19.	(((Requirement) AND uncertainty) AND “Software construction”)
20.	(((Requirement) AND uncertainty) AND “Software building”)
21.	(((Requirement) AND unstable) AND “Software architecture”)
22.	(((Requirement) AND unstable) AND “Software design”)
23.	(((Requirement) AND unstable) AND “Software structure”)
24.	(((Requirement) AND unstable) AND “Software construction”)
25.	(((Requirement) AND unstable) AND “Software building”)
26.	(((Requirement) AND inconsistent) AND “Software architecture”)
27.	(((Requirement) AND inconsistent) AND “Software design”)
28.	(((Requirement) AND inconsistent) AND “Software structure”)
29.	(((Requirement) AND inconsistent) AND “Software construction”)
30.	(((Requirement) AND inconsistent) AND “Software building”)
31.	(((Demand) AND volatility) AND “Software architecture”)
32.	(((Demand) AND volatility) AND “Software design”)
33.	(((Demand) AND volatility) AND “Software structure”)
34.	(((Demand) AND volatility) AND “Software construction”)
35.	(((Demand) AND volatility) AND “Software building”)
36.	(((Demand) AND change) AND “Software architecture”)
37.	(((Demand) AND change) AND “Software design”)
38.	(((Demand) AND change) AND “Software structure”)
39.	(((Demand) AND change) AND “Software construction”)
40.	(((Demand) AND change) AND “Software building”)
41.	(((Demand) AND changeability) AND “Software architecture”)

42.	(((Demand) AND changeability) AND “Software design”)
43.	(((Demand) AND changeability) AND “Software structure”)
44.	(((Demand) AND changeability) AND “Software construction”)
45.	(((Demand) AND changeability) AND “Software building”)
46.	(((Demand) AND uncertainty) AND “Software architecture”)
47.	(((Demand) AND uncertainty) AND “Software design”)
48.	(((Demand) AND uncertainty) AND “Software structure”)
49.	(((Demand) AND uncertainty) AND “Software construction”)
50.	(((Demand) AND uncertainty) AND “Software building”)
51.	(((Demand) AND unstable) AND “Software architecture”)
52.	(((Demand) AND unstable) AND “Software design”)
53.	(((Demand) AND unstable) AND “Software structure”)
54.	(((Demand) AND unstable) AND “Software construction”)
55.	(((Demand) AND unstable) AND “Software building”)
56.	(((Demand) AND inconsistent) AND “Software architecture”)
57.	(((Demand) AND inconsistent) AND “Software design”)
58.	(((Demand) AND inconsistent) AND “Software structure”)
59.	(((Demand) AND inconsistent) AND “Software construction”)
60.	(((Demand) AND inconsistent) AND “Software building”)
61.	(((Condition) AND volatility) AND “Software architecture”)
62.	(((Condition) AND volatility) AND “Software design”)
63.	(((Condition) AND volatility) AND “Software structure”)
64.	(((Condition) AND volatility) AND “Software construction”)
65.	(((Condition) AND volatility) AND “Software building”)
66.	(((Condition) AND change) AND “Software architecture”)
67.	(((Condition) AND change) AND “Software design”)
68.	(((Condition) AND change) AND “Software structure”)
69.	(((Condition) AND change) AND “Software construction”)
70.	(((Condition) AND change) AND “Software building”)
71.	(((Condition) AND changeability) AND “Software architecture”)
72.	(((Condition) AND changeability) AND “Software design”)
73.	(((Condition) AND changeability) AND “Software structure”)
74.	(((Condition) AND changeability) AND “Software construction”)
75.	(((Condition) AND changeability) AND “Software building”)
76.	(((Condition) AND uncertainty) AND “Software architecture”)
77.	(((Condition) AND uncertainty) AND “Software design”)
78.	(((Condition) AND uncertainty) AND “Software structure”)
79.	(((Condition) AND uncertainty) AND “Software construction”)
80.	(((Condition) AND uncertainty) AND “Software building”)
81.	(((Condition) AND unstable) AND “Software architecture”)
82.	(((Condition) AND unstable) AND “Software design”)
83.	(((Condition) AND unstable) AND “Software structure”)
84.	(((Condition) AND unstable) AND “Software construction”)
85.	(((Condition) AND unstable) AND “Software building”)
86.	(((Condition) AND inconsistent) AND “Software architecture”)
87.	(((Condition) AND inconsistent) AND “Software design”)
88.	(((Condition) AND inconsistent) AND “Software structure”)
89.	(((Condition) AND inconsistent) AND “Software construction”)

90.	(((Condition) AND inconsistent) AND “Software building”)
91.	(((Essential) AND volatility) AND “Software architecture”)
92.	(((Essential) AND volatility) AND “Software design”)
93.	(((Essential) AND volatility) AND “Software structure”)
94.	(((Essential) AND volatility) AND “Software construction”)
95.	(((Essential) AND volatility) AND “Software building”)
96.	(((Essential) AND change) AND “Software architecture”)
97.	(((Essential) AND change) AND “Software design”)
98.	(((Essential) AND change) AND “Software structure”)
99.	(((Essential) AND change) AND “Software construction”)
100.	(((Essential) AND change) AND “Software building”)
101.	(((Essential) AND changeability) AND “Software architecture”)
102.	(((Essential) AND changeability) AND “Software design”)
103.	(((Essential) AND changeability) AND “Software structure”)
104.	(((Essential) AND changeability) AND “Software construction”)
105.	(((Essential) AND changeability) AND “Software building”)
106.	(((Essential) AND uncertainty) AND “Software architecture”)
107.	(((Essential) AND uncertainty) AND “Software design”)
108.	(((Essential) AND uncertainty) AND “Software structure”)
109.	(((Essential) AND uncertainty) AND “Software construction”)
110.	(((Essential) AND uncertainty) AND “Software building”)
111.	(((Essential) AND unstable) AND “Software architecture”)
112.	(((Essential) AND unstable) AND “Software design”)
113.	(((Essential) AND unstable) AND “Software structure”)
114.	(((Essential) AND unstable) AND “Software construction”)
115.	(((Essential) AND unstable) AND “Software building”)
116.	(((Essential) AND inconsistent) AND “Software architecture”)
117.	(((Essential) AND inconsistent) AND “Software design”)
118.	(((Essential) AND inconsistent) AND “Software structure”)
119.	(((Essential) AND inconsistent) AND “Software construction”)
120.	(((Essential) AND inconsistent) AND “Software building”)
121.	(((Need) AND volatility) AND “Software architecture”)
122.	(((Need) AND volatility) AND “Software design”)
123.	(((Need) AND volatility) AND “Software structure”)
124.	(((Need) AND volatility) AND “Software construction”)
125.	(((Need) AND volatility) AND “Software building”)
126.	(((Need) AND change) AND “Software architecture”)
127.	(((Need) AND change) AND “Software design”)
128.	(((Need) AND change) AND “Software structure”)
129.	(((Need) AND change) AND “Software construction”)
130.	(((Need) AND change) AND “Software building”)
131.	(((Need) AND changeability) AND “Software architecture”)
132.	(((Need) AND changeability) AND “Software design”)
133.	(((Need) AND changeability) AND “Software structure”)
134.	(((Need) AND changeability) AND “Software construction”)
135.	(((Need) AND changeability) AND “Software building”)
136.	(((Need) AND uncertainty) AND “Software architecture”)
137.	(((Need) AND uncertainty) AND “Software design”)

138.	(((Need) AND uncertainty) AND “Software structure”)
139.	(((Need) AND uncertainty) AND “Software construction”)
140.	(((Need) AND uncertainty) AND “Software building”)
141.	(((Need) AND unstable) AND “Software architecture”)
142.	(((Need) AND unstable) AND “Software design”)
143.	(((Need) AND unstable) AND “Software structure”)
144.	(((Need) AND unstable) AND “Software construction”)
145.	(((Need) AND unstable) AND “Software building”)
146.	(((Need) AND inconsistent) AND “Software architecture”)
147.	(((Need) AND inconsistent) AND “Software design”)
148.	(((Need) AND inconsistent) AND “Software structure”)
149.	(((Need) AND inconsistent) AND “Software construction”)
150.	(((Need) AND inconsistent) AND “Software building”)

APPENDIX-B

Table B: QUALITY ASSESSMENT INCLUDING DISTRIBUTION OF STUDIES AND PARTICIPANTS.

Database & Participants	Study Name	Study Type	QA Score	Status
IEEE & P1	A Prescriptive Approach to Quality-Focused System Architecture	Research Article (Journal), 2015	1	Included
	Evaluating System Architecture Quality and Architecting Team Performance Using Information Quality Theory	Research Article (Journal), 2017	0.92	Included
	SAM-SoS: A Stochastic Software Architecture Modeling and Verification Approach for Complex System-of-Systems	Research Article (Journal), 2020	0.92	Included
	Automatically Detecting and Tracking Inconsistencies in Software Design Models	Research Article (Journal), 2011	0.71	Included
	Decision-Making Assistance in Engineering Change Management Process	Research Article (Journal), 2011	0.57	Included
IEEE & P2	Comparative Analysis of Requirement Change Management Challenges Between in-House and Global Software Development: Findings of Literature and Industry Survey	Research Article (Journal), 2019	0.57	Included
	An Industrial Survey of Safety Evidence Change Impact Analysis Practice	Research Article (Journal), 2016	0.92	Included
	Structuring Software Requirements for Architecture Design	Conference Paper (20th C), 2013	1	Included
	Software Architecture Matching by Meta-model Extension and Refinement	Conference Paper (19th C), 2012	1	Included
	Product Line Requirements Reuse Based on Variability Management	Conference Paper (19th C), 2012	0.65	Included
IEEE & P3	Quality-Driven Self-Adaptation: Bridging the Gap between Requirements and Runtime Architecture by Design Decision	Conference Paper (36th C), 2012	1	Included

IEEE & P3	Priority-Awareness of Non-Functional Requirements under Uncertainty	Conference Paper (28th C), 2020	0.57	Included
	An initial evaluation of requirements dependency types in change propagation analysis	Conference Paper (16th C), 2012	0.92	Included
	A Method of Specifying and Classifying Requirements Change	Conference Paper (22nd C), 2013	0.71	Included
	Towards requirements aware systems: Run-time resolution of design-time assumptions	Conference Paper (22nd C), 2011	0.85	Included
	RADAR: A Lightweight Tool for Requirements and Architecture Decision Analysis	Conference Paper (39th C), 2017	0.78	Included
	Towards search-based modeling and analysis of requirements and architecture decisions	Conference Paper (32nd C), 2017	0.78	Included
	Run-time Resolution of Uncertainty	Conference Paper (19th C), 2011	0.92	Included
	Assessing Architectural Patterns Trade-offs using Moment-based Pattern Taxonomies	Conference Paper (45th C), 2020	0.57	Included
	Improving Software Performance and Reliability with an Architecture-Based Self-Adaptive Framework	Conference Paper (34th C), 2010	0.57	Included
IEEE & P4	Managing Design Time Uncertainty	Conference Paper (20th C), 2017	0.92	Included
	Reducing Uncertainty in Architectural Decisions with AADL	Conference Paper (44th C), 2011	0.71	Included
	Inconsistency Management between Architectural Decisions and Designs Using Constraints and Model Fixes	Conference Paper (23rd C), 2014	0.78	Included
Wiley & P4	Impact of requirements volatility on software architecture: How do software teams keep up with ever-changing requirements?	Research Article (Special Issue Paper), 2019	1	Included
	Towards evolvable software architectures based on systems theoretic stability	Research Article (Journal), 2011	1	Included
	Managing changes in requirements: an empirical investigation	Research Article (Journal), 2013	0.85	Included
	An optimization-based tool to support the cost-effective production of software architecture documentation	Research Article (Journal), 2015	0.57	Included
	Classification and comparison of architecture evolution reuse knowledge—a systematic review	Research Article (Journal), 2014	0.64	Included

Wiley & P4	Measuring stability of object-oriented software architectures	Research Article (Journal), 2014	0.78	Included
	Exploring the missing link: an empirical study of software fixes	Research Article (Journal), 2013	0.57	Included
	Exploring factors affecting decision outcome and lead time in large-scale requirements engineering	Research Article (Journal), 2015	0.64	Included
	Do code data sharing dependencies support an early prediction of software's actual change impact set?	Research Article (Journal), 2018	1	Included
	A procedural and flexible approach for specification, modeling, definition, and analysis for self-adaptive systems	Research Article (Journal), 2020	1	Included
Wiley & P5	Supporting requirements update during software evolution	Research Article (Journal), 2015	1	Included
	Software smell detection techniques: A systematic literature review	Research Article (Journal), 2020	1	Included
	Increasing software development efficiency and maintainability for complex industrial systems – A case study	Research Article (Journal), 2011	0.57	Included
	Continuous software engineering and its support by usage and decision knowledge: An interview study with practitioners	Research Article (Special Issue Paper), 2019	0.92	Included
	Patterns in System Architecture Decisions	Research Article (Journal), 2016	0.92	Included
	Analyzing runtime adaptability of collaboration patterns	Research Article (Special Issue Paper), 2014	0.92	Included
	Pattern detection and design rationale traceability: an integrated approach to software design quality	Research Article (Journal), 2018	1	Included
	An optimization-based tool to support the cost-effective production of software architecture documentation	Research Article (Journal), 2015	0.85	Included
	The impact of traceability on software maintenance and evolution: A mapping study	Review Article (Journal), 2020	0.78	Included
Science Direct & P6	Software sustainability: Research and practice from a software architecture viewpoint	Accepted Manuscript (Journal), 2017	1	Included

Science Direct & P6	Stability assessment of aspect-oriented software architectures: A quantitative study	Research Article (Journal), 2010	0.78	Included
	10 years of software architecture knowledge management: Practice and future	Accepted Manuscript (Journal), 2015	1	Included
	Generation and validation of traces between requirements and architecture based on formal trace semantics	Research Article (Journal), 2014	0.78	Included
	Early validation of system requirements and design through correctness-by-construction	Accepted Manuscript (Journal), 2018	1	Included
	Optimization of adaptation plans for a service-oriented architecture with cost, reliability, availability, and performance tradeoffs	Research Article (Journal), 2013	0.92	Included
	Empirical research for software architecture decision making: An analysis	Accepted Manuscript (Journal), 2018	1	Included
	Software architecture evolution through evolvability analysis	Research Article (Journal), 2012	1	Included
	Managing the evolution of software architecture at minimal cost underperformance and reliability constraints	Accepted Manuscript (Journal), 2014	0.85	Included
	Characterizing software architecture changes: A systematic review	Research Article (Journal), 2010	0.92	Included
Science Direct & P7	An exploratory study of architectural effects on requirements decisions	Research Article (Journal), 2010	1	Included
	A documentation framework for architecture decisions	Research Article (Journal), 2011	0.92	Included
	Accessing decision-making in software design	Research Article (Journal), 2010	1	Included
	Impact propagation and risk assessment of requirement changes for software development projects based on design structure matrix	Research Article (Journal), 2011	0.78	Included
	A systematic review of software architecture evolution research	Research Article (Journal), 2012	1	Included
	Advanced quality prediction model for software architectural knowledge sharing	Research Article (Journal), 2011	0.92	Included
	Exploiting traceability uncertainty between software architectural models and extra-functional results	Research Article (Journal), 2017	1	Included

Science Direct & P7	A survey on software architectural assumptions	Accepted Manuscript (Journal), 2015	0.71	Included
	Evaluation of a process for architectural assumption management in software development	Accepted Manuscript (Journal), 2018	0.92	Included
	Towards supporting the software architecture life cycle	Research Article (Journal), 2012	1	Included
Science Direct & P8	Scaling up software architecture analysis	Research Article (Journal), 2012	0.71	Included
	Supporting runtime software architecture: A bidirectional-transformation-based approach	Research Article (Journal), 2011	0.5	Included
	Controlling software architecture erosion: A survey	Research Article (Journal), 2012	1	Included
	Requirements traceability technologies and technology transfer decision support: A systematic review	Accepted Manuscript (Journal), 2018 2018	1	Included
	Performance measurement of models specified through component-based software architectural styles	Research Article (Journal), 2015	1	Included
	Reconciling software architecture and source code in support of software evolution	Accepted Manuscript (Journal), 2016	0.78	Included
	Bridging the gap between software architecture rationale formalisms and actual architecture documents: An ontology-driven approach	Research Article (Journal), 2012	0.64	Included
	The strengths and weaknesses of software architecture design in the RUP, MSF, MBASE and RUP-SOA methodologies: A conceptual review	Accepted Manuscript (Journal), 2016	0.71	Included
	Automatic enforcement of constraints in real-time collaborative architectural decision making	Research Article (Journal), 2015	1	Included
	Towards semi-automated assignment of software change requests	Accepted Manuscript (Journal), 2016	0.92	Included
	Towards a reduction in architectural knowledge vaporization during agile global software development	Accepted Manuscript (Journal), 2019	1	Included

	Validating a model-driven software architecture evaluation and improvement method: A family of experiments	Accepted Manuscript (Journal), 2014	1	Included
	Software architecture awareness in long-term software product evolution	Research Article (Journal), 2010	1	Included
ACM & P9	Architecture-centric support for adaptive service collaborations	Research Article (Journal), 2014	1	Included
	Decision-making techniques for software architecture design: A comparative survey	Research Article (Journal), 2011	0.71	Included
	Maintaining Architecture-Implementation Conformance to Support Architecture Centrality: From Single System to Product Line Development	Research Article (Journal), 2018	0.85	Included
	Traceability and SysML design slices to support safety inspections: A controlled experiment	Research Article (Journal), 2014	0.92	Included
	The value of design rationale information	Research Article (Journal), 2013	0.64	Included
	Ensuring the Consistency between User Requirements and Task Models: A Behavior-Based Automated Approach	Research Article (Journal), 2020	0.57	Included
	Improving Scalability and Reward of Utility-Driven Self-Healing for Large Dynamic Architectures	Research Article (Journal), 2020	1	Included
	Requirements reflection: requirements as runtime entities	Conference Paper (32nd C), 2010	1	Included

APPENDIX-C

Table C: THE DATA EXTRACTIONS FORMS OF THE CONDUCTED SLR.

Entities	Respective Data
Title	A Prescriptive Approach to Quality-Focused System Architecture
Paper ID	I-1
Type	Journal Article
Publisher	IEEE Journal of Systems
QA Score	1
Answer to RQ1	SW Design, Architecture, and SQW Maintenance
Status	Yes

Entities	Respective Data
Title	Evaluating System Architecture Quality and Architecting Team Performance Using Information Quality Theory
Paper ID	I-2
Type	Journal Article
Publisher	IEEE Journal of Systems
QA Score	0.92
Answer to RQ1	Team and Architecture
Status	Yes

Entities	Respective Data
Title	SAM-SoS: A Stochastic Software Architecture Modeling and Verification Approach for Complex System-of-Systems
Paper ID	I-3
Type	Journal Article
Publisher	IEEE Journal of Access
QA Score	0.92
Answer to RQ1	Requirement Volatility, SW Design, Quality Assurance, and Architecture.
Status	Yes

Entities	Respective Data
Title	Automatically Detecting and Tracking Inconsistencies in Software Design Models
Paper ID	I-4
Type	Journal Article
Publisher	IEEE Journal of Transactions on Software Engineering
QA Score	0.71
Answer to RQ1	SW Design
Status	Yes

Entities	Respective Data
Title	Decision-Making Assistance in Engineering Change Management Process
Paper ID	I-5
Type	Journal Article
Publisher	IEEE Journal of Transactions on Systems, Man and Cybernetics.
QA Score	0.57
Answer to RQ1	Team, Communication issues, and SW Design
Status	Yes

Entities	Respective Data
Title	Comparative Analysis of Requirement Change Management Challenges Between in-House and Global Software Development: Findings of Literature and Industry Survey
Paper ID	I-6
Type	Journal Article
Publisher	IEEE Journal of Access
QA Score	0.57
Answer to RQ1	Resource Management, Dependency, Traceability, and Documentation.
Status	Yes

Entities	Respective Data
Title	An Industrial Survey of Safety Evidence Change Impact Analysis Practice
Paper ID	I-7
Type	Journal Article
Publisher	IEEE Journal of Transactions on Software Engineering
QA Score	0.92
Answer to RQ1	SW Artefacts
Status	Yes

Entities	Respective Data
Title	Structuring Software Requirements for Architecture Design (20^{th} C)
Paper ID	I-8
Type	Conference paper
Publisher	The International Conference and Workshop on Engineering of Computer-Based Systems
QA Score	1
Answer to RQ1	Requirement Volatility, SW Design, and Architecture.
Status	Yes

Entities	Respective Data
Title	Software Architecture Matching by Meta-model Extension and Refinement (19th C)
Paper ID	I-9
Type	Conference paper
Publisher	The Asia-Pacific Conference on Software Engineering
QA Score	1
Answer to RQ1	Architecture and Requirement Volatility
Status	Yes

Entities	Respective Data
Title	Product Line Requirements Reuse Based on Variability Management (19th C)
Paper ID	I-10
Type	Conference paper
Publisher	The Asia-Pacific Conference on Software Engineering
QA Score	0.65
Answer to RQ1	Organizational leadership and dependencies
Status	Yes

Entities	Respective Data
Title	Quality-Driven Self-Adaptation: Bridging the Gap between Requirements and Runtime Architecture by Design Decision (36th C)
Paper ID	I-11
Type	Conference paper
Publisher	The International Conference on Computer Software and Applications.
QA Score	1
Answer to RQ1	Adaption to change, Quality Assurance, SW Design, and Architecture.
Status	Yes

Entities	Respective Data
Title	Priority-Awareness of Non-Functional Requirements under Uncertainty (28th C)
Paper ID	I-12
Type	Conference paper
Publisher	The International Conference on Requirements Engineering
QA Score	0.57
Answer to RQ1	Communication issues, Trade-offs, and Integration of Linkage
Status	Yes

Entities	Respective Data
Title	An initial evaluation of requirements dependency types in change propagation analysis (16th C)
Paper ID	I-13
Type	Conference paper
Publisher	The International Conference on Evaluation & Assessment in Software Engineering
QA Score	0.92
Answer to RQ1	Requirement Volatility, SW Design, Code, Stakeholder, and Dependencies
Status	Yes

Entities	Respective Data
Title	A Method of Specifying and Classifying Requirements Change (22nd C)
Paper ID	I-14
Type	Conference paper
Publisher	The Australian Conference on Software Engineering
QA Score	0.71
Answer to RQ1	Dynamic Business Environment and Communication issue
Status	Yes

Entities	Respective Data
Title	Towards requirements aware systems: Run-time resolution of design-time assumptions (26th C)
Paper ID	I-15
Type	Conference paper
Publisher	The International Conference on Automated Software Engineering (ASE)
QA Score	0.85
Answer to RQ1	Stakeholder and SW Design
Status	Yes

Entities	Respective Data
Title	RADAR: A Lightweight Tool for Requirements and Architecture Decision Analysis (39th C)
Paper ID	I-16
Type	Conference paper
Publisher	The International Conference on Software Engineering (ICSE)
QA Score	0.78
Answer to RQ1	Requirement Volatility, Stakeholder, and Architecture.
Status	Yes

Entities	Respective Data
Title	Towards search-based modeling and analysis of requirements and architecture decisions (32th C)
Paper ID	I-17
Type	Conference paper
Publisher	The International Conference on Automated Software Engineering (ASE)
QA Score	0.78
Answer to RQ1	Requirement Volatility and Stakeholder.
Status	Yes

Entities	Respective Data
Title	Run-time Resolution of Uncertainty (19th C)
Paper ID	I-18
Type	Conference paper
Publisher	The International Conference on Requirements Engineering
QA Score	0.92
Answer to RQ1	Requirement Volatility and SW Design
Status	Yes

Entities	Respective Data
Title	Assessing Architectural Patterns Trade-offs using Moment-based Pattern Taxonomies (45th C)
Paper ID	I-19
Type	Conference paper
Publisher	The Latin American Computing Conference (CLEI)
QA Score	0.57
Answer to RQ1	Knowledge, Traceability, and Trade-off.
Status	Yes

Entities	Respective Data
Title	Improving Software Performance and Reliability with an Architecture-Based Self-Adaptive Framework (34th C)
Paper ID	I-20
Type	Conference paper
Publisher	The Annual International Computer Software and Applications Conference (COMPSAC)
QA Score	0.57
Answer to RQ1	Architecture
Status	Yes

Entities	Respective Data
Title	Managing Design Time Uncertainty (20th C)
Paper ID	I-21
Type	Conference paper
Publisher	IEEE International Conference on Model-Driven Engineering Languages and Systems (MODELS)
QA Score	0.92
Answer to RQ1	Knowledge and SW Design
Status	Yes

Entities	Respective Data
Title	Reducing Uncertainty in Architectural Decisions with AADL (44th C)
Paper ID	I-22
Type	Conference paper
Publisher	Annual Hawaii International Conference on System Sciences (HICSS)
QA Score	0.71
Answer to RQ1	Architectural Complexity
Status	Yes

Entities	Respective Data
Title	Inconsistency Management between Architectural Decisions and Designs Using Constraints and Model Fixes (23rd C)
Paper ID	I-23
Type	Conference paper
Publisher	The Australian Conference on Software Engineering
QA Score	0.78
Answer to RQ1	SW Design
Status	Yes

Entities	Respective Data
Title	Impact of requirements volatility on software architecture: How do software teams keep up with ever-changing requirements?
Paper ID	W-1
Type	Special Issue Paper
Publisher	Journal of Software: Practice & Experience
QA Score	1
Answer to RQ1	Requirement Volatility, SW Defects, Resource Management, Communication Issues, Documentation, Dependencies, and SW Architecture
Status	Yes

Entities	Respective Data
Title	Towards evolvable software architectures based on systems theoretic stability
Paper ID	W-2
Type	Research Article
Publisher	Journal of Software: Practice & Experience
QA Score	1
Answer to RQ1	Requirement Volatility and Architecture
Status	Yes

Entities	Respective Data
Title	Managing changes in requirements: an empirical investigation
Paper ID	W-3
Type	Journal: Research Article
Publisher	Journal of Software: Evolution & Process
QA Score	0.85
Answer to RQ1	Requirement Volatility
Status	Yes

Entities	Respective Data
Title	An optimization-based tool to support the cost-effective production of software architecture documentation
Paper ID	W-4
Type	Journal: Research Article
Publisher	Journal of Software: Evolution & Process
QA Score	0.57
Answer to RQ1	Resource Management, Documentation, Stakeholder, and Architectural Complexity.
Status	Yes

Entities	Respective Data
Title	Classification and comparison of architecture evolution reuse knowledge—a systematic review
Paper ID	W-5
Type	Journal: Research Article
Publisher	Journal of Software: Evolution & Process
QA Score	0.64
Answer to RQ1	Architecture
Status	Yes

Entities	Respective Data
Title	Measuring stability of object-oriented software architectures
Paper ID	W-6
Type	Journal: Research Article
Publisher	IET Software
QA Score	0.78
Answer to RQ1	Architecture
Status	Yes

Entities	Respective Data
Title	Exploring the missing link: an empirical study of software fixes
Paper ID	W-7
Type	Journal: Research Article
Publisher	Journal of Software: Software Testing, Verification & Reliability
QA Score	0.57
Answer to RQ1	SW Defects
Status	Yes

Entities	Respective Data
Title	Exploring factors affecting decision outcome and lead time in large-scale requirements engineering
Paper ID	W-8
Type	Journal: Research Article
Publisher	Journal of Software: Evolution & Process
QA Score	0.64
Answer to RQ1	Resource Management and Knowledge
Status	Yes

Entities	Respective Data
Title	Do code data sharing dependencies support an early prediction of software's actual change impact set?
Paper ID	W-9
Type	Research Article
Publisher	Journal of Software: Evolution & Process
QA Score	1
Answer to RQ1	Dependency, Requirement Volatility, and Code
Status	Yes

Entities	Respective Data
Title	A procedural and flexible approach for specification, modeling, definition, and analysis for self-adaptive systems
Paper ID	W-10
Type	Journal: Research Article
Publisher	Software: Practice & Experience
QA Score	1
Answer to RQ1	Adaption to Change and SW Design
Status	Yes

Entities	Respective Data
Title	Supporting requirements update during software evolution
Paper ID	W-11
Type	Journal: Research Article
Publisher	Journal of Software: Evolution & Process
QA Score	1
Answer to RQ1	Requirement Volatility and Code.
Status	Yes

Entities	Respective Data
Title	Software smell detection techniques: A systematic literature review
Paper ID	W-12
Type	Journal: Review Article
Publisher	Journal of Software: Evolution & Process
QA Score	1
Answer to RQ1	Code and SW Design
Status	Yes

Entities	Respective Data
Title	Increasing software development efficiency and maintainability for complex industrial systems – A case study
Paper ID	W-13
Type	Journal: Research Article
Publisher	Journal of Software: Process & Evolution
QA Score	0.57
Answer to RQ1	Architectural Complexity and SQW Maintenance.
Status	Yes

Entities	Respective Data
Title	Continuous software engineering and its support by usage and decision knowledge: An interview study with practitioners
Paper ID	W-14
Type	Journal: Special Issue Paper
Publisher	Journal of Software: Evolution & Process
QA Score	0.92
Answer to RQ1	Integration of Usage, Knowledge, and Traceability
Status	Yes

Entities	Respective Data
Title	Patterns in System Architecture Decisions
Paper ID	W-15
Type	Journal: Research Article
Publisher	The Journal of the International Council on System & Engineering
QA Score	0.92
Answer to RQ1	Knowledge and Architecture
Status	Yes

Entities	Respective Data
Title	Analyzing runtime adaptability of collaboration patterns
Paper ID	W-16
Type	Journal: Special Issue Paper
Publisher	Concurrency & Computation: Practice & Experience
QA Score	0.92
Answer to RQ1	Adaption to change and Communication Issues
Status	Yes

Entities	Respective Data
Title	Pattern detection and design rationale traceability: an integrated approach to software design quality
Paper ID	W-17
Type	Journal: Research Article
Publisher	IET Software
QA Score	1
Answer to RQ1	SW Design and Traceability
Status	Yes

Entities	Respective Data
Title	An optimization-based tool to support the cost-effective production of software architecture documentation
Paper ID	W-18
Type	Journal: Research Article
Publisher	Journal of Software: Evolution & Process
QA Score	1
Answer to RQ1	Resource Management, Stakeholder, Documentation, and SW Architecture.
Status	Yes

Entities	Respective Data
Title	The impact of traceability on software maintenance and evolution: A mapping study
Paper ID	W-19
Type	Journal: Review Article
Publisher	Journal of Software: Evolution & Process
QA Score	0.78
Answer to RQ1	Traceability and SQW Maintenance.
Status	Yes

Entities	**Respective Data**
Title	Software sustainability: Research and practice from a software architecture viewpoint
Paper ID	SD-1
Type	Journal: Accepted Manuscript
Publisher	Elsevier: The Journal of System & Software
QA Score	1
Answer to RQ1	Technical Debt, Code, and Architecture
Status	Yes

Entities	**Respective Data**
Title	Stability assessment of aspect-oriented software architectures: A quantitative study
Paper ID	SD-2
Type	Journal: Research Article
Publisher	Elsevier: The Journal of System & Software
QA Score	0.78
Answer to RQ1	Architectural Complexity
Status	Yes

Entities	**Respective Data**
Title	10 years of software architecture knowledge management: Practice and future
Paper ID	SD-3
Type	Journal: Accepted Manuscript
Publisher	Elsevier: The Journal of System & Software
QA Score	1
Answer to RQ1	Knowledge
Status	Yes

Entities	Respective Data
Title	Generation and validation of traces between requirements and architecture based on formal trace semantics
Paper ID	SD-4
Type	Journal: Research Article
Publisher	Elsevier: The Journal of System & Software
QA Score	0.78
Answer to RQ1	Tractability and Architecture
Status	Yes

Entities	Respective Data
Title	Early validation of system requirements and design through correctness-by-construction
Paper ID	SD-5
Type	Journal: Accepted Manuscript
Publisher	Elsevier: The Journal of System & Software
QA Score	1
Answer to RQ1	SW Design
Status	Yes

Entities	Respective Data
Title	Optimization of adaptation plans for a service-oriented architecture with cost, reliability, availability, and performance tradeoff
Paper ID	SD-6
Type	Journal: Research Article
Publisher	Elsevier: The Journal of System & Software
QA Score	0.92
Answer to RQ1	Adaption to Change and Architecture
Status	Yes

Entities	Respective Data
Title	Empirical research for software architecture decision making: An analysis
Paper ID	SD-7
Type	Journal: Accepted Manuscript
Publisher	Elsevier: The Journal of System & Software
QA Score	1
Answer to RQ1	Human Behavior and Architecture
Status	Yes

Entities	Respective Data
Title	Software architecture evolution through evolvability analysis
Paper ID	SD-8
Type	Journal: Research Article
Publisher	Elsevier: The Journal of System & Software
QA Score	1
Answer to RQ1	Architecture
Status	Yes

Entities	Respective Data
Title	Managing the evolution of software architecture at minimal cost underperformance and reliability constraints
Paper ID	SD-9
Type	Journal: Accepted Manuscript
Publisher	Elsevier: Science of Computer Programming
QA Score	0.85
Answer to RQ1	Architecture and Human Behavior
Status	Yes

Entities	Respective Data
Title	Characterizing software architecture changes: A systematic review
Paper ID	SD-10
Type	Journal: Research Article
Publisher	Elsevier: Information & Software Technology
QA Score	0.92
Answer to RQ1	Requirement Volatility and Architecture
Status	Yes

Entities	Respective Data
Title	An exploratory study of architectural effects on requirements decisions
Paper ID	SD-11
Type	Journal: Research Article
Publisher	Elsevier: The Journal of System & Software
QA Score	1
Answer to RQ1	Requirement Volatility, Knowledge, and Architecture.
Status	Yes

Entities	Respective Data
Title	A documentation framework for architecture decisions
Paper ID	SD-12
Type	Journal: Research Article
Publisher	Elsevier: The Journal of System & Software
QA Score	0.92
Answer to RQ1	Stakeholder and Architecture
Status	Yes

Entities	Respective Data
Title	Accessing decision-making in software design
Paper ID	SD-13
Type	Journal: Research Article
Publisher	Elsevier: Design Studies
QA Score	1
Answer to RQ1	SW Design and Team
Status	Yes

Entities	Respective Data
Title	Impact propagation and risk assessment of requirement changes for software development projects based on design structure matrix
Paper ID	SD-14
Type	Journal: Research Article
Publisher	Elsevier: International Journal of Project Management
QA Score	0.78
Answer to RQ1	Requirement Volatility and Architecture
Status	Yes

Entities	Respective Data
Title	A systematic review of software architecture evolution research
Paper ID	SD-15
Type	Journal: Research Article
Publisher	Elsevier: Information & Software Technology
QA Score	1
Answer to RQ1	Architecture
Status	Yes

Entities	Respective Data
Title	Advanced quality prediction model for software architectural knowledge sharing
Paper ID	SD-16
Type	Journal: Research Article
Publisher	Elsevier: The Journal of Systems & Software
QA Score	0.92
Answer to RQ1	Knowledge and Architecture
Status	Yes

Entities	Respective Data
Title	Exploiting traceability uncertainty between software architectural models and extra-functional results
Paper ID	SD-17
Type	Journal: Research Article
Publisher	Elsevier: The Journal of System & Software
QA Score	1
Answer to RQ1	Traceability and Security
Status	Yes

Entities	Respective Data
Title	A survey on software architectural assumptions
Paper ID	SD-18
Type	Journal: Accepted Manuscript
Publisher	Elsevier: The Journal of System & Software
QA Score	1
Answer to RQ1	Architecture
Status	Yes

Entities	Respective Data
Title	Evaluation of a process for architectural assumption management in software development
Paper ID	SD-19
Type	Journal: Accepted Manuscript
Publisher	Elsevier: Science of Computer Programmer
QA Score	0.92
Answer to RQ1	Architecture
Status	Yes

Entities	Respective Data
Title	Towards supporting the software architecture life cycle
Paper ID	SD-20
Type	Journal: Research Article
Publisher	Elsevier: The Journal of System & Software
QA Score	1
Answer to RQ1	Architecture and SW Design
Status	Yes

Entities	Respective Data
Title	Scaling up software architecture analysis
Paper ID	SD-21
Type	Journal: Research Article
Publisher	Elsevier: The Journal of System & Software
QA Score	0.71
Answer to RQ1	Architectural Complexity
Status	Yes

Entities	Respective Data
Title	Supporting runtime software architecture: A bidirectional-transformation-based approach
Paper ID	SD-22
Type	Journal: Research Article
Publisher	Elsevier: The Journal of Systems & Software
QA Score	0.5
Answer to RQ1	Requirement Volatility and Architecture
Status	Yes

Entities	Respective Data
Title	Controlling software architecture erosion: A survey
Paper ID	SD-23
Type	Journal: Research Article
Publisher	Elsevier: The Journal of Systems & Software
QA Score	1
Answer to RQ1	Adaption to Change and Architecture
Status	Yes

Entities	Respective Data
Title	Requirements traceability technologies and technology transfer decision support: A systematic review
Paper ID	SD-24
Type	Journal: Accepted Manuscript
Publisher	Elsevier: the Journal of System & Software
QA Score	1
Answer to RQ1	Traceability
Status	Yes

Entities	Respective Data
Title	Performance measurement of models specified through component-based software architectural styles
Paper ID	SD-25
Type	Journal: Research Articles
Publisher	Elsevier: Measurement
QA Score	1
Answer to RQ1	Architectural Complexity
Status	Yes

Entities	Respective Data
Title	Reconciling software architecture and source code in support of software evolution
Paper ID	SD-26
Type	Journal: Accepted Manuscript
Publisher	Elsevier: The Journal of System & Software
QA Score	0.78
Answer to RQ1	Architecture and Code
Status	Yes

Entities	Respective Data
Title	Bridging the gap between software architecture rationale formalisms and actual architecture documents: An ontology-driven approach
Paper ID	SD-27
Type	Journal: Research Article
Publisher	Elsevier: Science of Computer Programming
QA Score	0.64
Answer to RQ1	Architecture and Documentation
Status	Yes

Entities	Respective Data
Title	The strengths and weaknesses of software architecture design in the RUP, MSF, MBASE, and RUP-SOA methodologies: A conceptual review
Paper ID	SD-28
Type	Journal: Accepted Manuscript
Publisher	Elsevier: Computer Standards & Interfaces
QA Score	0.71
Answer to RQ1	SW Design and Architecture
Status	Yes

Entities	Respective Data
Title	Automatic enforcement of constraints in real-time collaborative architectural decision making
Paper ID	SD-29
Type	Journal: Research Article
Publisher	Elsevier: The Journal of Systems & Software
QA Score	1
Answer to RQ1	Knowledge, Trade-off, Stakeholder, and Architectural complexity
Status	Yes

Entities	Respective Data
Title	Towards semi-automated assignment of software change requests
Paper ID	SD-30
Type	Journal: Accepted Manuscript
Publisher	Elsevier: The Journal of Systems & Software
QA Score	0.92
Answer to RQ1	Requirement Volatility
Status	Yes

Entities	Respective Data
Title	Towards a reduction in architectural knowledge vaporization during agile global software development
Paper ID	SD-31
Type	Journal: Accepted Manuscript
Publisher	Elsevier: Information & Software Technology
QA Score	1
Answer to RQ1	Knowledge and architecture
Status	Yes

Entities	Respective Data
Title	Validating a model-driven software architecture evaluation and improvement method: A family of experiments
Paper ID	SD-32
Type	Journal: Accepted Manuscript
Publisher	Elsevier: Information & Software Technology
QA Score	1
Answer to RQ1	Architecture
Status	Yes

Entities	Respective Data
Title	Software architecture awareness in long-term software product evolution
Paper ID	SD-33
Type	Journal: Research Article
Publisher	Elsevier: The Journal of Systems & Software
QA Score	1
Answer to RQ1	Architecture, Communication, and Knowledge
Status	Yes

Entities	Respective Data
Title	Architecture-centric support for adaptive service collaborations
Paper ID	A-1
Type	Journal: Research Article
Publisher	ACM Transactions on Software Engineering and Methodology
QA Score	1
Answer to RQ1	Dynamic Business Environment, Communication issues, SW Design, Architectural Complexity, dependencies, and SW defects.
Status	Yes

Entities	Respective Data
Title	Decision-making techniques for software architecture design: A comparative survey
Paper ID	A-2
Type	Journal: Research Article
Publisher	ACM Computing Surveys
QA Score	0.71
Answer to RQ1	Architecture, SW Design, and Trade-off.
Status	Yes

Entities	Respective Data
Title	Maintaining Architecture-Implementation Conformance to Support Architecture Centrality: From Single System to Product Line Development
Paper ID	A-3
Type	Journal: Research Article
Publisher	ACM Transactions on Software Engineering and Methodology (TOSEM)
QA Score	0.85
Answer to RQ1	Architectural Complexity, Code, and SQW Maintenance.
Status	Yes

Entities	Respective Data
Title	Traceability and SysML design slices to support safety inspections: A controlled experiment
Paper ID	A-4
Type	Journal: Research Article
Publisher	ACM Transactions on Software Engineering and Methodology (TOSEM)
QA Score	0.92
Answer to RQ1	SW Design, Traceability, SW Defects
Status	Yes

Entities	Respective Data
Title	The value of design rationale information
Paper ID	A-5
Type	Journal: Research Article
Publisher	ACM Transactions on Software Engineering and Methodology (TOSEM)
QA Score	0.64
Answer to RQ1	SW Design or Design Implementation, Documentation
Status	Yes

Entities	Respective Data
Title	Ensuring the Consistency between User Requirements and Task Models: A Behavior-Based Automated Approach
Paper ID	A-6
Type	Journal: Research Article
Publisher	Proceedings of the ACM on Human-Computer Interaction (PACMHCI)
QA Score	0.57
Answer to RQ1	Stakeholder, Human Behavior, SW Design, and SW Artefacts.
Status	Yes

Entities	Respective Data
Title	Improving Scalability and Reward of Utility-Driven Self-Healing for Large Dynamic Architectures
Paper ID	A-7
Type	Journal: Research Article
Publisher	ACM Transactions on Autonomous and Adaptive Systems
QA Score	1
Answer to RQ1	Self-Healing Mechanism
Status	Yes

Entities	Respective Data
Title	Requirements reflection: requirements as runtime entities (32th C)
Paper ID	A-8
Type	Conference paper
Publisher	ACM 32nd International Conference on Software Engineering
QA Score	1
Answer to RQ1	Software Architecture, SW Artefacts, and Requirement Volatility.
Status	Yes

APPENDIX-D

Table D: EXECUTION OF DATA ENCODING TECHNIQUE

Paper ID	Paper Statement	Respective Code	Data Encoding
I1	"The most critical requirements for the lifetime value of a system are its nonfunctional requirements (NFRs) such as reliability, security, maintainability, changeability, etc. These are collectively known as the "ilities," and they are typically not addressed in system design until the functional architecture has been completed."	I1L4, I1L8, I1L7	Maintenance AND Architecture AND SW Design
I2	"As engineering projects grow in complexity, estimating the required engineering effort during the development phase of a project has become more art than science. Predictions of required engineering effort are based on empirical models fitted to historical data with additional subjective factors, such as "team cohesion," applied based on the judgment of the user of the mode."	I2L9&10	Team Cohesion
I3	"Dynamic system who's Constituent Systems (CSs) is not known precisely at design time, and the environment in which they operate is uncertain. Moreover, unknown conditions and volatility have significant effects on crucial Quality Attributes (QAs) such as performance, reliability and security."	I3L3, I3L5, I3L6	SW Design AND Volatility AND Quality Assurance
I4	"Software models typically contain many inconsistencies and consistency checkers help engineers find them. Even if engineers are willing to tolerate inconsistencies, they are better off knowing about their existence to avoid follow-on errors and unnecessary rework. "	I4L1	SW Implementation OR Design
I5	"Both the integration of the various tools intervening in the life cycle of a product and the management of the communication between the various multidisciplinary teams working on a product are difficult tasks. "	I5L1, I5L4, I5L5	Design Implementation AND Communication Issues AND Team
I6	"The survey result indicates that there are four out of nine challenges, namely impact analysis, requirement traceability,	I6L3, I6L4, I6L7,	Traceability AND Dependency

	requirement dependency, and system instability having the same impact in both in-house and GSD approaches. On the other hand, cost/time estimation, artifacts documents management, user involvement, requirement consistency, and requirement prioritization need more attention while implemented in GSD paradigm.“	I6L8	AND Cost Estimation AND Documentation
I7	“In many application domains, critical systems must comply with safety standards. This involves gathering safety evidence in the form of artefacts such as safety analyses, system specifications, and testing results.”	I7L4	SW Artefacts
I8	“Characteristics of individual requirements (e.g. the complexity or volatility of requirement) also impact the design of architectures. Consequently, systematically handling the impact of individual requirements on the architecture can facilitate the design of architectures.”	I8L2 AND I8L7	Requirement Volatility AND Design Implementation AND Architecture
I9	“Nowadays, software runs in an open, dynamic and changeable environment, which requires the SA should be dynamic and able to adapt to changes. The inconsistency of the software architecture caused by adapting to changes may lead to architecture mismatching, which becomes a new challenge for the software development. “	I9L4, I9L7	Requirement Changes OR Requirement Volatility AND Architecture
I10	“As organizations respond to changing environments new software products emerge as a compromise between customer requirements, extensions of existing products and commercial needs. “	I10L1, I10L3	Organizational Leadership AND Customer Needs
I11	“Running with static requirements and design decisions, a software system cannot always perform optimally in a highly uncertain and rapidly changing environment. Quality-driven self-adaptation, which enables Software system to continually adapt its structure and behavior to improve the overall quality satisfaction, thus becomes a promising capability of software systems. “	I11L7, I11L8, I11L10	Adaption to Change AND Quality Assurance SW Design OR Architecture
I12	“As the run-time context changes, the system may need to re-configure itself, and since resources are finite this may require trade-offs between the SAS’s non-functional requirements (NFRs). A number	I12L4, I12L9	Trade-Off AND Lack of Explicit Linkage AD Requirements

	of runtime modelling techniques have been developed for resolution of uncertainty in SASs. However, current techniques lack the explicit runtime representation of NFR priorities, leading to the risk that adaptations may fail to respect the NFRs' priorities. "		
I13	"Change propagation analysis helps predict the parts of the software that may be affected if a change is made. Existing research on change propagation focuses on design and code level changes. However, as software evolves, the requirements that drive these changes also have intricate dependencies."	I13L3, I13L5, I13L8	Requirement change AND SW Design AND Code AND Dependencies
I14	"Change is one aspect of business that is inevitable. The volatile nature of business requirements is considered one of the main contributors to information technology project failure. One of the key reasons for difficulty in managing change is the lack of adequate methods to communicate change from business to the IT department."	I14l2&3, I14L7	Dynamic Business Environment AND Communication Issues
I15	"In earlier work we proposed the idea of Requirements-aware systems that could introspect about the extent to which their goals were being satisfied at runtime."	I15L4	Stakeholder Goals
I16	"Uncertainty and conflicting stakeholders' objectives make many requirements and architecture decisions particularly hard. Quantitative probabilistic models allow software architects to analyze such decisions using stochastic simulation and multi- objective optimization, but the difficulty of elaborating the models is an obstacle to the wider adoption of such techniques."	I16L1 AND I16L3	Stakeholders Objectives AND Architecture
I17	"Many requirements engineering and software architecture decisions are complicated by uncertainty and multiple conflicting stakeholders objectives. Using quantitative decision models helps clarify these decisions and allows the use of multi-objective simulation optimization techniques in analyzing the impact of decisions on objectives. "	I17L2, I17L4	Decision Knowledge AND Stakeholder Objectives
I18	"Requirements awareness should help optimize requirements satisfaction when factors that were uncertain at design time are resolved at runtime."	I18L	Requirement Volatility AND SW Design

I19	"Large software systems are designed to satisfy or accommodate many requirements; architectural patterns are a well-known technique to reuse design knowledge. Thus, a key concern of systems architects is understanding trade-offs among alternative solutions; e.g., a pattern may favor performance at the expense of scalability or security, another may privilege scalability, and yet another may push security."	I19L5, I19L3, I19L6	Knowledge AND Traceability AND Trade-off
I20	"A self- adaptive approach that integrates monitoring, analyzing, and actuation functionalities has the potential to accommodate to a dynamically changing environment. The main objective of this paper is to develop an architecture-based self-adaptive framework to improve performance and resource efficiency of a server while maintaining reliable services."	I12L6	Architecture
I21	"Any software system is the accumulated result of many design decisions taken by its developers. During the course of development, however, developers are often uncertain about how to make these decisions. This uncertainty reflects lack of knowledge about the design of the system, rather than about the environment in which the system is intended to operate. It is therefore called design-time uncertainty, and is different from environmental uncertainty."	I21L2 I21L7, I21L10	Decision Knowledge AND SW Design OR Design Implementations
I22	"A model-driven approach to real-time software systems development enables the conceptualization of software, fostering a more thorough understanding of its often complex architecture and behavior and promoting the documentation and analysis of concerns common to real-time embedded systems such as scheduling, resource allocation, and performance. "	I22L5, I22L6, I22L9	Architecture Complexity AND Documentation And Resource management
I23	"The software architecture community has proposed to document the design rationale of software architectures by means of architectural design decisions."	I23L	SW Design OR Design Implementations
W1	"Requirements volatility is a major issue in software development, causing problems such as higher defect density, project delays, and cost overruns. Software architecture that guides the overall vision of	W1L3, W1L4, W1L6	Higher defect density AND Resource management AND Architecture

	software product is one of the areas that is greatly affected by requirements volatility.”		
W2	“In today's increasingly volatile environments, evolvability is quickly becoming the most desirable characteristic of information systems. Current information systems still struggle to provide these high levels of evolvability. Based on the concept of stability from systems theory, we require that information systems should be stable with respect to a set of anticipated changes in order to exhibit high evolvability.”	W2L1, W2L2	Volatility AND Anticipated changes
W3	“This paper describes the challenges of handling changing requirements in software companies. This empirical investigation deals with the different sources of changes and with the different approaches to requirements evolution.”	W3L2, W3L5	Requirements AND Changes user needs
W4	“Some of the challenges faced by most software projects are tight budget constraints and schedules, which often make managers and developers prioritize the delivery of a functional product over other engineering activities, such as software documentation. In particular, having little or low-quality documentation of the software architecture of a system can have negative consequences for the project, as the architecture is the main container of the key design decisions to fulfill the stakeholders' goals.”	W4L2&3, W4L8, W4L12, W4L14	Budget constraints AND Low quality documentation AND Architecture AND Stakeholder
W5	“The existing research and practices for ACSE primarily focus on design-time evolution and runtime adaptations to accommodate changing requirements in existing architectures.”	W5L2, W5L5	Design Issues AND Architecture
W6	“The software architecture represents those design decisions that are hardest to change. Stability in this context means preserving cross-architectural components communications and structural relationships unchanged.”	W6L2, W6L4	Design Decisions AND Architectural crosscutting concerns
W7	“Many papers have been published on analysis and prediction of software faults and/or failures, but few addressed the software fixes made to correct the faults and prevent failures from reoccurring. Furthermore, the types of fixed software were highly correlated with fault type and	W7L2, W7L9	SW Fault AND Pre-release failure AND Post release failure

	they had different distributions for prerelease and post-release failures."		
W8	"Minimizing lead time allows software companies to focus their resources on the most profitable functionality and enables them to remain competitive within the quickly changing software market."	W8L2	Resource Management
W9	"Existing studies have shown that structural dependencies within code are good predictors for code actual change impact set—a set of entities that repeatedly changing together to ensure a consistent and complete change."	W9L2, W9L3	Dependencies AND Code Change
W10	"An adaptive system can modify its settings at runtime as a response to changes in its operational environment. To analyze this kind of systems at design time is a difficult task since it requires considering the system together with the adaptation operations, and taking into account how such adaptations act on the system."	W10L1&L8, W10L4	Adaption to change AND Design issues
W11	"Updating the requirements specification when software systems evolve is a manual task that is expensive and time consuming. Therefore, maintainers usually apply the changes to the code directly and leave the requirements unchanged."	W11L1, W11L5	Requirement Specification AND Code changes
W12	"Software smells indicate design or code issues that might degrade the evolution and maintenance of software systems. Detecting and identifying these issues are challenging tasks. This paper explores, identifies, and analyzes the existing software smell detection techniques at design and code levels."	W12L1, W12L2	Design Issues AND Code Issues
W13	"It is difficult to manage complex software systems. Thus, many research initiatives focus on how to improve software development efficiency and maintainability. However, the trend in the industry is still alarming, software development projects fail, and maintenance is becoming more and more expensive. One problem could be that research has been focusing on the wrong things. Most research publications address either process improvements or architectural improvements."	W13L5, W13L13	SW Maintenance AND Architecture

W14	"The integration of usage and decision knowledge into CSE, practitioners perceives accountability and traceability as major benefits, while raising concerns about its feasibility and user groups. Study conclude that CSE remains partially difficult to capture for practitioners, while their attitude toward integrating usage and decision knowledge into CSE is positive."	W14L1, W14L2, 4W14L3	Integration of usage AND Decision Knowledge AND Traceability
W15	"This paper proposes a set of six canonical classes of architectural decisions derived from the tasks described in the system architecture body of knowledge and from real system architecture problems. These patterns can be useful in modeling architectural decisions in a wide range of complex engineering systems."	W15L1&2, W15L5	Architectural Decision AND Knowledge
W16	"A system's provided user-centric communication and coordination mechanism have a significant impact on its runtime management. Hence, it is highly important for a system designer to becoming aware of the most suitable interaction mechanism and their implications on system adaptability."	W16L2, W16L8	Communication OR Interaction Mechanism AND Adaption
W17	"Ambiguous representation of design rationale goals is just one of the many limitations that contribute to the intricacy of design patterns; thereby this research aims to introduce an approach to support the structuring, evaluation, and analysis of design patterns."	W17L1, W17L4	Design Implementations AND Design Patterns
W18	"The budget constraints and schedule issues push developers to prioritize their engineering activities. Such as documentations. Whereby the architecture is the main part which deals the software design to fulfill the stakeholder goal."	W18L1, W18L6, W18L4	Budget and Schedule issues AND Stakeholder AND Documentation
W19	"Software traceability plays a critical role in software maintenance and evolution."	W19L1, W19L2	Traceability AND SW Maintenance
SD1	"From a software architecture perspective, this allows several issues to overlap including, but not limited to: the accumulation of technical debt design decisions of individual components and systems leading to coupling and cohesion issues; sustainability debt and the broader cumulative effects of flawed architectural	SD1L4, SD1L9, SD1L10	Technical Debt AND Code Smell AND Architecture

	design choices over time resulting in code smells, architectural brittleness, erosion, and drift, which ultimately lead to decay and software death."		
SD2	"Design of stable software architectures has increasingly been a deep challenge to software developers due to the high volatility of their concerns and respective design decisions. Architecture stability is the ability of the high-level design units to sustain their modularity properties and not succumb to modifications. Architectural aspects are new modularity units aimed at improving design stability through the modularization of otherwise crosscutting concerns."	SD2L5	Architectural complexity
SD3	"The importance of architectural knowledge (AK) management for software development has been highlighted over the past ten years, where a significant amount of research has been done."	SD3L1	Knowledge
SD4	"Less attention has been paid to relating requirements (R) with architecture (A) by using well-defined semantics of traces. Traces between R&A might be manually assigned. This is time-consuming, and error prone. Traces might be incomplete and invalid."	SD4L3, SD4L2	Traceability AND Architecture
SD5	"This rigorous design takes place through the incremental construction of a model using the BIP (Behavior-Interaction-Priorities) component framework. It allows building complex designs by composing simpler reusable designs enforcing given properties."	SD5L1	SW Design
SD6	"However, service adaptations often do not consider software quality attributes and, if they do, they rely on a single attribute in isolation. In this paper, we present an optimization model, which aims to minimize the adaptation costs of a Service-Oriented Architecture (SOA), in correspondence with a certain change scenario (i.e., a set of new requirements) under reliability, availability and performance tradeoff."	SD6L1, SD6L7	Adaption to Change AND Architecture
SD7	"Despite past empirical research in software architecture decision making, we have not yet systematically studied how to perform such empirical research. Software	SD7L5, SD7L4	Human Behavior AND Architecture

	architecture decision making involves humans, their behavioral issues and practice.”		
SD8	“We describe software architecture evolution characterization, and propose an architecture evolvability analysis process that provides replicable techniques for performing activities to aim at understanding and supporting software architecture evolution. “	SD8L2	Architecture
SD9	“Managing software architecture after the deployment phase is a very complex task due to frequent changes in the software requirements and environment. The software architecture must evolve in order to tackle such changes. The goal of this paper is to provide support for the decisions that software architects make after deployment.”	SD9L1&5	Architecture
SD10	“With today’s ever increasing demands on software, software developers must produce software that can be changed without the risk of degrading the software architecture. One way to address software changes is to characterize their causes and effects. A software change characterization mechanism allows developers to characterize the effects of a change using different criteria, e.g. the cause of the change, the type of change that needs to be made, and the part of the system where the change must take place.”	SD10L1, SD10L4, SD10L10	Requirement volatility AND Architecture
SD11	“The question of the “manner in which existing software architecture affects requirements decision- making” is considered important in the research community; however, to our knowledge, this issue has not been scientifically explored. We do not know, for example, the characteristics of such architectural effects.’	SD11L2, SD11L5	Architecture AND Requirement Volatility AND Knowledge
SD12	“We introduce a documentation framework for architecture decisions. The four viewpoints, a Decision Detail viewpoint, a Decision Relationship viewpoint, a Decision Chronology viewpoint, and a Decision Stakeholder Involvement viewpoint satisfy several stakeholder concerns related to architecture decision management. “	SD12L2, SD12L5	Architecture AND Stakeholder

SD13	"A descriptive model of decision-making, developed by the authors, has been used to analyze the protocols of the three software design teams. The results give insight in how software designers process their activities, on the influence of individual or team differences, and what the consequences for their outcomes are."	SD13L3, SD13L4	SW Design AND Team
SD14	"This paper predicts the risk of change propagation in terms of change propagation probability and change impact. First, the process of software requirement changes is discussed. Then, a probabilistic model based on design structure matrix (DSM) is established to evaluate the risk of change propagation from requirements to software architecture."	SD14L4 & SD14L9	Requirement Volatility AND Architecture
SD15	"However, no systematic review has been conducted previously to provide an extensive overview of software architecture evolvablity research."	SD15L3	Architecture
SD16	"In the field of software architecture, a paradigm shift is occurring from describing the outcome of architecting process to describing the Architectural Knowledge (AK) created and used during architecting. "	SD16L1, SD16L4	Architecture AND Knowledge
SD17	"The goal of this paper is to automate the traceability between software architectural models and extra- functional results, such as performance and security, by investigating the uncertainty while bridging these two domains."	SD17L2 SD17L4	Traceability AND Security
SD18	"Managing architectural assumptions (AA) during the software lifecycle, as an important type of architecture knowledge, are critical to the success of projects."	SD18L1	Architectural Assumptions
SD19	"Architectural assumption management is critical to the success of software development projects."	SD19L1	Architectural Assumptions
SD20	"Software architecture is a central element during the whole software life cycle. Among other things, software architecture is used for communication and documentation, for design, for reasoning about important system properties, and as a blueprint for system implementation."	SD20L1, SD20L5	Architecture AND SW Design

SD21	"This paper will show how architecture design and analysis techniques rest on a small number of foundational principles. We will show how those principles have been instantiated as a core set of techniques."	SD21L1	Architectural Complexity
SD22	"Runtime software architectures (RSA) are architecture-level, dynamic representations of running software systems, which help monitor and adapt the systems at a high abstraction level. The key issue to support RSA is to maintain the causal connection between the architecture and the system, ensuring that the architecture represents the current system, and the modifications on the architecture cause proper system changes."	SD22L1, SD22L10	Architecture AND Requirement Volatility
SD23	"As the potential frequency and scale of software adaptations increase to meet rapidly changing requirements and business conditions, controlling such architecture erosion becomes an important concern for software architects and developers."	SD23L2, SD23L4	Adaption to change AND Architecture
SD24	"Requirements traceability (RT) is a core activity in Requirements Engineering. Various types of RT technologies have been extensively studied for decades."	SD24L1	Traceability
SD25	"Measuring the performance related properties at the architectural level and before implementation, is very important while designing complex software systems."	SD25L2	Architectural Complexity
SD26	"Even in the eighties, the need of managing software evolution has been detected as one of the most complex aspects of the software lifecycle. In this context, software architecture has been highlighted as an integral element of the software evolution process. However, no matter how much effort is put into the architecture, it must eventually be translated into source code."	SD26L4, SD26L9	Architecture AND Code
SD27	"Documenting software architecture rationale is essential to reuse and evaluate architectures, and several modeling and documentation guidelines have been proposed in the literature."	SD27L1, SD27L3	Documentation AND Architecture
SD28	"The importance of Software Architecture (SA) design has been acknowledged as a very important factor for a high-quality software development."	SD28L2, SD28L1	SW Design AND Architecture

SD29	"The remoteness of different decision stakeholders, ranging from local distribution in an office environment to globally distributed teams, as well as the different domain knowledge, expertise and responsibilities of the stakeholders hinder effective and efficient collaboration. Existing tools and methods for collaborative architectural decision making focus mainly on sharing and reusing of knowledge, making trade-offs, and achieving consensus, but do not consider the various stakeholders' decision making constraints due to their roles in the development process."	SD29L1, SD29L5, SD29L10, SD29L8	Stakeholder AND Knowledge AND Trade-off AND Architectural Complexity
SD30	"Change Requests (CRs) are key elements to software maintenance and evolution. Finding the appropriate developer to a CR is crucial for obtaining the lowest, economically feasible, fixing time. Nevertheless, assigning CRs is a labor-intensive and time consuming task."	SD30L1	Requirement Change OR Requirement Volatility
SD31	"One important challenge is architectural knowledge (AK) management, since agile developers prefer sharing knowledge through face-to-face interactions, while in GSD the preferred manner is documents."	SD31L1, SD31l2	Architecture AND Knowledge
SD32	"Software architectures should be evaluated during the early stages of software development in order to verify whether the Non-Functional Requirements (NFRs) of the product can be fulfilled. "	SD32L1	Architecture
SD33	"We discuss how explicating the existing architecture needs to be complemented by social protocols to support the communication and knowledge sharing processes of the walking architecture."	SD33L2, SD33L3, SD33L4	Architecture AND Communication AND Knowledge
A1	"In today's volatile business environments, collaboration between information systems, both within and across company borders, has become essential to success. A key challenge is to manage the ever-growing design complexity. In this article, we argue that software architecture should play a more prominent role in the development of collaborative applications. "	A1L1, A1L5, A1L7, A1L9	Dynamic business Environment AND SW Design AND Architecture Complexity AND Communication Issues
A2	"The architecture of a software-intensive system can be defined as the set of relevant design decisions that affect the qualities of the overall system functionality; therefore,	A2L1, A2L3, A2L9	Architecture AND SW Design AND

	architectural decisions are eventually crucial to the success of a software project. As such, there is no systematic way for software engineers to choose among decision-making techniques for resolving tradeoffs in architecture design."		Trade-off
A3	"Architecture-centric development addresses the increasing complexity and variability of software systems by focusing on architectural models, which are generally easier to understand and manipulate than source code."	A3L2, A3L6	Architectural Complexity AND Code
A4	"Certifying safety-critical software and ensuring its safety requires checking the conformance between safety requirements and design. Inspecting safety conformance by comparing design models against safety requirements requires safety inspectors to browse through large models and is consequently time consuming and error-prone."	A4L4, A4L9	SW Design AND Error Prone
A5	"A complete and detailed (full) Design Rationale Documentation (DRD) could support many software development activities, such as an impact analysis or a major redesign.'	A5L1, A5L2, A5L3,	SW Design AND Documentation AND Design Implementations
A6	"Evaluating and ensuring the consistency between user requirements and modeling artifacts is a long-time issue for model-based software design."	A6L2, A6L3, A6L4	User Requirements AND SW Artefacts AND SW Design
A7	"We use this adaptation scheme for architecture-based self-healing of large software systems. For this purpose, we define the utility for large dynamic architectures of such systems based on patterns that define issues the self-healing must address. Moreover, we use pattern-based adaptation rules to resolve these issues."	A7L2	Self-healing Mechanism
A8	"To date, however, reflection is mainly applied either to the software architecture or its implementation. We know of no approach that fully supports requirements reflection that is, making requirements available as runtime objects. Although there is a body of literature on requirements monitoring, such work typically generates runtime artefacts from requirements and so the requirements themselves are not directly accessible at runtime. "		Architecture AND Requirements AND SW Artefacts

APPENDIX-E

TABLE E: EXECUTION OF IMPLICIT/EXPLICIT REMOVAL

Sr #	Paper ID	Constructs	Implicit & Explicit Removal
1.	W1, W7, A1, A4	Higher defect density, fault proneness, defect proneness, SW failure logs, error handling, pre-release failure, and post-release failure.	SW Defects
2.	I6, W1, W4, W8, W18	Cost overrun, resource estimation, time and resource management, budget constraints, schedule issues, and project size.	Resource Management
3.	I19, I21, W8, W14, W15, SD3, SD11, SD16, SD29, SD31, SD33	Decision knowledge and decision issues	Knowledge
4.	I5, I12, I14, W1, W16, SD33, A1	Poor communication, user-centric communication, coordination mechanism, interaction mechanism, lack of communication, communication gap, message exchange, and information distortion.	Communication Issues
5.	I6, I10, I13, W1, W9, A1	External Dependencies, Change Dependencies, Dependency on modules, Requirement Dependencies, Dependencies b/w SW Components, Structural Dependencies, Data dependencies, Architectural Dependencies and Task Dependencies.	Module Dependencies
6.	I6, I19, W14, W17, W19, SD4, SD17, SD24, A4	Inability to trace design, Tracing patterns, Design rationale Traceability, Traceability Links, Tracing Inconsistencies, and Tracing the architectural Implementation.	Traceability
7.	I14, A1	Dynamic Business Environment	Dynamic Business Environment
8.	I13, I15, I16, I17, W4, W18, SD12, SD29, A6	Stakeholder Synchronization, Stakeholder Goal, Stakeholder Involvement, User Involvement, and Stakeholder Objectives.	Stakeholder
9.	I1, I2, I3,I8, I9, I11, I16, I19, W1, W2, W5, W6, W15, W18, SD1, SD4, SD6, SD7,	Architectural Knowledge, Architectural Decision, Architectural Integration, Architectural Assumptions, Architectural Erosion, Architectural Erosion, Architectural Styles, Architectural Specification, Structural Relationship, and	Architecture

	SD8, SD9, SD10, SD11, SD12, SD14, SD15, SD16, SD18, SD19, SD20, SD22, SD23, SD26, SD27, SD28, SD31, SD32, SD33, A2, A8	Architectural crosscutting concern.	
10.	I1, I3, I4,I5, I8,I11, I13, I15, I18, I21, I22, W10, W12, W17, W5, D5, SD13, SD20, SD28, A1, A2, A4, A5, A6	Design Decisions, Design Patterns, and Design Issues.	SW Design
11.	I10	Wrong Organizational Choice and Basic Incompetency.	Wrong Organizational Choice.
12.	I11, W10, W16, SD6, SD23	Adaption to strategies and policies, Adaption flexibility, Strategy change, and On-demand Adaption.	Adaption strategies and Policies
13.	I1, W13, W19, A3	Maintenance Prediction and SW Maintenance.	Maintenance
14.	I7, A6, A8	SW Artefacts, Design Artefacts, Architecture Req. Artifacts, Artefacts document management, and safety artifacts.	SW Artefacts
15.	W14	Integration of Usage and Utilization of Usage.	Usage
16.	I12, I19, SD29, A2	Trade-off Analysis and Architectural trade-off.	Trade-off
17.	I13, W9, W11, W12, SD2, SD26, A3	Code Smell, Coherent set of code, and code issues.	Code
18.	SD1	Architectural Technical Debt. and Technical Debt Design.	Technical Debt.
19.	SD7, SD9, A6	Human Behaviour and Human cognitive constraints.	Human Behaviour
20.	W9, SD22, SD30, SD8	Lack of verification, Emotional and relational problems, and lack of clarity in the business objective.	Lack of verification
21.	I2, I5, SD13	Team cohesion, Developer focus, and effective collaboration.	Team Cohesion
22.	I12	Lack of Explicit Linkage	Lack of Explicit Linkage
23.	I6, W1, W4, W18,SD27, A5	Architectural Documentation, Poor Documentation, and low-quality documentation.	Documentation

24.	I21, W4, W13, SD2, SD21, SD25, SD29, A1, A3	Increased complexity and Architectural complexity.	Complexity concerns
25.	I3, I8, I9, I13, I16, I17, I18, W1, W2, W3,W9, W11, SD10, SD11, SD14, SD22, SD30, SD8	Ambiguous Requirement, Awareness of requirement volatility, High level of Evaluability, Changing user needs, Tracing the requirement, requirement specification, non-functional requirements, unnecessary changes, anticipated changes, changing to code, knowledge of initial changes, and scope change.	Requirement Volatility
26.	I3, I11	Quality Assurance, Maintaining quality attributes, and Quality Attributes.	Quality Assurance Concerns
27.	SD17	Security	Security Concerns
28.	A7	Self-Healing Mechanism	Self-Healing Mechanism

APPENDIX-F

EXPERT REVIEW EVALUATION FORM

INVITATION LETTER

Respected Sir/Madam,

My name is Sumaira Anwar Baig and I am a student of MS (Software Engineering) at the National University of Modern Languages (NUML), Islamabad. The research topic of my MS degree is *'AN EMPIRICAL STUDY ABOUT POSITIVE IMPLICATIONS OF REQUIREMENTS VOLATILITY ON THE SOFTWARE ARCHITECTURE'* on account of my MS thesis. Whereby, this upcoming study will be able to intimate about the positive implications of requirements volatility (if any) and their impact on the Software Architecture. For this, I have conducted a Systematic Literature Review (SLR) and came up with a list of around 29 factors. To proceed, there is a dire need to analyze- these identified factors from the worthy experts/practitioners of the field. As a result, I will be able to accomplish my second phase of research work. Accordingly, an 'Expert Review' is being carried out. Therefore, you are requested to spare some time to validate my research tasks, which would be highly appreciated and I will be extremely grateful to you.

Yours Truly,

Sumaira Anwar Baig

Student of MS (SE)

Department: Software Engineering (SE)

National University of Modern Languages (NUML),

Islamabad.

SECTION I:

PERSONAL INFORMATION OF EXPERT REVIEW:

Name:

Designation:

Year of Experience:

Expertise:

Domain:

Educational Qualification:

Additional Skills:

SECTION II:

TASKS TO BE PERFORMED BY THE REVIEWER:

Task 1: To verify the naming conventions for a certain factor generated from their sub-factors/data units.

Task 2: To verify the accuracy of each identified factor's classification in terms of category 1, category 2, and category 3.

Category 1: Internal Factors

Internal factors refer to anything within the company and under the control of the firm/company.

Category 2: External Factors

External factors refer to anything outside the firm/company that impacts its success.

Category 3: Both (External & Internal)

Factors refer to the controllable and uncontrollable aspects that could affect the upcoming system.

Acronyms:

KLOC: Thousands of Lines of Code
SW: Software

F #	Sub factors & Data Units	Factors	Description	Category 1 (Internal)	Category 2 (External)	Category 3 (Both)
1.	Higher Defect Density Defect Proneness SW failure logs Error Handling Pre-release failure Post-release failures	SW Defects	This indicates the number of defects confirmed in SW/module. Enable one to decide if a piece of code is ready to release. Represent the KLOC.	I	_	_
2.	Cost overrun Resource Estimation Time and Resource Management Budget Constraints Schedule Issues Project Size	Resource Management	Indicates project resources i.e. time constraints or deadlines, budgeting, scheduling, and tracking.	_	_	Both
3.	Decision Knowledge Decision Issues	Knowledge	Indicates the pre-determined criteria to measure and ensure the optimal outcome for a specific topic.	I	_	_
4.	Poor Communication User Centric Communication Coordination Mechanism Interaction Mechanism Lack of Communication Communication Gap Message Exchange Information Distortion	Communication Issues	Indicates discrepancy B/W said or heard. Moreover, also intimates about the process of communication.	_	_	Both

5.	External Dependencies Change Dependencies Dependency on modules Requirement Dependencies Dependency B/w SW Components Structural Dependencies Data dependencies Architectural Dependencies Task Dependencies	Modules Dependencies	Indicates the relationship b/w project activities and non-project activities. Enable status-completed on schedule.	I	–	–
6.	Inability to trace design Tracing patterns Design rationale Traceability Traceability Links Tracing Inconsistencies Tracing the architectural Implementation	Traceability	Indicates the extent to which documentation or code can be backtracked to its point of origin. Ensure the ability to verify the history, location, and application.	I	–	–
7.	Dynamic Business Environment	Dynamic Business Environment	Indicates rapid changes. Managers and organizations must need to consider this quickly.	–	–	Both

8.	Stakeholder Synchronization	Stakeholder	Indicates variations at the ends of the goals, and objectives of the stakeholders.	–	–	Both
	Stakeholder Goal					
	Stakeholder Involvement					
	User Involvement					
	Stakeholder Objectives					
9.	Architectural Knowledge	Architecture	Indicates the vision of architecture design as well as their decisions and assumptions. Represents the upcoming system.	I	–	–
	Architectural Decision					
	Architectural Integration					
	Architectural Assumptions					
	Architectural Erosion					
	Architectural Styles					
	Architectural Specification					
	Structural Relationship					
	Architectural crosscutting concern					
10.	Design Decision	SW Design	Indicates the complete vision of the plan design and organize decision and challenges towards implementation of the design.	I	–	–
	Design Patterns					
	Design Issues					
11.	Wrong Organizational Choice	Wrong Organizational Choice	Indicates poor planning sets, Inadequate support, lack of resources, lack of priorities, and inadequate change and leadership.	–	–	Both
	Basic Incompetency					

12.	Adaption strategies and policies Adaption Flexibility Strategy Change On-Demand Adaption	Adaption Strategies and Policies	Indicates the changing conditions and intimates the triggers fired by a change user, service, or network.	I	–	–
13.	Maintenance Prediction SW Maintenance	Maintenance	Indicates attributes that bear the capability of SW to maintain its level of performance.	I	–	–
14.	SW artifacts Design Artefacts Architecture Req. Artefacts Artefacts documents management Safety Artefacts	SW Artefacts	Indicates the core development programs and also represents the development process, which may include design, documents, test matrices, prototypes, data models and diagrams etc.	I	–	–
15.	Integration of Usage Utilization of Usage	Usage	Indicates a process of bringing together the different types of software sub-systems. As a result, a unified single system is generated.	I	–	–
16.	Trade-off Analysis Architectural Trade-off	Trade-off	Indicates the evaluation method of SW architecture relative to quality attributes goals.	I	–	–

17.	Code Smell Coherent sets of code Code Issues	Code	Indicates the structures in the source code and intimates deeper problems and violations of fundamental design principles.	I	–	–
18.	Architectural Technical Debt Technical Debt Design	Technical Debt.	Indicates sub-optimal architectural design/decisions and implementation choices.	I	–	–
19.	Human Behaviour Human Cognitive Constraints	Human Behaviour	Indicates the human involvement or group to respond towards the project.	–	E	–
20.	Lack of Verification Emotional and Relational Problems Lack of Clarity in Business Objectives	Lack of Verification	Indicates the confirmation of the pre-requisite, shortage, or absence of desired requirements.	–	–	Both
21.	Team Cohesion Developer Focus Effective Collaboration	Team Cohesion	Indicates the team member's relationship and intimates their positivity to stay in the team.	I	–	–
22.	Lack of Explicit Linkage	Lack of Explicit Linkage	Indicates about lack of standardization of the system and team integration.	I	–	–

23.	Architectural Documentation Poor Documentation Low-Quality Documentation	Documentation	Indicates the drawing plans with scale measurement, specification of the type and quality of material to be used, and other particulars of the upcoming system.	I	–	–
24.	Increased Complexity Architectural Complexity	Complexity Concerns	Indicate the measure of the link b/w architectural complexity that arises within the system due to lack or breakdown of hierarchy or modularity and intimates project cost vibrations.	I	–	–
25.	Ambiguous Requirement Awareness of Requirement Volatility High Level of Evaluability Changing User Needs Tracing the requirements Requirement Specification Non-Functional Requirements	Requirement Volatility	Indicates variations in the project in terms of modifications, upgrading, changes, or new adoptions.	–	–	Both

	Unnecessary changes					
	Anticipated changes					
	Changing to code					
	Knowledge of Initial Changes					
	Scope Change					
26.	Quality Assurance Concerns	Quality Assurance Concerns	Indicates the monitoring terms used towards the adopted methods of the upcoming systems, to ensure the quality.	I	–	–
	Maintaining quality Attributes					
	Quality Attributes					
27.	Security	Security Concerns	Indicates the threat and vulnerability of the system assets.	I	–	–
28.	Self-Healing Issues	Self-Healing Mechanism	Indicates the detection and reaction to the malfunctions of the system.	–	–	Both

APPENDIX-G

SECTION-I

SURVEY FORM

INVITATION LETTER

Respected Sir/Madam,

We would like to invite you to participate in a research survey conducted by Ms. Sumaira Anwar Baig, daughter of Muhammad Anwar Baig scholar of the National University of Modern Languages (NUML), Islamabad as a part of their MS thesis work. This survey aims to elicit the positive implications of Requirements Volatility on the software architecture implemented by the industry, for the successful completion of their project.

2. For this purpose, we would like to invite you to participate in this survey. Your participation in this survey is entirely voluntary. We expect that it should take your 20 minutes, only. However, if you require, we could share the outcomes of this research after the finalization of the results.

3. All information gathered through the questionnaire survey is for research purposes only. Such information will be treated in the STRICTEST CONFIDENCE and any publication (s) from this study will present information in aggregate form such that individual organizations or individual respondents participating in the research cannot be identified.

4. You are free to withdraw your participation from this research at any time you wish and without giving any reason.

5. We should appreciate it if you would agree to participate in this research.

Sumaira Anwar Baig,
Scholar of MS (SE),
Software Engineering Department,
National University Modern Languages (NUML),
Islamabad.
Email: sumaira.numl006@gmail.com
sonia.baig2008@gmail.com

Dr. Huma Hayat Khan,
Supervisor,
Head of Department (Software Engineering),
National University of Modern Languages (NUML),
Rawalpindi.
Email: hnauman@numl.edu.pk

Dr. Muhammad Noman Malik,
Co-Supervisor,
Head of Department (Computer Science),
National University Modern Languages (NUML),
Rawalpindi.
Email: mnauman@numl.edu.pk

SECTION II:

PERSONAL INFORMATION

1.1 Practitioners Detail

Full Name:

Designation:

Qualification:

Software Development experience (in the year) in current/previous organizations:

Organization Address:

Email:

Phone Cell:

Have your company considered the positive implications of requirement volatility on software architecture?

☐ Yes ☐ No

Homan many projects have you performed on the twin peaks of the SDLC i.e. requirement volatility and software architecture?

_____

_____

Write the name of one such project.

_____

1.1 Company Demographics

Company Organization country in which it is located?

_____

What is the scope of your company? (Please tick as appropriate)

☐ National ☐ Multi-National ☐ Don't know

Approximately how many staff is employed by your company/organization? (Please tick as appropriate)

☐ Less than 20

☐ 20-100

☐ Greater than 100

☐ Not sure

What type of certifications your company has achieved?

- ○ CMMI
- ○ ISO
- ○ Other

What type of software is your company concerned with? (You may tick more than one)

- ☐ Web Application
- ☐ Mobile application development
- ☐ Database development
- ☐ Real-time systems
- ☐ Games development
- ☐ Web Design
- ☐ System Software
- ☐ Graphics Designing
- ☐ Desktop application
- ☐ Multimedia Software
- ☐ Other

Is your company care about requirement volatility and its impact on software architecture?

☐ Yes ☐ No

Is your company cares ever participated in the development of any software project to consider the fragile nature of requirements volatility, throughout the development? If yes, write the name of at least one such project.

_____

_____

SECTION III:

2.1 Empirical Investigation about positive implications of Requirements Volatility on Software Architecture (RVSAs)

Initially, the literature highlighted the implications of the requirements volatility factors on SW architecture adopted by the practitioners. Accordingly, define the category of each factor based on three different categories i.e. A, B, and C, respectively. Where, the 'A' represents the 'Internal' factor, 'B' represents the 'External Factor' and 'C' represents the 'both'. In this section, we intended to validate whether these implementations raised 'positive'

impacts on software architecture, based on your experience, please check the appropriate box, given in the front of each practice placed in the table, and suggest any other practice, if any.

RVSA1-Software Defects

This is a category C type factor i.e. both which indicate the numbers of defects confirmed in the software module e.g. higher defect density, defect proneness, SW failure logs, Error handling, pre-release and post-release failures. Further, enable one to decide if a piece of code is ready to release. Accordingly, represent the KLOC.

- O Strongly Agree
- O Agree
- O Neutral
- O Disagree
- O Strongly Disagree

RVSA2-Resource Management

This is a category C type factor i.e. both which deal with the project resources i.e. cost overrun, resource management, time constraints or deadlines, budgeting, scheduling, and tracking related matters.

- O Strongly Agree
- O Agree
- O Neutral
- O Disagree
- O Strongly Disagree

RVSA3-Knowledge

This is a category A type factor i.e. internal which indicates the knowledge i.e. Decision Knowledge, Decision issues, and pre-determined criteria to measure and ensure the optimal outcome for a specific task.

- O Strongly Agree
- O Agree
- O Neutral
- O Disagree
- O Strongly Disagree

RVSA4-Communication Issues

This is a category C type factor i.e. both indicate about discrepancy B/W said or heard. Moreover, also intimates about the process of communication and their issues i.e. poor communication, user-centric communication, coordination mechanism, communication gap, information distortion, and message exchange-related matters.

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

RVSA5-Dependencies

This is a category C type factor i.e. both which indicate the dependencies b/w project and non-project activities i.e. change dependencies, external dependencies, data dependencies, and task dependencies. Moreover, more curiously focused on architectural and module dependencies.

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

RVSA6-Traceability

This is a category A type factor i.e. Internal which deals with tracking related matters to its point of origin i.e. tracing patterns, traceability links, and tracing towards design and architectural implementation. Besides this, verify the inabilities to trace a design or trace inconsistencies.

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

RVSA7-Dynamic Business Environment

This is a category C type factor i.e. both which deal with the matters of rapid changes towards business environment i.e. taste and preferences and changes in technology issues.

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

RVSA8-Stakeholder Synchronization

This is a category C type factor i.e. both which indicate the stakeholder goals and objectives including user involvement-related matters.

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

RVSA9-Architecture

This is a category A type factor i.e. internal which represents the architectural design and deals with the architecture in terms of their decision, assumptions, styles, specifications, and integrations-related matters.

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

RVSA10-Design Implementations

This is a category C type factor i.e. both which deal with the part of the SW design i.e. Design decisions, design patterns, and design issues related matters.

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

RVSA11-Organizational Leadership

This is a category C type factor i.e. both which deal with the organization's choices, planning, resources, priorities, and leadership-related matters.

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

RVSA12-Adaption to change

This is a category C type factor i.e. both which deal with the adaption strategies and policies towards implementation of changes and intimates about the triggers fired by the change user, services, and conditions.

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

RVSA13-SQW Maintenance

This is a category A type factor i.e. internal which deals with the SW maintenance-related matters and has capabilities towards maintenance prediction to its level of performance.

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

RVSA14-Artefacts

This is a category A type factor i.e. internal which deals with the core development programs and also represents their process including design documentation, test matrices, prototypes, data models and diagrams, etc.

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

RVSA15-Integration of Usage

This is a category A type factor i.e. internal which deals with the process of bringing together the different types of software sub-systems, to consider the utilization of usage regarding the continuous SE process.

- ○ Strongly Agree
- ○ Agree
- ○ Neutral
- ○ Disagree
- ○ Strongly Disagree

RVSA16-Trade-off

This is a category A type factor i.e. internal which deals with the evaluation method of SW architecture relative to its quality attribute goals in terms of architectural trade-off.

- ○ Strongly Agree
- ○ Agree
- ○ Neutral
- ○ Disagree
- ○ Strongly Disagree

RVSA17-Code

This is a category A type factor i.e. internal which indicates the structures in the source code and intimates deeper problems and violations of fundamental design principal. Moreover, represents the code smell, a coherent set of rules, and code-related matters.

- ○ Strongly Agree
- ○ Agree
- ○ Neutral
- ○ Disagree
- ○ Strongly Disagree

RVSA18-Technical Debt.

This is a category A type factor i.e. internal which indicates the optimal architectural design/decisions and implementation choices. Where, the code-level technical debts are detected and analyzed by the static analyzers. Moreover, represents the choices about the architectural structures, frameworks, technologies, languages, etc.

- ○ Strongly Agree
- ○ Agree
- ○ Neutral
- ○ Disagree
- ○ Strongly Disagree

RVSA19-Human Behavior

This is a category C type factor i.e. both indicate the human involvement and group to respond concerning the human cognitive constraints and behavior.

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

RVSA20-Team

This is a category C type factor i.e. both which indicate the developer focus, team members, and their effective collaboration.

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

RVSA21-Integration of Linkage

This is a category C type factor i.e. both which indicate the standardization of the system and team integration.

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

RVSA22-Documentation

This is a category A type factor i.e. internal which deals with the drawing plans with scale measurements, specification of types, and quality of material to be used. Besides this, deal with the low quality or poor quality documentation sub-factors.

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

RVSA23-Architectural Complexity

This is a category A type factor i.e. internal which deals with the architectural complexity and their sub-factor i.e. increased complexity that arises within the system due to lack of breakdown of hierarchy or modularity and intimates about the project cost variations.

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

RVSA24-Requirement Volatility

This is a category C type factor i.e. both which indicate the modifications, upgrading, changes, or new adoptions. Besides this, more specifically deal with all possible sub-factors i.e. ambiguous requirement, awareness of requirement volatility, High level of evaluability, changing user needs, non-functional requirements, unnecessary changes, and scope change-related matters.

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

RVSA25-Quality Assurance

This is a category C type factor i.e. both, which deal with the quality assurance concerns towards maintaining the quality attributes and more specifically ensuring the quality of the upcoming product.

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

RVSA26-Security

This is the category C type factor i.e., which deals with the security concerns of the systems in terms of threat and vulnerability of the systems.

- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly Disagree

RVSA27-Self Healing Mechanism

This is the category C type factor i.e., which deals with the healing mechanism regarding the detection and reaction to the malfunctions of the system.

- ○ Strongly Agree
- ○ Agree
- ○ Neutral
- ○ Disagree
- ○ Strongly Disagree

APPENDIX-H

FINAL LIST OF IDENTIFIED LIST OF FACTORS ALONG WITH THEIR CATEGORIES.

Sr No.	Paper ID	Sub-factors/Data Units	Factor (s)	Category
1.	W1, W7, A1, A4	Higher Defect Density	SW Defects	Both
		Defect Proneness		
		SW failure logs		
		Error Handling		
		Pre-release failure		
		Post-release failures		
2.	I6, W1, W4, W8, W18	Cost overrun	Resource Management	Both
		Resource Estimation		
		Time and Resource Management		
		Budget Constraints		
		Schedule Issues		
		Project Size		
3.	I19, I21, W8, W14, W15, SD3, SD11, SD16, SD29, SD31, SD33	Decision Knowledge	Knowledge	Internal
		Decision Issues		
4.	I5, I12, I14, W1, W16, SD33, A1	Poor Communication	Communication Issues	Both
		User-Centric Communication		
		Coordination Mechanism		
		Communication Gap		
		Message Exchange		
		Information Distortion		
5.	I6, I10, I13, W1, W9, A1	External Dependencies	Dependencies	Both
		Change Dependencies		
		Dependency on modules		
		Requirement Dependencies		
		Data dependencies		
		Architectural Dependencies		
		Task Dependencies		
6.	I6, I19, W14, W17, W19, SD4, SD17, SD24, A4	Inability to trace a design	Traceability	Internal
		Tracing patterns		
		Design rationale Traceability		
		Traceability Links		
		Tracing Inconsistencies		
		Tracing the architectural Implementation		
7.	I14, A1	Dynamic Business Environment	Dynamic Business Environment	Both

8.	I13, I15, I16, I17, W4, W18, SD12, SD29, A6	Stakeholder Synchronization Stakeholder Goal & Objectives Stakeholder Involvement	Stakeholder	Both
9.	I1, I2, I3,I8, I9, I11, I16, I19, W1, W2, W5, W6, W15, W18, SD1, SD4, SD6, SD7, SD8, SD9, SD10, SD11, SD12, SD14, SD15, SD16, SD18, SD19, SD20, SD22, SD23, SD26, SD27, SD28, SD31, SD32, SD33, A2, A8	Architectural Knowledge Architectural Decision Architectural Integration Architectural Assumptions Architectural Erosion Architectural Styles Architectural Specification Architectural crosscutting concern	Architecture	Internal
10.	I1, I3, I4, I5, I8, I11, I13, I15, I18, I21, I22, W5, W10, W12, W17, SD5, SD13, SD20, SD28, A1, A2, A4, A5, A6	Design Decision Design Patterns Design Issues	SW Design & Design Implementation	Internal
11.	I10	Wrong Organizational Choice Basic Competency	Organizational Leadership	Both
12.	I11, W10, W16, SD6, SD23	Adaption Strategies and Policies Adaption Flexibility Strategy Change On-Demand Adaption	Adaption to Change	Both
13.	I1, W13, W19, A3	Maintenance Prediction SW Maintenance	SQW Maintenance	Internal
14.	I7, A6, A8	SW Artefacts Design Artefacts Architecture Req. Artifacts Artifacts Documents Management	Artifacts	Internal
15.	W14	Integration of Usage Utilization of Usage	Integration of Usage	Internal
16.	I12, I19, SD29, A2	Trade-off Analysis Architectural Trade-off	Trade-off	Internal
17.	I13, W9, W11, W12, SD2, SD26, A3	Code Smell Coherent sets of code Code Issues	Code	Internal
18.	SD1	Architectural Technical Debt Technical Debt Design	Technical Debt.	Internal

19.	SD7, SD9, A6	Human Behavior	Human Behaviour	Both
		Human Cognitive Constraints		
20.	I2, I5, SD13	Team Cohesion	Team	Both
		Developer Focus		
21.	I12	Lack of Explicit Linkage	Integration of Linkage	Both
22.	I6, W1, W4, W18, SD27, A5	Architectural Documentation	Documentation	Internal
		Poor Documentation		
		Low-Quality Documentation		
23.	I21, W4, W13, SD2, SD21, SD25, SD29, A1, A3	Increased Complexity	Architectural Complexity	Internal
		Architectural Complexity		
24.	I3, I8, I9, I13, I16, I17, I18, W1, W2, W3, W9, W11, SD08, SD10, SD11, SD14, SD22, SD30	Ambiguous Requirement	Requirement Volatility	Both
		Awareness of Requirement Volatility		
		High Level of Evaluability		
		Changing User Needs		
		Tracing the requirements		
		Requirement Specification		
		Non-Functional Requirements		
		Unnecessary changes		
		Anticipated changes		
		Changing to code		
		Knowledge of Initial Changes		
		Scope Change		
		Lack of Verification		
		Emotional and Relational Problems		
		Lack of Clarity in Business Objectives		
25.	I3, I11	Quality Assurance Concerns	Quality Assurance	Both
		Maintaining Quality Attributes		
		Quality Attributes		
26.	SD17	Security	Security	Internal
27.	A7	Self-Healing Mechanism	Self-Healing Mechanism	Both

APPENDIX-I
LIST OF INCLUDED PRIMARY STUDIES ALONG WITH THE PAPER ID.

Sr No.	Paper ID	Study Name
1.	**I1**	A Prescriptive Approach to Quality-Focused System Architecture
2.	**I2**	Evaluating System Architecture Quality and Architecting Team Performance Using Information Quality Theory
3.	**I3**	SAM-SoS: A Stochastic Software Architecture Modelling and Verification Approach for Complex System-of-Systems
4.	**I4**	Automatically Detecting and Tracking Inconsistencies in Software Design Models
5.	**I5**	Decision-Making Assistance in Engineering Change Management Process
6.	**I6**	Comparative Analysis of Requirement Change Management Challenges Between in-House and Global Software Development: Findings of Literature and Industry Survey
7.	**I7**	An Industrial Survey of Safety Evidence Change Impact Analysis Practice
8.	**I8**	Structuring Software Requirements for Architecture Design
9.	**I9**	Software Architecture Matching by Meta-model Extension and Refinement
10.	**I10**	Product Line Requirements Reuse Based on Variability Management
11.	**I11**	Quality-Driven Self-Adaptation: Bridging the Gap between Requirements and Runtime Architecture by Design Decision
12.	**I12**	Priority-Awareness of Non-Functional Requirements under Uncertainty
13.	**I13**	An initial evaluation of requirements dependency types in change propagation analysis
14.	**I14**	A Method of Specifying and Classifying Requirements Change
15.	**I15**	Towards requirements aware systems: Run-time resolution of design-time assumptions
16.	**I16**	RADAR: A Lightweight Tool for Requirements and Architecture Decision Analysis
17.	**I17**	Towards search-based modeling and analysis of requirements and architecture decisions
18.	**I18**	Run-time Resolution of Uncertainty
19.	**I19**	Assessing Architectural Patterns Trade-offs using Moment-based Pattern Taxonomies
20.	**I20**	Improving Software Performance and Reliability with an Architecture-Based Self-Adaptive Framework
21.	**I21**	Managing Design Time Uncertainty
22.	**I22**	Reducing Uncertainty in Architectural Decisions with AADL
23.	**I23**	Inconsistency Management between Architectural Decisions and Designs Using Constraints and Model Fixes
24.	**W1**	Impact of requirements volatility on software architecture: How do software teams keep up with ever-changing requirements?
25.	**W2**	Towards evolvable software architectures based on systems theoretic stability
26.	**W3**	Managing changes in requirements: an empirical investigation
27.	**W4**	An optimization-based tool to support the cost-effective production of software architecture documentation

28.	**W5**	Classification and comparison of architecture evolution reuse knowledge—a systematic review
29.	**W6**	Measuring stability of object-oriented software architectures
30.	**W7**	Exploring the missing link: an empirical study of software fixes
31.	**W8**	Exploring factors affecting decision outcome and lead time in large-scale requirements engineering
32.	**W9**	Do code data sharing dependencies support an early prediction of software's actual change impact set?
33.	**W10**	A procedural and flexible approach for specification, modeling, definition, and analysis for self-adaptive systems
34.	**W11**	Supporting requirements update during software evolution
35.	**W12**	Software smell detection techniques: A systematic literature review
36.	**W13**	Increasing software development efficiency and maintainability for complex industrial systems – A case study
37.	**W14**	Continuous software engineering and its support by usage and decision knowledge: An interview study with practitioners
38.	**W15**	Patterns in System Architecture Decisions
39.	**W16**	Analyzing runtime adaptability of collaboration patterns
40.	**W17**	Pattern detection and design rationale traceability: an integrated approach to software design quality
41.	**W18**	An optimization-based tool to support the cost-effective production of software architecture documentation
42.	**W19**	The impact of traceability on software maintenance and evolution: A mapping study
43.	**SD1**	Software sustainability: Research and practice from a software architecture viewpoint
44.	**SD2**	Stability assessment of aspect-oriented software architectures: A quantitative study
45.	**SD3**	10 years of software architecture knowledge management: Practice and future
46.	**SD4**	Generation and validation of traces between requirements and architecture based on formal trace semantics
47.	**SD5**	Early validation of system requirements and design through correctness-by-construction
48.	**SD6**	Optimization of adaptation plans for a service-oriented architecture with cost, reliability, availability, and performance trade-off
49.	**SD7**	Empirical research for software architecture decision making: An analysis
50.	**SD8**	Software architecture evolution through evolve-ability analysis
51.	**SD9**	Managing the evolution of software architecture at minimal cost underperformance and reliability constraints
52.	**SD10**	Characterizing software architecture changes: A systematic review
53.	**SD11**	An exploratory study of architectural effects on requirements decisions
54.	**SD12**	A documentation framework for architecture decisions
55.	**SD13**	Accessing decision-making in software design
56.	**SD14**	Impact propagation and risk assessment of requirement changes for software development projects based on design structure matrix
57.	**SD15**	A systematic review of software architecture evolution research
58.	**SD16**	Advanced quality prediction model for software architectural knowledge sharing

59.	**SD17**	Exploiting traceability uncertainty between software architectural models and extra-functional results
60.	**SD18**	A survey on software architectural assumptions
61.	**SD19**	Evaluation of a process for architectural assumption management in software development
62.	**SD20**	Towards supporting the software architecture life cycle
63.	**SD21**	Scaling up software architecture analysis
64.	**SD22**	Supporting runtime software architecture: A bidirectional-transformation-based approach
65.	**SD23**	Controlling software architecture erosion: A survey
66.	**SD24**	Requirements traceability technologies and technology transfer decision support: A systematic review
67.	**SD25**	Performance measurement of models specified through component-based software architectural styles
68.	**SD26**	Reconciling software architecture and source code in support of software evolution
69.	**SD27**	Bridging the gap between software architecture rationale formalisms and actual architecture documents: An ontology-driven approach
70.	**SD28**	The strengths and weaknesses of software architecture design in the RUP, MSF, MBASE, and RUP-SOA methodologies: A conceptual review
71.	**SD29**	Automatic enforcement of constraints in real-time collaborative architectural decision making
72.	**SD30**	Towards semi-automated assignment of software change requests
73.	**SD31**	Towards a reduction in architectural knowledge vaporization during agile global software development
74.	**SD32**	Validating a model-driven software architecture evaluation and improvement method: A family of experiments
75.	**SD33**	Software architecture awareness in long-term software product evolution
76.	**A1**	Architecture-centric support for adaptive service collaborations
77.	**A2**	Decision-making techniques for software architecture design: A comparative survey
78.	**A3**	Maintaining Architecture-Implementation Conformance to Support Architecture Centrality: From Single System to Product Line Development
79.	**A4**	Traceability and SysML design slices to support safety inspections: A controlled experiment
80.	**A5**	The value of design rationale information
81.	**A6**	Ensuring the Consistency between User Requirements and Task Models: A Behaviour-Based Automated Approach
82.	**A7**	Improving Scalability and Reward of Utility-Driven Self-Healing for Large Dynamic Architectures
83.	**A8**	Requirements reflection: requirements as runtime entities