Automated Conflicts Detection of Software Requirements using Natural Language Processing

Author

Aleena Arshad

FALL 2015-MS-15(CSE) 00000119838

MS-15 (CSE)

Supervisor

Dr. Wasi Haider Butt

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

JULY, 2019

Automated Conflicts Detection of Software Requirements using Natural Language Processing

Author

Aleena Arshad

FALL 2015-MS-15(CSE) 00000119838

A thesis submitted in partial fulfillment of the requirements for the degree of

MS Software Engineering

Thesis Supervisor:

Dr. Wasi Haider Butt

Thesis Supervisor's Signature:_____

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY, ISLAMABAD

JULY, 2019

DECLARATION

I certify that this research work titled "Automated Conflicts Detection in Software Requirements using Natural Language Processing*"* is my own work under the supervision of Dr. Wasi Haider Butt. The work has not been presented elsewhere for assessment. The material that has been used from other sources it has been properly acknowledged / referred.

Signature of Student

Aleena Arshad

FALL 2015-MS-15(CSE) 00000119838

LANGUAGE CORRECTNESS CERTIFICATE

This thesis has been read by an English expert and is free of typing, syntax, semantic, grammatical and spelling mistakes. Thesis is also according to the format given by the university.

Signature of Student

Aleena Arshad

FALL 2015-MS-15(CSE) 00000119838

Signature of Supervisor

COPYRIGHT STATEMENT

ACKNOWLEDGEMENTS

I am thankful to my Creator Allah Subhana-Watala to have guided me throughout this work at every step and for every new thought which You setup in my mind to improve it. Indeed I could have done nothing without Your priceless help and guidance. Whosoever helped me throughout the course of my thesis, whether my parents or any other individual was Your will, so indeed none be worthy of praise but You.

I am profusely thankful to my beloved parents who raised me when I was not capable of walking and continued to support me throughout in every department of my life.

I would also like to express special thanks to my supervisor Dr. Wasi Haider Butt for his help throughout my thesis and also for Software Requirements Engineering (SRE) course which he has taught me. I can safely say that I haven't learned any other engineering subject in such depth than the ones which he has taught.

I would also like to thank my Guidance Committee Members Dr. Arslan Shaukat and Dr. Usman Akram for being on my thesis guidance and evaluation committee. Their recommendations are very valued for improvement of the work. I am also thankful to Abrar Ahmed, Imran Ahsan, and Anum Amjad for their support and cooperation.

Finally, I would like to express my gratitude to all the individuals who have rendered valuable assistance to my study.

Dedicated to my exceptional parents and adored siblings whose tremendous support and cooperation led me to this wonderful accomplishment

ABSTRACT

Contradictory and inconsistent sentences in a set of requirements are known as conflicting requirements. In the Requirements Engineering phase of Software Development Life Cycle (SDLC) software requirements are gathered, analyzed, negotiated back and forth manually to come to a final requirements specification document that is free from a known problem – conflicting requirements. By automating conflict detection during requirements analysis phase, time, effort, and resources can be saved in going back and forth and checking for conflicts manually. Natural Language Processing (NLP) is a way to pre-process software requirements contextually before a manual or automated model or algorithm can be applied on them. SLR (Systematic Literature Review) has been performed to distinguish 23 papers published during 2009 to 2018. The idea is to determine conflict detection models in software requirements using NLP. Furthermore, to identify tools, models, and case studies that have been vital in conflict detection since 2009. We have identified 10 tools, 23 models, and 14 case studies that have proposed conflict detection while using NLP techniques. We gathered that there is no known automated conflict detection model in software requirements using NLP techniques and contextual rules. Finally, we applied our approach to our data set and achieved complete conflict detection by comparing manual and automated testing.

Keywords: conflict detection, inconsistency detection, automated requirements analysis, Natural Language Processing (NLP), software requirements

TABLE OF CONTENTS

DECLARATION ..... **i**
LANGUAGE CORRECTNESS CERTIFICATE ..... **ii**
COPYRIGHT STATEMENT ..... **iii**
ACKNOWLEDGEMENTS ..... **iv**
ABSTRACT ..... **vi**
LIST OF FIGURES ..... **ix**
LIST OF TABLES ..... **xi**
1. CHAPTER 1: INTRODUCTION **13**
1.1 Background Study 13
1.1.1 Conflict Detection in Software Requirements 13
1.1.2 Conflict Detection Using NLP Techniques 13
1.2 Problem Statement 14
1.3 Proposed Methodology 15
1.3.1 Research Contribution 15
1.4 Thesis Organization 16
2. Chapter 2: Literature Review **19**
2.1. Review Protocol 19
2.1.1. Categories Definition 19
2.1.2. Selection and Rejection Criteria 20
2.1.3. Search Process 21
2.1.4. Quality Assessment 23
2.1.5. Data Extraction and Synthesis 25
2.2. Research and Analysis 26
2.2.1. Conflict Detection Models and Algorithms 26
2.2.2. Conflict Detection Tools 27
2.2.3. Conflict Detection Specific Case Studies/Data Sets 29
2.3. Research Gaps 30
3. Chapter 3: Proposed Methodology **33**
3.1. Targeted Problem 33
3.1.1. Requirements Engineering Phases 35
3.1.2. Detecting Conflicts during Requirements Analysis 36
3.2. Proposed Solution 37
4. Chapter 4: Implementation **40**
4.1. Requirements Specification 40
4.2. NLP Pre-Processing Techniques 41

4.3. Transformation Contextual Rules 42
4.3.1. Extracted Text Classification 42
4.3.2. Contextual Rules 43
4.4. Conflict Detection Architecture 45
5. Chapter 5: Verification and Validation 48
5.1. CMS Requirements Data Set 48
5.1.1. Requirement Specification 48
5.1.2. Conflict Detection Algorithm 49
5.1.3. Conflict Detection Verification 54
5.2. PMRB Software Development QA Guidance Document 61
5.2.1. Requirement Specification 61
5.2.2. NLP Pre-Processing 62
5.2.3. Conflict Detection Verification 63
Chapter 6: Discussion and Limitation 73
6.1. Discussion 73
6.2. Limitations 74
Chapter 7: Conclusion and Future Work 76
REFERENCES 77

LIST OF FIGURES

Figure 1.1: NLP Techniques **14**
Figure 1.2: Conflict Detection Process **14**
Figure 1.3: Research Flow **15**
Figure 1.4: Thesis Outline **17**
Figure 2.1: Search Process **23**
Figure 2.2: Selected Researches per Year **24**
Figure 2.3: Selected Publishers per Year **24**
Figure 3.1: The Software Development Life Cycle (SDLC) **33**
Figure 3.2: All Stages of SDLC **34**
Figure 3.3: Requirements Engineering Process **35**
Figure 3.4: Iterative Requirement Engineering Process **36**
Figure 3.5: Requirements Analysis Process **37**
Figure 3.6: Commonly used NLP Techniques **38**
Figure 4.1: Requirements Specification **40**
Figure 4.2: NLP Pre-Processing Techniques used **41**
Figure 4.3: Architecture of Automated Conflict Detection **45**
Figure 5.1: Requirements Specification **49**
Figure 5.2: Libraries used in Conflict Detection Algorithm **49**
Figure 5.4: NLP Pre-Processing Techniques Part 2 ..... **50**
Figure 5.3: NLP Pre-Processing Techniques Part 1 ..... **50**
Figure 5.5: NLP Pre-Processing Techniques Part 3 ..... **50**
Figure 5.6: Classification of Labels Part 1 ..... **51**
Figure 5.7: Classification of Labels Part 2 ..... **51**
Figure 5.8: Processed Requirements List **52**
Figure 5.9: Contextual Rules Part 1 ..... **52**
Figure 5.11: Contextual Rules Part 3 ..... **53**
Figure 5.10: Contextual Rules Part 2 ..... **53**
Figure 5.12: Automated Conflict Detection for R0 ..... **54**
Figure 5.13: Automated Conflict Detection for R1 ..... **55**
Figure 5.14: Automated Conflict Detection for R2 ..... **55**
Figure 5.15: Automated Conflict Detection for R3 ..... **56**
Figure 5.16: Automated Conflict Detection for R4 ..... **56**
Figure 5.17: Automated Conflict Detection for R5 ..... **57**
Figure 5.18: Automated Conflict Detection for R6 ..... **57**
Figure 5.19: Automated Conflict Detection for R7 ..... **58**

Figure 5.20: Automated Conflict Detection for R8 58
Figure 5.21: Automated Conflict Detection for R9 59
Figure 5.22: Requirements Specification..... 62
Figure 5.23: Processed Requirements List 63
Figure 5.24: Automated Conflict Detection for R0 64
Figure 5.25: Automated Conflict Detection for R1 64
Figure 5.26: Automated Conflict Detection for R2 65
Figure 5.27: Automated Conflict Detection for R3 65
Figure 5.28: Automated Conflict Detection for R4 66
Figure 5.29: Automated Conflict Detection for R5, R6, R7, and R8..... 66
Figure 5.30: Automated Conflict Detection for R9 67
Figure 5.31: Automated Conflict Detection for R10 67
Figure 5.32: Automated Conflict Detection for R11 68
Figure 5.33: Accuracy, Recall, and Precision Score Interpretation for Results 71

LIST OF TABLES

Table 2.1: Details of research works per database 20
Table 2.2: Details of search terms and search results 22
Table 2.3: Data Extraction and Synthesis 25
Table 2.4: Conflict Detection Models and Algorithms 27
Table 2.5: Conflict Detection Tools 28
Table 2.6: Conflict Detection specific Case Studies/Data Sets 29
Table 2.7: Comparison of Selected Researches 30
Table 4.1: Contextual Rules 43
Table 5.1: Automated Testing Results for CS1 59
Table 5.2: Manual Testing Results for CS1 60
Table 5.3: Automated Testing Results for CS2 68
Table 5.4: Manual Testing Results for CS2 69
Table 5.5: Comparison of Automated Vs Manual Testing for CS2 70

Chapter 1

Introduction

1. CHAPTER 1: INTRODUCTION

This chapter offers a detailed introduction of our research. Section 1.1 discusses the background study, Section 1.2 presents the problem statement, Section 1.3 gives proposed methodology in, and Section 1.4 contains thesis organization.

1.1 Background Study

The purpose of providing the background study is to introduce the main concepts used in this research. The concepts involved are; 1) Conflict detection in software requirements, and 2) Conflict detection using NLP techniques. The details of the following are given in subsequent sections.

1.1.1 Conflict Detection in Software Requirements

Gathering requirements can be a long and arduous task in the Software Development Life Cycle (SDLC), and this task can result in errors that can be a cause of worry later on. Understanding the root cause of false requirements is a necessity in order to avoid adverse effects of eliciting and incorporating wrong requirements that can ultimately hinder the progress of an efficient software project. The analysis and detection of conflicts in the requirements phase are one of the most critical tasks in requirements engineering [1].

1.1.2 Conflict Detection Using NLP Techniques

Requirement elicitation and analysis is one of the main step in the development of a product. Usually this involves a list of reasons that can hinder proper and efficient requirements elicitation, analysis and specification. One of the issues faced is conflict emergence between requirements' documents. If the requirements are not processed and issues not found and corrected, the base of a product starts as hollow. For software requirements, there have been methods and techniques introduced and adapted over the years to weed out the conflicts, inconsistencies, and ambiguities amongst other issues. One of the most commonly used technique is Natural Language

Processing (NLP) on a set of requirements. NLP is done on any set of requirements using a number of pre-processing techniques as shown in **Figure *1.1* [2].**

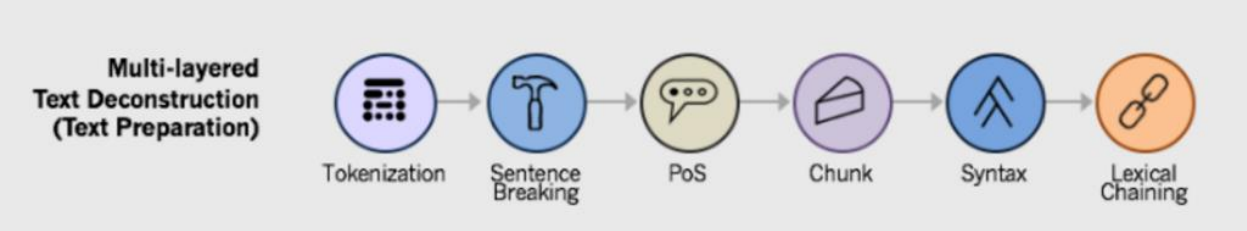

Figure 1.1: NLP Techniques

In order to find conflicts in software requirements, a set of rules are applied in an order of hierarchy of loops, rules and parameters to find out the requirements that can have conflict amongst them. This completes the conflict detection architecture of our targeted problem's solution.

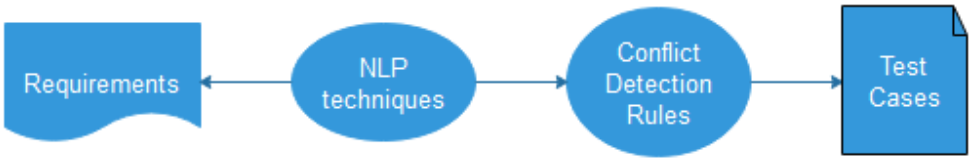

Figure 1.2: Conflict Detection Process

1.2 Problem Statement

Due to the growing complexity of products, it has become vital to find out issues in software requirements at an early stage. Detecting conflicts has become difficult in parallel. Moreover, hidden conflicts can cause issues in all the leading steps in a products' development lifecycle. To avoid these problems, conflicts must be found at the requirements engineering phase. Without a proper method to evaluate requirements or a set of rules to identify general categories of conflicts in requirements, the problem statement becomes moot. So, in order to detect in software requirements, we have proposed a set of rules using the Natural Language Processing (NLP) to detect conflicts in a set of requirements by a method of comparison. This has led to the results being a collective findings of "Yes" and "No" in terms of conflicts between the compared requirements.

1.3 Proposed Methodology

The entire research is done in a systematic way. Flow of the research is shown in ***Figure 1.3***. First of all, we identify the problem, then we propose a solution to the identified problem. Then, we carry out a comprehensive systematic literature review which becomes the foundation of the proposed solution. Researches related to the proposed solution are analyzed and compared.

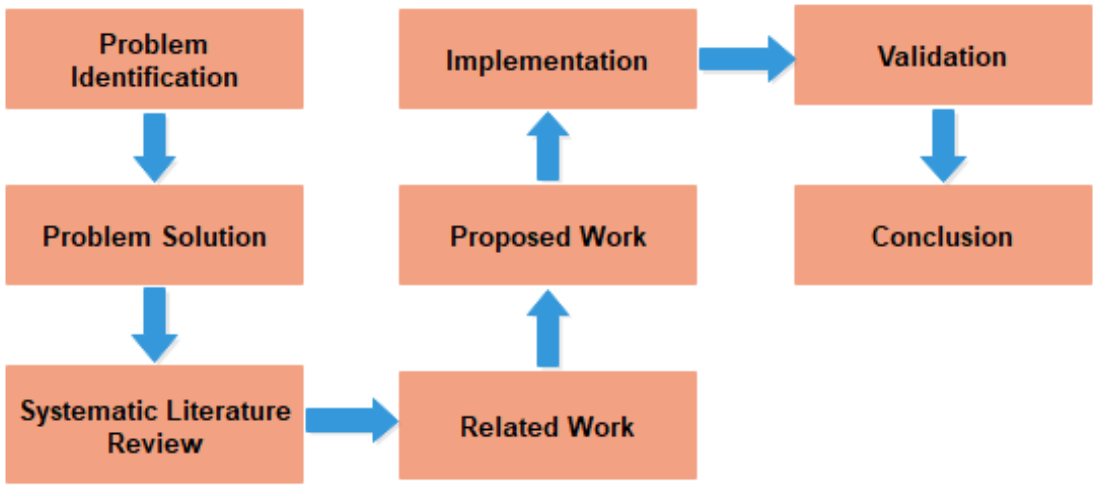

Figure 1.3: Research Flow

1.3.1 Research Contribution

Contributions from this research work are two-fold i.e. finding all the existing tools, methods and algorithms used in the detection of conflicts in software requirements and devising a set of rules using NLP to detect conflicts in a set of software requirements. Detailed set of contributions of the proposed approach are as follows:

- We have researched and collected existing tools, techniques, models, and algorithms and specific conflict detection case studies or data sets.
- We have further researched the use of NLP techniques most common to finding specific issues in software requirements.
- We have worked on an algorithm that uses these common NLP techniques and applied them during multiples phases of our algorithm to detect conflicts in our data set of software requirements.
- We have proposed an algorithm based approach in order to obtain our results.

The main artifact produced during this approach is a complete set of rules to compare processed requirements to detect conflicts. Other artifacts in the development phase include using NLP techniques after the requirements are read and stored in a matrix and again during the comparison phase of our development. The entire development is done using Python on the PyCharm Educational Setup with the Natural Language Toolkit (NLTK). We have provided validation of our proposed work using manual testing and automated testing of the requirements and comparing the results of both. We have used a set of local requirements for training and a Quality Assurance document to test out our proposed algorithm.

1.4 Thesis Organization

Organization of the thesis is represented in ***Figure 1.4***. **Chapter 1: Introduction** offers a brief introduction containing the background study, problem statement, research contribution and thesis organization. **Chapter 2: Literature Review** provides the detailed literature review highlighting the work done in the domain of software requirements analysis and NLP. **Chapter 3: Proposed Methodology** covers the details of proposed methodology used for identification of problem. **Chapter 4: Implementation** presents the detailed implementation regarding the proposed algorithm. **Chapter 5: Validation** provides the validation performed for our proposed methodology using two important case studies. From the two case studies selected, one is for training of data and the second is for testing of our proposed algorithm. **Chapter 6: Discussion And Limitation** contains a brief discussion on the work done and also contains the limitations to our research. **Chapter 7: Conclusion And Future Work** concludes the research and recommends a future work for the research.

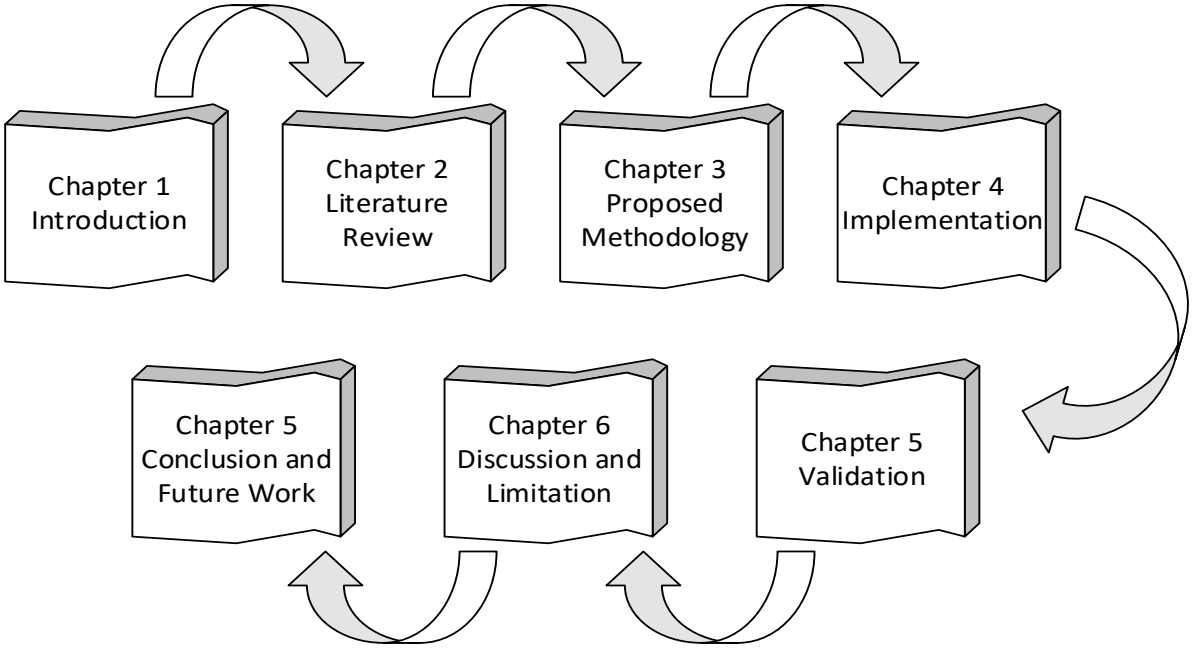

Figure 1.4: Thesis Outline

Chapter 2

Literature Review

2. Chapter 2: Literature Review

This chapter presents the literature review carried out for the research. Section 2.1 discusses the review protocol, Section 2.2 presents the results obtained from the review protocol and Section 2.3 highlights the research gaps which form the foundation of our research.

2.1. Review Protocol

We carried out the review protocol development for our study, based on already defined Systematic Literature Review by Kitchenham [3]. This review protocol demonstrates the category definition, criteria of selection and rejection, assessment of quality, extraction of data and the mechanism used for data synthesis. The details of these elements are given in following subsections.

2.1.1. Categories Definition

We define three categories to simplify the data extraction and synthesis process. The description of each category is given below.

Software Requirements: This category sets the start of research collection for our study. This included software requirements and the issues found in requirements during elicitation, analysis or specifications phase of requirements phase in the development lifecycle of a product.

Conflict Detection: This category deals with all the studies that focused on conflict detection in software requirements. This includes all the software requirements based studies that focused on finding conflicts. Conflict detection alone in software specific requirements were not quite enough, so any set of requirement for any product that focused on conflict detection was included to start with a database of studies.

Natural Language Processing Techniques: Further categories include the use of Natural Language Processing (NLP) techniques in detection of conflicts in software requirements irrespective of a specific industry. This includes all studies that make the use of one or more techniques of NLP on a set of requirements to focus on conflict detection.

2.1.2. Selection and Rejection Criteria

The standard and benchmark for the inclusion and exclusion of this study are declared by using seven parameters. These factors defined to certify the validness of the responses of our questions. The studies that do not comply with and do not fulfill these six parameters are not considered. Selection parameters for research works are given below:

1. **Subject Relevance:** We selected only those papers that dealt with conflict detection in requirements and NLP. Further selection was done on the basis of the responses of the research questions that we asked for. Furthermore, we rejected unrelated research studies that did not include both conflict detection and NLP in them.
2. **2009-2018:** We ensured the collection of the latest studies by opting for those studies which lie in the years 2009 to 2018, and by not considering those researches that lie outside of our selected time range.
3. **Publishers:** Primarily four famous scientific databases were used, which are IEEE, ACM, ELSEVIER, and SPRINGER; to ensure the inclusion of authentic and state of the art research works we opted for those papers which have been brought forward by the specified publishers. Details are given in ***Table 2.1***.
4. **Result-oriented:** The studies that we opted for are model/algorithm oriented with focus on NLP.
5. **Redundancy:** We rejected redundant research studies and only most outstanding one of them was used.
6. **Valid models/algorithms used:** Selected researches that proposed algorithms or used existing models for conflict detection.

Table 2.1: Details of research works per database

Sr.#	Scientific Database	Type	Selected Research Works	No. of Researches
1	IEEE	Journal	0	0
		Conference	[4-12]	9
2	SPRINGER	Journal	0	0

		Conference	0	0
3	ELSEVIER	Journal	[13-16]	4
		Conference	[17, 18]	2
4	ACM	Journal	0	0
		Conference	[19-26]	8

2.1.3. Search Process

The selection and rejection criteria depicts that we have opted for four prime databases of publication (i.e. ACM, IEEE, Springer and Elsevier) to perform the systematic literature review process. We used "2009–2018" year-filter on all the search terms to get the searches put out during 2009–2018, merely. Some of the search terms included (e.g. Conflict detection, software requirements, inconsistency detection, automated requirement analysis, requirement analysis, and NLP) as mentioned in ***Table 2.2***. We used the "AND" operator to accomplish the possible investigation outcomes necessary for our study. We followed the search process flow diagram as illustrated in ***Figure 2.1***.

1. **Identification:** We specified multiple search expressions in four scientific databases and got about 36, 563 results.
2. **Screening:** We excluded 35,060 studies in the screening process because their KWs did not comply with our criteria. Plus, out of 1,503 we excluded 400 further studies because their title did not comply with our criteria
3. **Eligibility:** We considered 1,103 researches and by accessing their full text and by reading their abstracts and results we discarded 978 researches because they did not match with our selection and rejection criteria. For example., [27] presented an article based on our keywords but the study was incomprehensible, existing of a single page only and in no way could contribute to our research. We rejected this study because it did not meet our eligibility criteria of validation mentioned in Section 2.1.5
4. We performed a thorough qualitative and quantitative study of 122 researches by extracting their data and synthesizing it later for our research questions. After detailed

examination of our 122 papers we rejected 99 studies which did not fulfill our merit quantitative and qualitative criteria.

5. **Included:** We finally included remaining 23 papers because they fully comply with our set criteria for selection and rejection.

6. The details of selected research studies as per the publishers.

Table 2.2: Details of search terms and search results

Search Terms	Operator	IEEE	ACM	SPRINGER	ELSEVIER
Conflict, software requirements	AND	124	107	2,397	7,483
Inconsistency, software requirements	AND	60	65	607	2,858
Automated requirement analysis	AND	480	475	4,907	9,362
Natural language processing, requirement analysis	AND	74	71	2,010	4,497
Natural language processing, conflict detection, software requirement	AND	0	0	130	159
Natural language processing, inconsistency detection, software requirement	AND	0	0	46	109

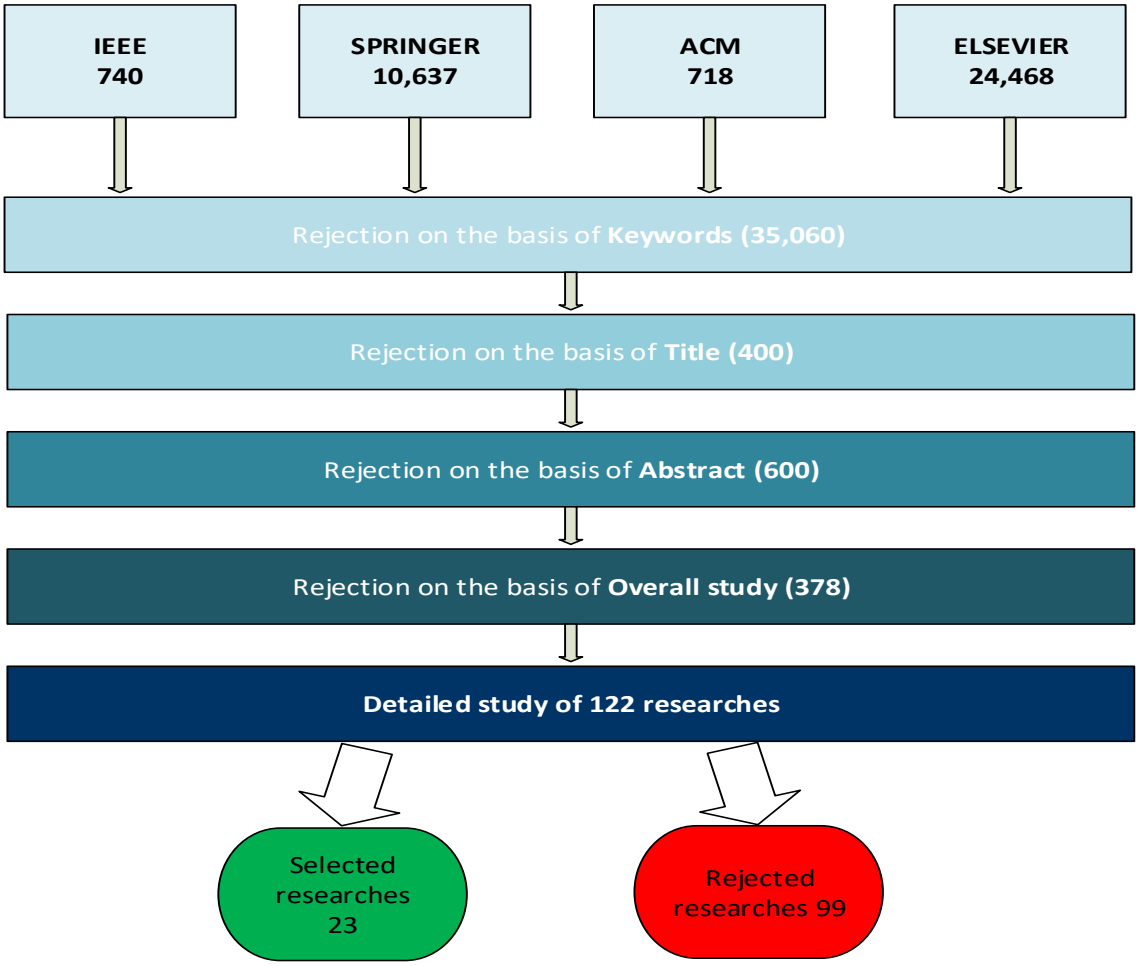

Figure 2.1: Search Process

2.1.4. Quality Assessment

We established the quality assessment criteria for understanding the importance of our result from the selected research studies. These criteria also help to define the trustworthiness of each research work we have selected and its fundamental discoveries:

1. The data evaluation of the researches is free from the ambiguous statements and relies on the solid facts and theoretical discerning.

2. Selected researches have been validated using appropriate validation techniques and approaches e.g. validation on some website or using case studies etc.

3. Tools information that has been used to perform different activities that helped us to validate our findings is provided.

4. We have clearly and logically prepared and sorted the research by focusing on themes or ideas rather than the authors.

5. Uniqueness of the study is another important feature. Therefore, we have only included those research studies that are published in at least one of the following four well-known and internationally recognized scientific databases which are: ACM, SPRINGER, IEEE, and ELSEVIER. Details are shown in ***Figure 2.2*** and ***Figure 2.3***.

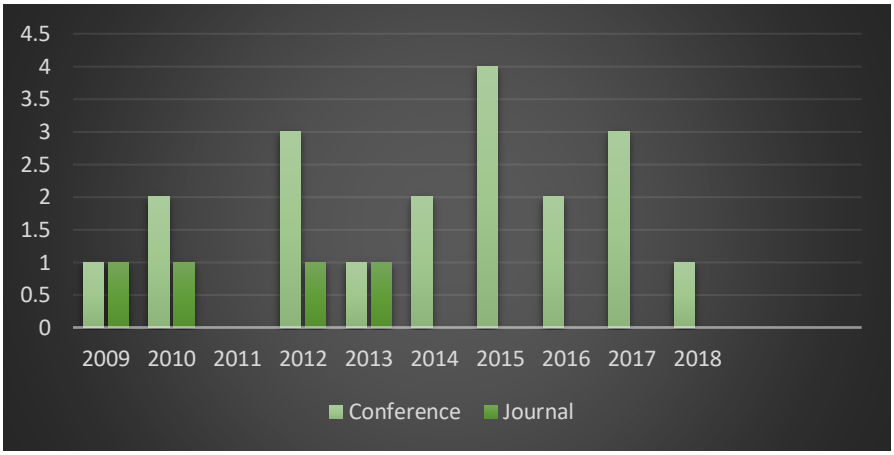

Figure 2.2: Selected Researches per Year

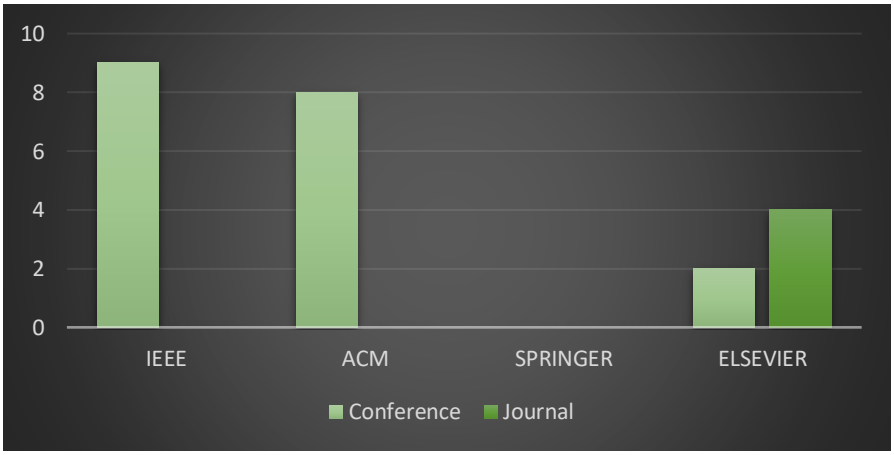

Figure 2.3: Selected Publishers per Year

During our data extraction and synthesis phase, quality of research studies reduced due to our constraints on research and specific keywords. We found that by the time last studies relative to our study were combined, the scientific library Springer produced zero studies as shown in ***Figure 2.3.***

2.1.5. Data Extraction and Synthesis

Table 2.3 shows the data extraction and synthesis performed for our nominated researches to attain the answers of our research questions. After the data extraction, we conducted an inclusive analysis on requirements, conflict detection, and NLP.

Table 2.3 contains the details of data extraction and synthesis. We have defined some parameters, from serial number 2 to 6 for data extraction, from which we extracted the details of each selected research study to make sure that it conforms to our selection and rejection criteria. We have defined some parameters, from serial number 7 to 9 for data synthesis, considering these parameters we have performed detailed investigation of each selected research study. Each selected research study has been studied and investigated in detail in order to assign it to the equivalent category. Each selected research study has been studied intensively in order to extract the correct information regarding the models/algorithms, tools, and data sets as defined in serial number 7 to 9 respectively.

Table 2.3: Data Extraction and Synthesis

Sr. #	Descriptions	Details
1	Bibliographic information	Author, Title, Publication Year, Publisher, Type of Research (Journal/Conference)
Data Extraction		
2	Overview	Main objective of the selected paper
3	Results	Results acquired from the selected paper
4	Data Collection	Qualitative and quantitative method used
5	Assumptions	To validate the outcome
6	Validation	Manual and Automated testing comparison
Data Synthesis		
7	Model/Algorithm selection	Models and Algorithms used for conflict detection

8	Tool Selection	Tools used for conflict detection
9	Case study/Data set	Requirements data set for conflict detection

2.2. Research and Analysis

We have determined this Systematic Literature from 23 significant research studies and then we organized the selected researches into three pre-defined categories (Section 2.1.1). This was done to acknowledge the relevant research work.

2.2.1. Conflict Detection Models and Algorithms

The data extracted to answer this question shows that machine learning, ontology based, AND Aspect Oriented Requirements Engineering (AORE) based models and algorithms have been cited by both journals and conferences while semantic and graphic analysis based models and algorithms have been used in conferences only.

In this research question, we have looked at the models and algorithms from the selected papers to find a correlation in the types of models and algorithms that have been used until now and have been used commonly. The studies using machine learning techniques were papers [9, 14, 23]. In papers [8, 10, 15, 17], ontology based models were used to detect conflicts, inconsistencies and other lexical issues in requirements. Mostly, machine learning models with regression modeling and ontology based models were used with few using comparative studies and graph analyses and semantic based models to find conflicts in requirements. The remaining papers [13, 16, 19, 25] used AORE modeling to match requirements to detect conflicts in them. Details of extracted data is shown in ***Table 2.4***.

Table 2.4: Conflict Detection Models and Algorithms

Sr. #	Conflict Detection Models	No. of References	References Identification
1	**Machine Learning** (Regression Linear Model, Multi-Sentence Modelling of Requirements, K-means)	3	[9, 14, 23]
2	**Ontology-based** (OWL, Generalized Upper Model, Domain Ontologies, Ontology of Uncertainty)	4	[8, 10, 15, 17]
3	**Data Models** (Semantic Data Models, Verb-centric General Semantic Model)	2	[10, 26]
4	**AOP** UML Models, OMG Models (MDA), Theme/Doc Approach, KAOS	4	[13, 16, 19, 25]
5	**Graph Analysis**	1	[4]
6	**Study-based**	2	[22, 25]
7	**Self-Proposed**	5	[7, 18, 23, 24, 26]
8	**Formal Method**	1	[20]
9	**Algebric Models**	1	[21]

2.2.2. Conflict Detection Tools

This part of our paper presents the tools that were found to be used along with models and algorithms in detection of conflicts and inconsistencies in software requirements. Tools used were collected from the selected 23 research papers for our study as shown in ***Table 2.5***.

Tools used in the selected studies have been selected on the basis of their usage for extracting and analyzing conflicts or inconsistencies in requirements. Tools selected belong to both the research and public sector. Most of the tools are automated (without much focus on conflicts or specific use of NLP) like Algebraic Grammar Graph, GATE, cTAKES, Open NLP, ReVerb, GUITAR, SEMIOS, and SAT-Analyzer. Manual tools include: text based retrieval system and ReqWiki. The only tool that is semi-automated is the Drools Expert. The tools are also categorized as analyzing tools, extracting tools, or both in the Knowledge category. Most of the tools like GATE, Open NLP, and Stanford Core NLP are JAVA based Natural Language Processing Tools.

Table 2.5: Conflict Detection Tools

Sr. #	Conflict Detection Tools	Research Identification
1	Drools Expert	[23]
2	Text Based Retrieval System	[18]
3	General Architecture for Text Engineering (GATE)	[23]
4	Algebric Grammar Graph (AGG)	[19]
5	cTAKES	[21]
6	ReVerb	[17]
7	Goal-Use case Integration Tool for Analysis of Requirements (GUITAR)	[8]
8	SEMIOS	[9]
9	ReqWiki	[25]
10	SAT-Analyzer	[12]

2.2.3. Conflict Detection Specific Case Studies/Data Sets

The third research question relative to our study is the collection of specific data sets and case studies that were used in the selected papers. This section includes a variety of categories from public repositories of data to industry-specific requirements as shown in ***Table 2.6***.

From the selected 23 researches, a total of 14 studies focused on data sets whereas others had proposed models and algorithms or focused mainly on a collection of tools and models for comparison or proposed algorithms related to their study.

Table 2.6: Conflict Detection specific Case Studies/Data Sets

Sr.#	Case Study / Data Set	No. of References	Research Identification
1	**PROMISE Repository**	2	[4, 23]
2	**Industry Specific** (Italian aerospace and defense company, Clinical notes, Automotive specifications from Mercedes-Benz car development, Slot machine, Confidential aeronautic and automobile data, Telecommunication Company)	6	[6, 9, 13, 21, 24, 28]
3	**Program/System Specific** (Remote Patient Monitoring System (RPMS), Mozilla & MP- a business application, ATM system & Complaint System in banks, hostel management system & Coach tour management system)	5	[10, 12, 14, 19]
4	**Online Data Collection**	1	[17]
5	**Own Data Set**	1	[5]

2.3. Research Gaps

This section discusses the research gaps encountered and their possible solutions. We have collected 23 researches after an extensive screening process to look for researches that provide automated conflict detection for software requirements. The selected studies have varying test cases from software requirements, some are industry specific while others have used random sentences as test cases. The studies also show how few have focused on the use of tools altogether. ***Table 2.7*** provides an analytical view of the comparison of all 23 researches.

The gap found in our selected studies is that there is no research that worked on conflict detection using NLP and rule based model nor proposed an automated approach that combines both our research criteria. However, there are two semi-automated [4, 26] approaches and two automated approaches: one is a tool [8] and the other is focused on analyzing LTL patterns using NLP [5].

Table 2.7: Comparison of Selected Researches

Sr.#	Research	Tool Support	Model		Language Specification	Conflict Specification	Types of Target		Case Study Implementation
			Developed	Proposed					
1	[13]	×	✓	-	×	×	✓	✓	✓
2	[20]	×	✓	-	×	×	✓	✓	×
3	[23]	✓	-	✓	✓	×	✓	×	✓
4	[26]	✓	-	✓	✓	×	✓	×	×
5	[25]	✓	✓	-	×	×	✓	×	✓
6	[11]	×	✓	-	×	✓	✓	✓	×
7	[19]	✓	×	×	×	×	✓	×	✓
8	[21]	✓	✓	-	✓	×	✓	×	✓
9	[6]	✓	-	✓	✓	×	✓	×	×
10	[5]	✓	✓	-	✓	×	✓	✓	✓

11	[8]	✓	✓	-	×	×	✓	✓	×
12	[9]	✓	✓	-	×	×	✓	×	✓
13	[18]	✓	-	✓	✓	×	✓	×	×
14	[4]	✓	-	-	✓	×	✓	✓	✓
15	[7]	✓	-	✓	✓	×	✓	×	×
16	[24]	✓	✓	-	✓	×	✓	×	✓
17	[10]	×	✓	-	×	×	✓	×	✓
18	[28]	×	✓	-	×	×	✓	✓	✓
19	[17]	✓	-	✓	✓	×	✓	×	✓
20	[16]	×	-	✓	✓	×	✓	×	×
21	[14]	×	✓	-	×	×	✓	✓	✓
22	[15]	×	✓	-	×	×	✓	✓	✓
23	[12]	✓	-	✓	✓	×	✓	✓	×

Chapter 3

Proposed Methodology

3. Chapter 3: Proposed Methodology

This chapter contains details of the proposed methodology. Section 3.1 discusses the targeted problem and Section 3.2 provides detailed proposed solution.

3.1. Targeted Problem

Requirements elicitation is a step in the Software Development Lifecycle (SDLC). In fact, it is the first phase of the software development lifecycle through which quality software can be developed according to the customers' need and handed over to them in any given time [29] as shown in ***Figure 3.1***.

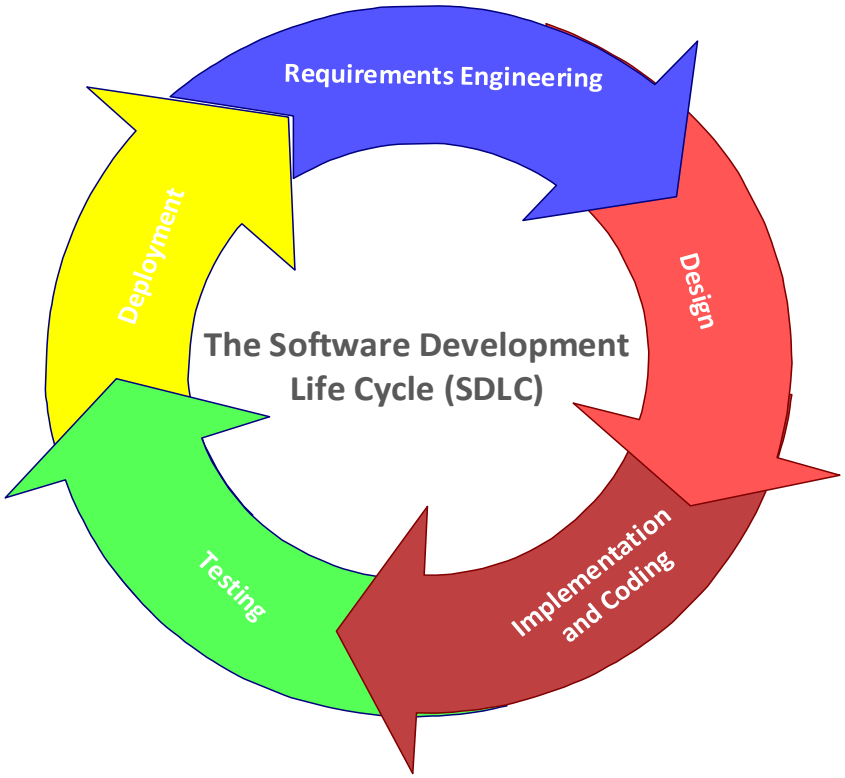

Figure 3.1: The Software Development Life Cycle (SDLC)

Figure 3.1 shows the five major phases of the SDLC. In detail, there are seven major phases in the SDLC starting from planning, requirements, design, development, testing, deployment, and maintenance as shown in ***Figure 3.2.***

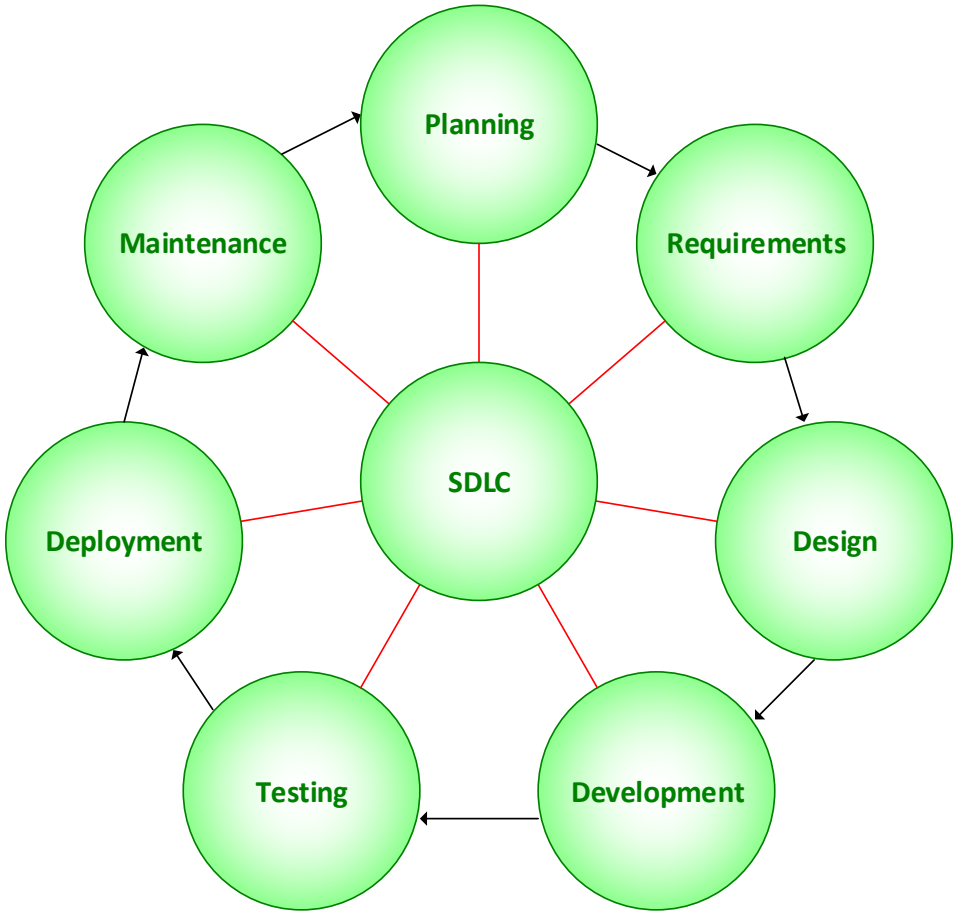

Figure 3.2: All Stages of SDLC

The basic purpose of requirements elicitation is to extract requirements of every scope from the client and then to process those given requirements into a well-developed requirements specification document which is then passed off to the next step and becomes the basis of a baseline upon which all further phases are completed and the eventual software product developed. Usually, this phase requires the longest time to complete because of the requirement engineers' going back and forth with the clients' or the requirement team from the clients' end to sort out all requirements before they can be closed off and passed on to the next phase as a final requirements specification document. This is done to weed out problems that may arise due to human error such as ambiguity

in meaning, conflict and inconsistency in needs, or incompatible priorities of multiples stakeholders. We have focused on the inconsistent and conflicting requirements during the requirements analysis step in the requirements engineering phase. ***Figure 3.3*** shows all the steps in a requirements engineering phase.

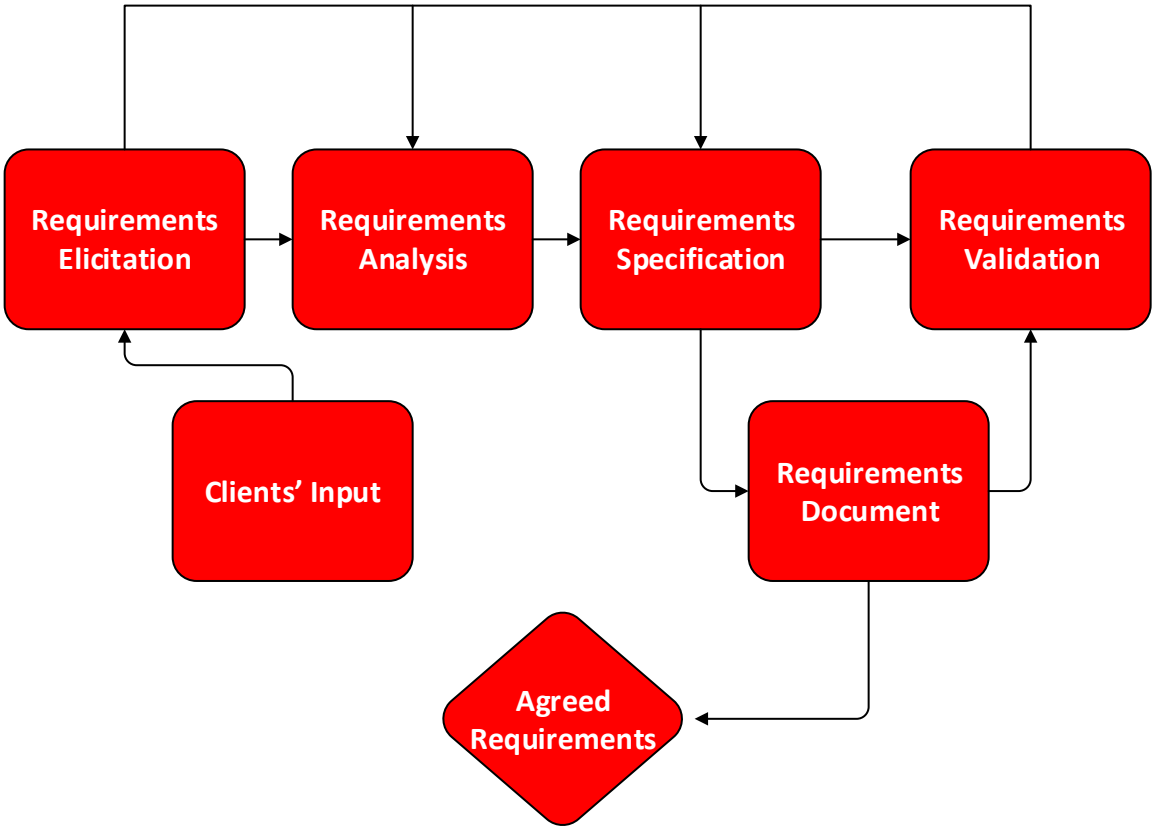

Figure 3.3: Requirements Engineering Process

3.1.1. Requirements Engineering Phases

In the requirements engineering phase, all steps are iterative. Most importantly, at the center is the requirements analysis step in which requirements analysts read the gathered set of requirements, highlight problems, and discuss them in requirements review meetings [30]. This is time consuming manual process with countless resources being utilized again and again. We have focused on the requirements analysis, an iterative step of requirements engineering from the

software development life cycle to elaborate on a solution which is automated in detecting conflicts while using the traditional methods of contextual rules, but automated.

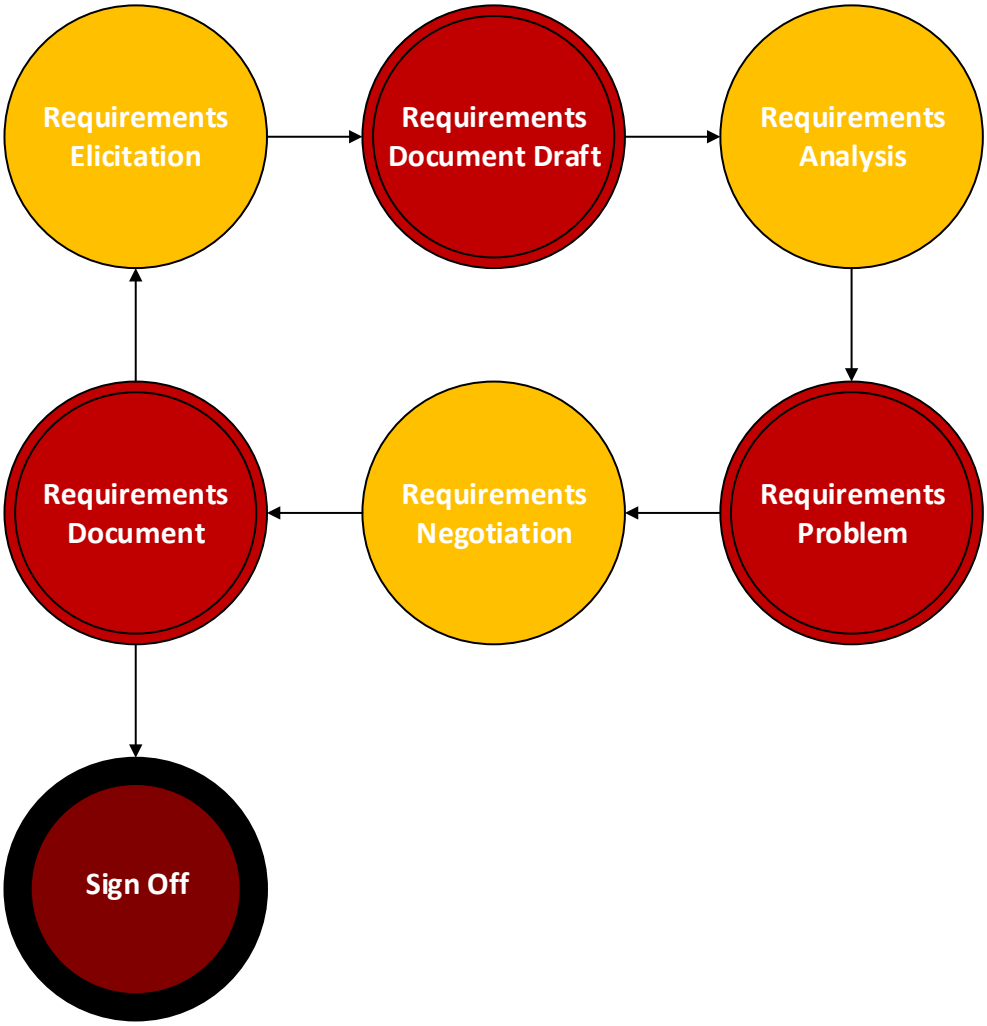

Figure 3.4: Iterative Requirement Engineering Process

3.1.2. Detecting Conflicts during Requirements Analysis

In any product development, a concise and true requirements document is vital before the development phases can commence. In requirements engineering, one of the steps is the consistency checking and conflict detection of requirements to solve contradictory requirements issue which can later on impact the development of a software product. The quality of the requirements phase effects the overall quality of the subsequent phases and hence, the software

product. Having a good software requirements specification (SRS) document is essential to a good final product.

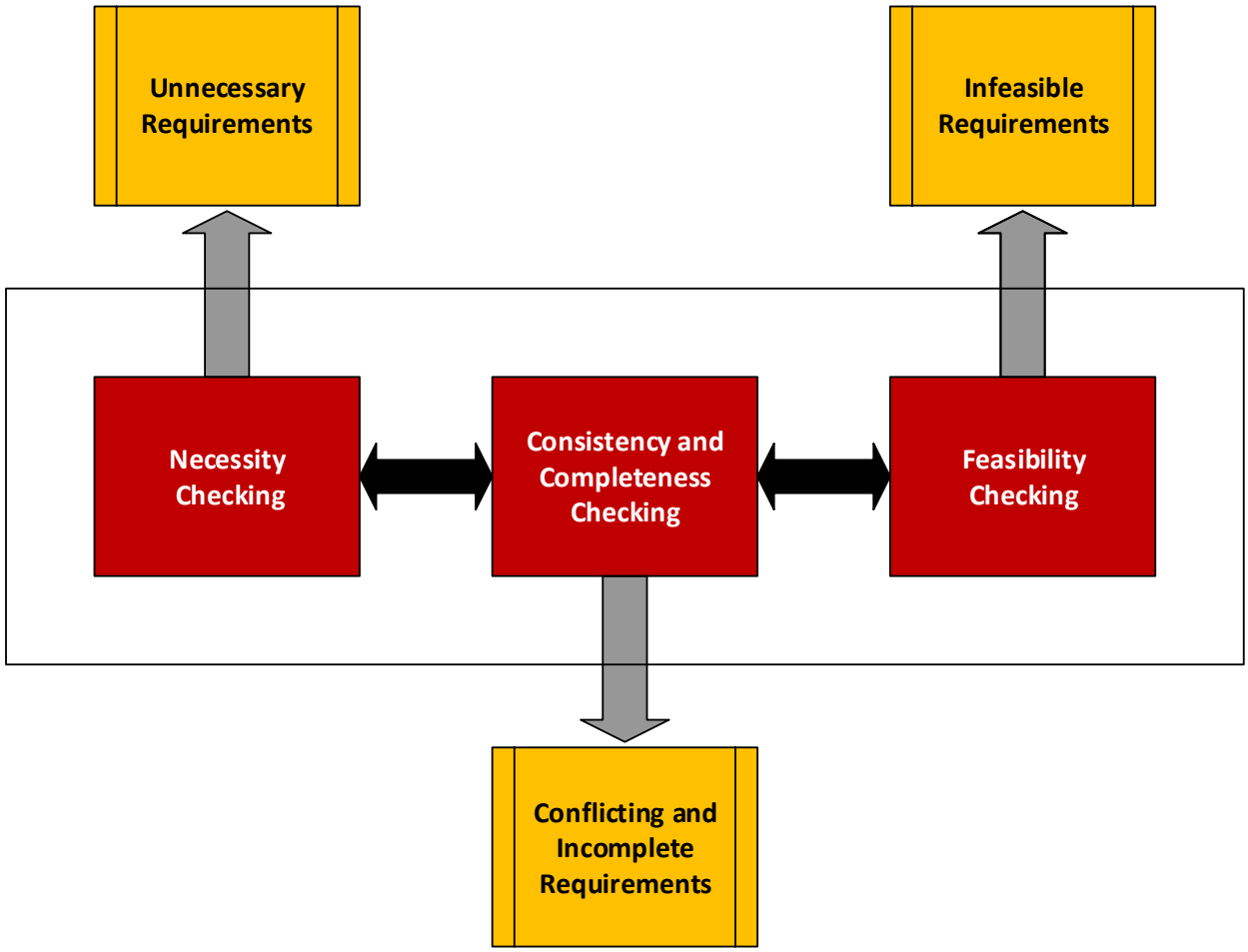

Figure 3.5: Requirements Analysis Process

3.2. Proposed Solution

We have proposed an automated algorithm based on Natural Language Processing (NLP) techniques. NLP techniques include contextual rule-based commands like Parts of Speech (PoS) tagging, word tokenization, lemmatization, and stop word removal amongst many others. NLP techniques can be used by importing Natural Language Processing Toolkit (NLTK), which is the most generally used toolkit available and can be used on various platforms.

Our work starts off with inputting a series of lines together in a joint thread of text, separated by full-stops. In order to separate each requirement as an individual sentence, first of the NLP technique is used- that is sentence tokenization. This is followed by splitting the complete

text into a separate line for each requirement. This way we get a list of requirements. Eventually we want to compare each requirement against every other requirement so as to form an RxR matrix of comparisons that is the end result of our algorithm. After slitting the requirements into a separate line for each requirement, we have used tokenization to separate each word which translate into an array of items in every row. This will help us extract unnecessary words that can be ignored like the pronouns and the articles. Another NLP technique being used is the Chunker Parser which is programmed to sort out a series of tokenized words for each requirement, relative to our algorithm, on which we have applied our series of contextual rules of comparison to detect conflict eventually.

We have focused on going back and forth with NLP techniques to fully isolate context from each requirement after careful isolation of the requirement itself. Normally, NLP techniques are used in a sequence but our algorithm requires a lot of going back and forth for the eventual use of contextual rules, something that has made our algorithm successful and automated.

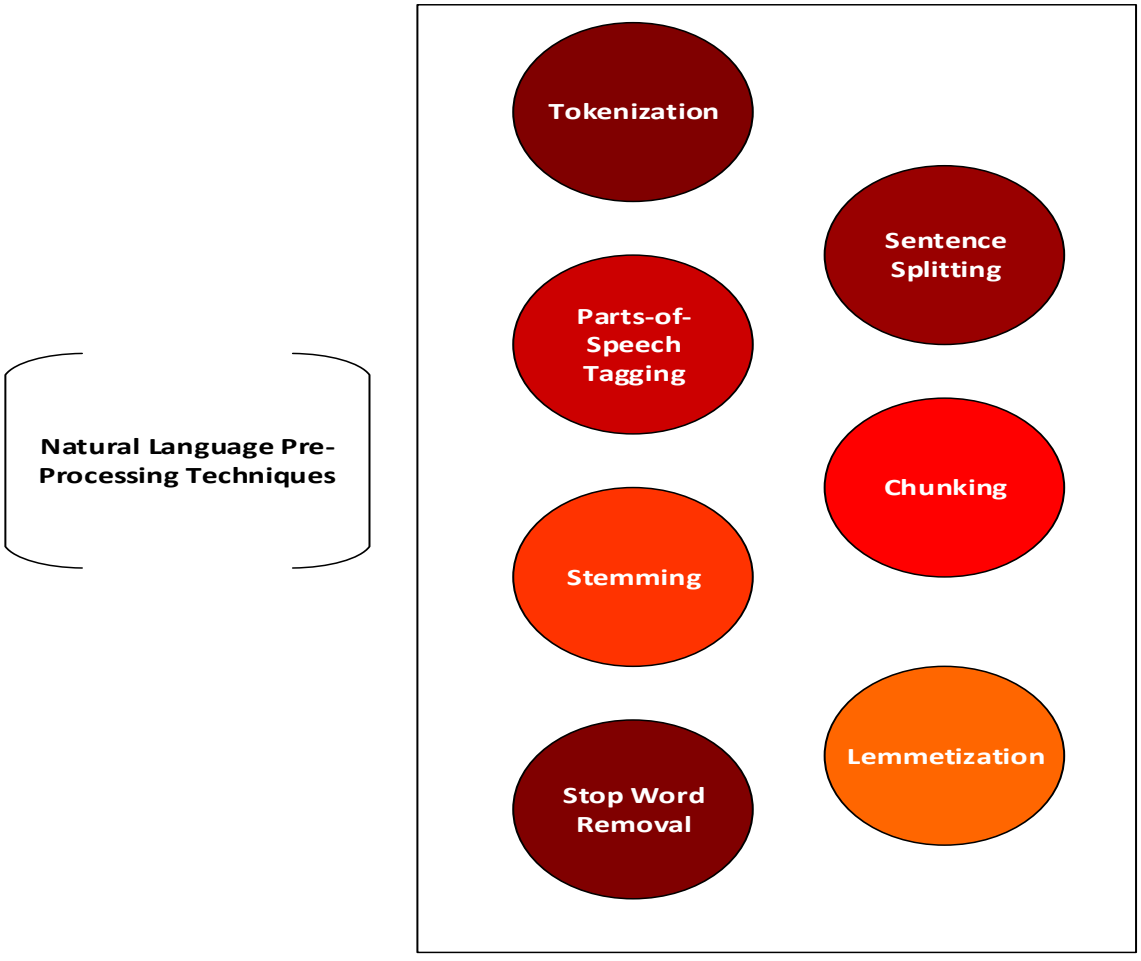

Figure 3.6: Commonly used NLP Techniques

Chapter 4

Implementation

4. Chapter 4: Implementation

In this chapter we present the implementation phase of our work which contains the use of NLP techniques, followed by contextual rules application and eventual RxR matrix mapping of detected conflicts. Firstly pre-processing on input set of requirements is discussed in Section 4.2. This is followed by discussion of application of Automated Contextual Rules in Section 4.3. Then the complete architecture of our automated conflict detection is discussed in Section 4.4.

4.1. Requirements Specification

The first part of selection of our requirements is based upon the structure of the sentence. We propose an approach that extracts chunks from each requirement based on the position of noun and verb in the sentence, for this, we have specific requirements that make up a clause instead of a phrase to have inclusion of both the noun and the verb. Another requirement specification is the written voice of the sentence from two categories: Active Vs Passive Voice. These requirement specifications are necessary for the logical extraction of processed data from the input requirements which will later on be indexed in an RxR matrix for comparison and then yielding the results that will determine if there exists a conflict between two requirements. The flow of requirements specification is shown in **Figure *4.1***

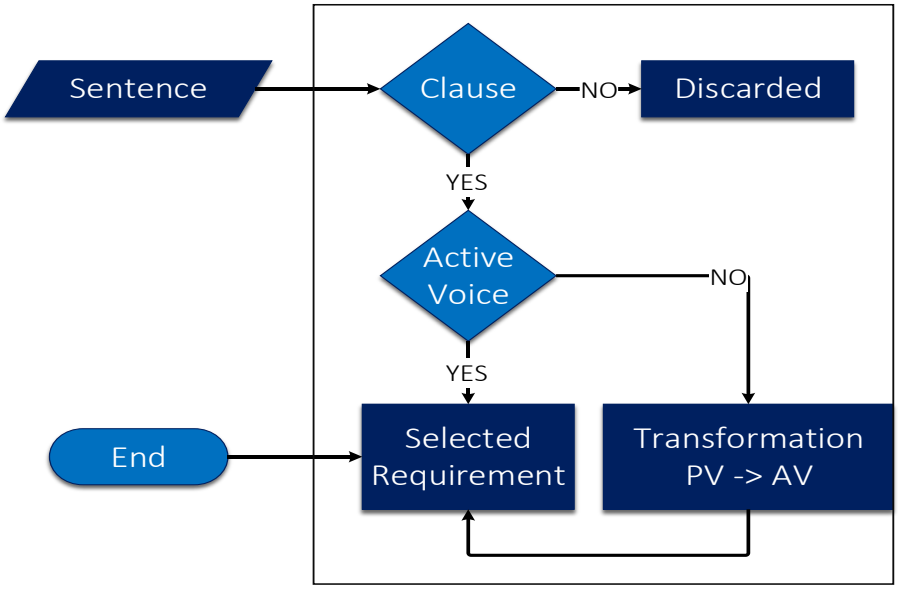

Figure 4.1: Requirements Specification

4.2. NLP Pre-Processing Techniques

We have used a self-provided data set of ten software requirements from a Course Management System (CMS) as our training data set. The requirements are stored in a list directly. This is then followed by pre-processing techniques of NLP to extract contextual data which is to be used for contextual rule analysis later on. **Figure *4.2*** shows the specific techniques used during the pre-processing step of our model.

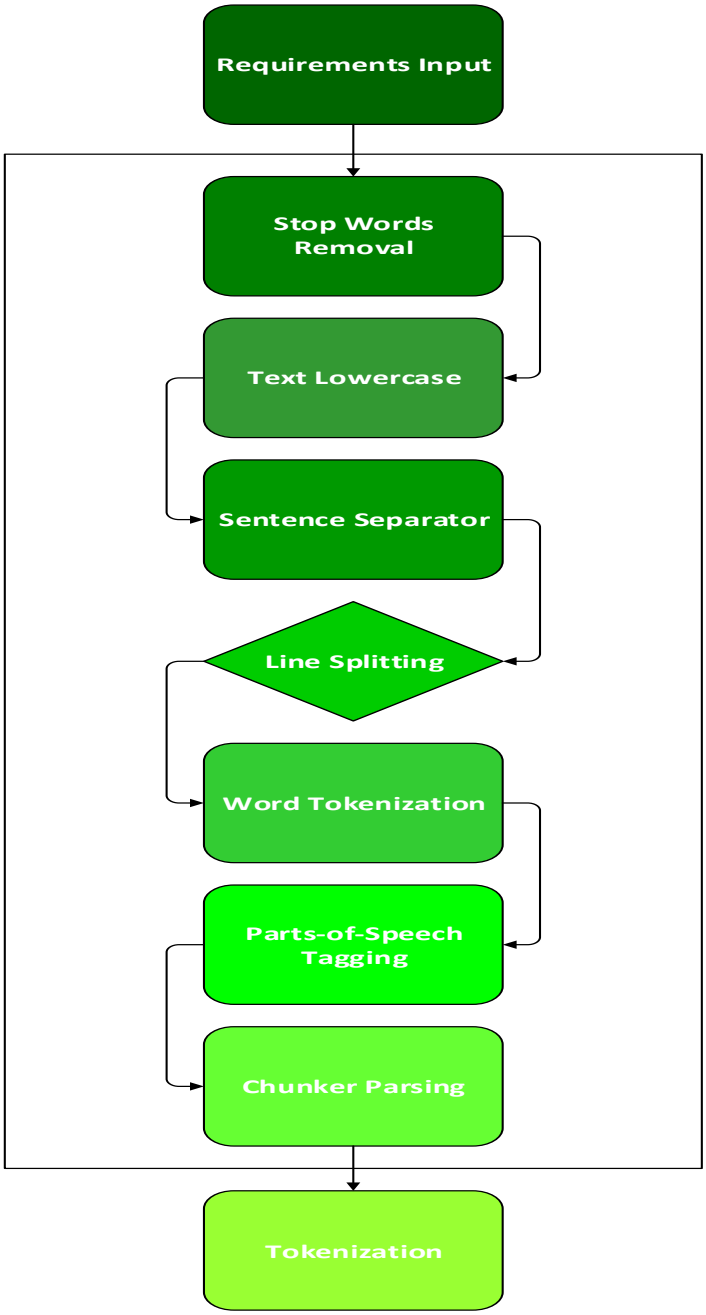

Figure 4.2: NLP Pre-Processing Techniques used

Figure *4.2* shows the first leg of our algorithm, where after the input requirements are stored, a series of NLP pre-processing techniques are applied to extract the contextual data we need for our eventual conflict detection. Details of the used NLP techniques are explained below:

Stop Words Removal: It includes taking text as input and removing stop words like articles: "The, a, an, in" pronouns, "he, she, and it" or helping words from the text by using stopwords.word(). The resultant text is a string of words that make up the context of the text.

Text Lowercase: This includes taking text and converting any or all uppercase words into smaller case words in the text by using text.lower(). This is done to avoid using different indexes for the same word. Text lowercase is a part of word tokenization and stemming to convert a given word to its root form.

Sentence Tokenization: Sentence tokenization is done to separate text into a number of indexes which is equal to the number of sentences present in the original input text corpus by using sent.tokenize().

Word Tokenization: This is done to extract tokens from string of characters (text) by using tokenize.word() method. It splits words based on white space and punctuation and results in a tokenized word.

Parts-of-Speech (PoS) Tagging: It includes reading text and then assigning parts of speech to each tokenized word by using pos_tag(). Each word is displayed separately along with its part of speech such as: verb, noun, adjective, etc.

Chunking Parser: It includes chunking together words based on a series of rules parsed to it. PoS tagging is usually followed by Chunker parsing where the PoS tagged words are passed to the Chunker function to add more structure to the already processed tokenized words.

4.3. Transformation Contextual Rules

In this section, we have explained the automated text classification into three labels and then application of contextual rules that are applied on our NLP pre-processed text extracted after performing a series of NLP techniques.

4.3.1. Extracted Text Classification

Mapping the extracted words from each requirements into a series of labels, we identify Subject, Action, and Object. We have set these three labels against each extracted word based on

the *Chunking Parser* technique in the last step of using NLP techniques. This provides us with a set of labels for each requirement, against which later on, we will compare every other requirement. Setting labels identifies the different *PoS tagged* words from each requirement.

4.3.2. Contextual Rules

We have developed a set of contextual rules based on identified words against labels. We have classified labels such as: Subject, Action, and Object for each extracted word from our data set of requirements. For our approach, we focus only on the direct conflict that occurs for these three specified labels.

Table 4.1: Contextual Rules

Rule #	**Contextual Rules**			**Conflict Detection (Conflict/No Conflict)**
Rule 1	Subject	=	Subject	No Conflict
	Action	=	Action	
	Object	=	Object	
Rule 2	Subject	=	Subject	Conflict
	Action	≠	Action	
	Object	=	Object	
Rule 3	Subject	=	Subject	No Conflict
	Action	=	Action	
	Object	≠	Object	
Rule 4	Subject	=	Subject	No Conflict

	Action	≠	Action	
	Object	≠	Object	
Rule 5	Subject	≠	Subject	Conflict
	Action	=	Action	
	Object	=	Object	
Rule 6	Subject	≠	Subject	No Conflict
	Action	≠	Action	
	Object	=	Object	
Rule 7	Subject	≠	Subject	No Conflict
	Action	=	Action	
	Object	≠	Object	
Rule 8	Subject	≠	Subject	No Conflict
	Action	≠	Action	
	Object	≠	Object	

In ***Table 4.1***, we have presented the 8 contextual rules that will automate the conflict detection on our selected requirements data set, with the initial modeling applied beforehand. These rules are developed on the basis of extracted context set against the three identified labels of Subject, Action, and Object. In this way of extracting only the informational text from a requirement, we lose the possibility of Mis-identifying context. Mis-identified text against a large

set of labels or without labels can change the context of a requirement during comparison modeling and the conflict is improperly identified or not identified at all.

4.4. Conflict Detection Architecture

Complete architecture of our automated conflict detection is described in detail in **Figure *4.3***. We have implemented an automated conflict detection architecture based on algorithm based test case idea. This conflict detection architecture fully automates the conflict detection process in the Requirements Engineering phase in the SDLC by providing automated comparison of test case. Our conflict detection architecture is composed of three main components which are NLP techniques, Extracted words labeling and classification, and Contextual Rules application. Details of each component in a flow are described below.

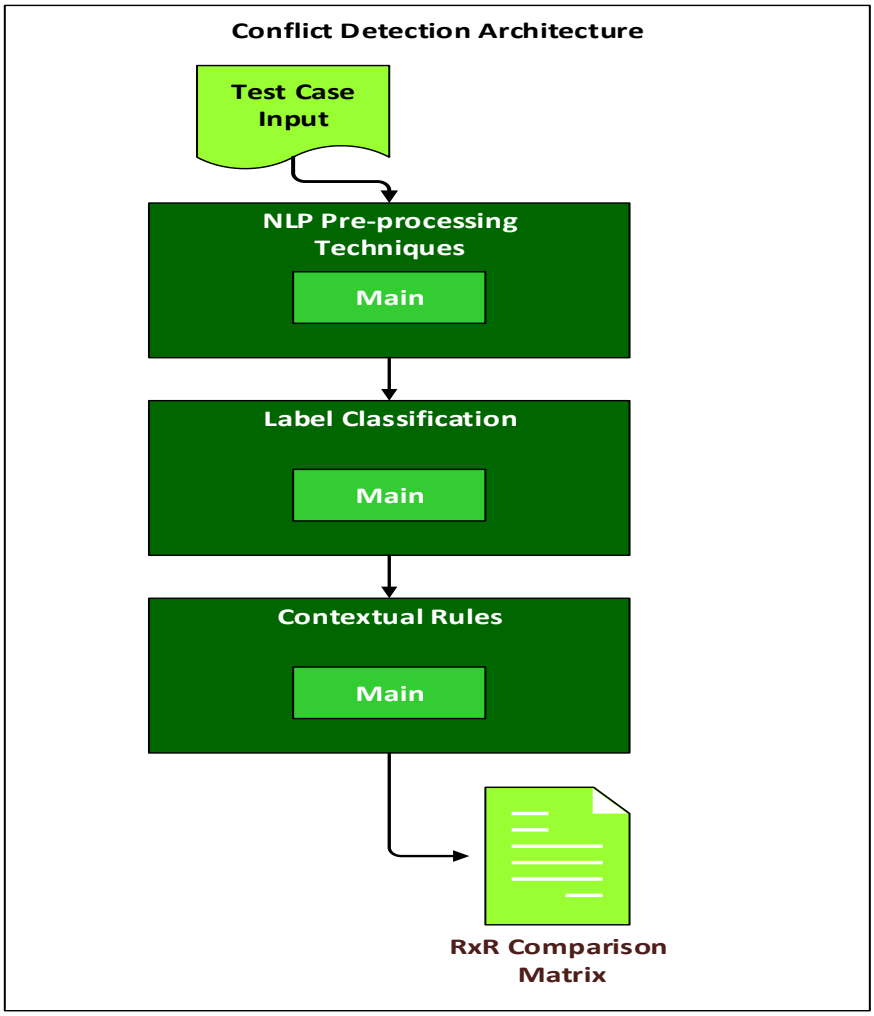

Figure 4.3: Architecture of Automated Conflict Detection

Test Case Input: Test cases are a set of requirements fed into the input variable "text" directly as a series of sentences that will be separated afterwards.

NLP Pre-processing: Starting from main sequence, the pre-input test case is split and separated into R number of lines per requirement. This is followed by a series of NLP pre-processing techniques as presented in **Figure *4.2***.

Label Classification: NLP pre-processing on test case is followed by label classification on the resultant extracted words from each test case requirement. This will be vital during the conflict detection contextual rules because label classification is done to arrange the words in a set sequence.

Contextual Rules: The final step is the application of a set of contextual rules on the extracted set of labeled and extracted words from the initial test case. The rules defined in ***Table 4.1*** are applied against a matrix of RxR where each requirement is compared against every other requirement to detect conflicts according to the contextual rule set.

After successful flow of functions, the architecture gives an RxR matrix with results such as "Conflict" or "No Conflict" for each requirement against every other. For ease of showing, we have worked to display the resultant RxR matrix as a series of sentences displaying the end result for every comparison made between the test case requirements.

Chapter 5

Validation

5. Chapter 5: Verification and Validation

In this chapter, the applicability and validity of our proposed approach is presented with the help of a CMS based requirements data set used as our test case along with a manual and automated testing comparison. Section 5.1 presents the case study used for training of our algorithm and Section 5.2 presents the case study chosen for our test case.

5.1. CMS Requirements Data Set

The data set for our research and implementation has been explained and validated using four sections. Section 5.1.1 covers the requirement specification of our CMS requirements specific to software requirements. Section 5.1.2 presents the NLP techniques pre-processing on our data set using the NLTK in the tool PyCharm, Classification of extracted words into set labels, and the last step of contextual rules application on the classified and extracted use case. Lastly, the resultant automated conflict detection verification is presented in 2 parts in Section 5.1.3. The Section 5.1.3.1 presents the automated conflict detection and the Section 5.1.3.2 presents the manual conflict detection respectively. Section 5.1.3.3 gives a comparison analysis of both testing methods used in our approach.

5.1.1. Requirement Specification

The purpose of our study is the detection of conflicts in software requirements during the Requirements Engineering phase of SDLC [29]. To validate our study, we have chosen ten software requirements from a Course Management System. The requirements are selected on a basis of requirement specifications that fit our criteria as shown in ***Figure 5.1***.

0. System should allow students to view their attendance.
1. System should allow students to view their grades.
2. System should allow students to view their CGPA.
3. System should allow facultymembers to add grades.
4. System should allow facultymembers to edit grades.
5. System should allow facultymembers to mark attendance.
6. Student grades should only be edited by deputycontroller once facultymembers submit results.
7. Whenever facultymembers request, the system should estimate CGPA.
8. After training of two hours, everyone should be able, like users to access the system.
9. System should allow facultymembers to change their passwords.

Figure 5.1: Requirements Specification

Figure 5.1 shows the carefully selected requirements based on the requirements specification set for the requirements that will be used as test case in our approach.

5.1.2. Conflict Detection Algorithm

In this section, we have discussed the complete flow of our architecture along with the models, techniques and processes used. ***Figure 4.3*** shows the flow process of our conflict detection algorithm. In **Section 5.1.2.1** the NLP techniques pre-processing on our data set using the NLTK in the tool PyCharm are discussed. **Section 5.1.2.2** discusses the classification of extracted words into set labels. In section **5.1.2.3** the last step of contextual rules application on the classified and extracted use case is discussed.

5.1.2.1. NLP Pre-Processing Techniques

Starting off, we used the libraries from the Natural Language Tool Kit (NLTK) in the PyCharm setup as shown in ***Figure 5.2.*** This section presents the all-around of NLP techniques in our implementation phase of our study as shown in ***Figure 5.3***, ***Figure 5.4***, and ***Figure 5.5***.

```
2
3   from nltk import *
4   import nltk
5   from nltk.corpus import stopwords
6   from nltk.tokenize import word_tokenize, sent_tokenize
7
```

Figure 5.2: Libraries used in Conflict Detection Algorithm

```
stopWords = set(stopwords.words("english"))
text = text.lower()
sentences = sent_tokenize(text)

```

Figure 5.4: NLP Pre-Processing Techniques Part 1

```

for f in sentences:
    for line in f.splitlines():
        newline = line
        print("\nNEW SENTENCE: ", newline)
        tokens = word_tokenize(newline)
        print(tokens)
        words = [i for i in tokens if not i in stopWords]
        print(words)
        poswords = pos_tag(words)
        print(poswords)
        R = poswords
        print("\nR",a," ",poswords)
```

Figure 5.3: NLP Pre-Processing Techniques Part 2

```

chunkGram = r"""Chunk: {<NN.?>*<VB.?>*<NN.?>?}""" #any form of nouns and looking for 0 or more of these nouns in the sentence, <NN.?> any noun

chunkParser = nltk.RegexpParser(chunkGram)
chunked = chunkParser.parse(poswords)
print("\n", chunked)
A=[]
output = []
for subtree in chunked.subtrees(filter=lambda t: t.label() == 'Chunk'):
   output.append(" ".join([a for (a,b) in subtree.leaves()]))
print("\nOUTPUT:", output)
mystring=output[1]
print(mystring)

```

Figure 5.5: NLP Pre-Processing Techniques Part 3

Figure 5.3 presents the use of NLP techniques: stop words removal, lowercase transition, and tokenization. These are the pre-processing steps from our NLP selected techniques that are performed on the whole sentence. ***Figure 5.4*** then presents the splitting of input sentences into new lines as a pretext to storing our test cases into a matrix. Finally, ***Figure 5.5*** presents the altered chunkParser that is performed on the data to extract our final test cases.

5.1.2.2. Classification of Labels

NLP pre-processing on our data set is followed by classification of labels. This step of the algorithm also deals with excess of extracted words that don't meet our standard set of three-specific set of word and can cause problem during comparison phase of the algorithm in the flow of our architecture. ***Figure 5.6*** and ***Figure 5.7*** present the steps used for this step.

```
                    z = 0
                    output[ll] = L #output and putput[ll] are same array
                    print("\nOutput After Labels:", output[ll])
                    ll =+ll
                    break
            if ll <10:
                print("Final Output Array:", output[ll])
                M = output[ll]
                ll = +ll
            a += 1
        if m <4:
            print("Matrix[m][n]:", M)
            m = +m
        Matrix.append(M)
        print ("Final Matrix: ", Matrix)
        m = +m
```

Figure 5.6: Classification of Labels Part 1

```
for i in output:
    print("output:", i) #['system', 'students view attendance']
    for j in i:
        L = word_tokenize(i)
        print("Tokenized", L) #['system', ['students', 'view', 'attendance']]
        for k in L:
            if z == 0:
                x = L[z]
                xl = ['Subject']
                xl.append(x)
                print (xl)
            elif z == 1:
                x = L[z]
                x2 = ['Action']
                x2.append(x)
                print (x2)
            elif z == 2:
                x = L[z]
                x3 = ['Object']
                x3.append(x)
                print (x3)
            else:
                print("\nBreak\n")
            z = z + 1
```

Figure 5.7: Classification of Labels Part 2

5.1.2.3. Contextual Rules

Following NLP pre-processing and classification of labels, we get a new list of processed requirements, as shown in ***Figure 5.8.***

```
Final Matrix
['students', 'view', 'attendance']
['students', 'view', 'grades']
['students', 'view', 'cgpa']
['facultymembers', 'add', 'grades']
['facultymembers', 'edit', 'grades']
['facultymembers', 'mark', 'attendance']
['facultymembers', 'submit', 'results']
['system', 'estimate', 'cgpa']
['users', 'access', 'system']
['facultymembers', 'change', 'passwords']
```

Figure 5.8: Processed Requirements List

Following the transformation of test cases into our required set of requirements, we applied a series of contextual rules on the resultant list of strings. We discussed earlier about having set three labels for classification, and following this, the rules were devised on the basis of 2^3 which came to a total of 8 sets of rules as shown in ***Table 4.1***. Further showing the steps in our algorithm, ***Figure 5.9, Figure 5.10,*** and ***Figure 5.11*** show the coded format of applying conditional contextual rules.

```
while x < 4:
    while z < 10:
#1
        if Matrix[x][y] == Matrix[z][y]:
            if Matrix[x][y+1] == Matrix[z][y+1]:
                if Matrix[x][y+2] == Matrix[z][y+2]:
                    print ("Rule #1 There is No Conflict between", Matrix[x], "and ", Matrix[z])
                    r += r
                    k += k
                    c = c+1
#2
        if Matrix[x][y] == Matrix[z][y]:
            if Matrix[x][y+1] != Matrix[z][y+1]:
                if Matrix[x][y+2] == Matrix[z][y+2]:
                    print ("Rule #2 There is Conflict between", Matrix[x], "and ", Matrix[z])
                    r += r
                    k += k
                    c = c+1
#3
        if Matrix[x][y] == Matrix[z][y]:
            if Matrix[x][y+1] == Matrix[z][y+1]:
                if Matrix[x][y+2] != Matrix[z][y+2]:
                    print ("Rule #3 There is No Conflict between", Matrix[x], "and ", Matrix[z])
                    r += r
                    k += k
                    c = c+1
```

Figure 5.9: Contextual Rules Part 1

```
#4
        if Matrix[x][y] == Matrix[z][y]:
            if Matrix[x][y+1] != Matrix[z][y+1]:
                if Matrix[x][y+2] != Matrix[z][y+2]:
                    print ("Rule #4 There is No Conflict between", Matrix[x], "and ", Matrix[z])
                    r += r
                    k += k
                    c = c+1
#5
        if Matrix[x][y] != Matrix[z][y]:
            if Matrix[x][y+1] == Matrix[z][y+1]:
                if Matrix[x][y+2] == Matrix[z][y+2]:
                    print ("Rule #5 There is Conflict between", Matrix[x], "and ", Matrix[z])
                    r += r
                    k += k
                    c = c+1
#6
        if Matrix[x][y] != Matrix[z][y]:
            if Matrix[x][y+1] == Matrix[z][y+1]:
                if Matrix[x][y+2] != Matrix[z][y+2]:
                    print ("Rule #6 There is No Conflict between", Matrix[x], "and ", Matrix[z])
                    r += r
                    k += k
                    c = c+1
```

Figure 5.11: Contextual Rules Part 2

```
#7
        if Matrix[x][y] != Matrix[z][y]:
            if Matrix[x][y+1] != Matrix[z][y+1]:
                if Matrix[x][y+2] == Matrix[z][y+2]:
                    print ("Rule #7 There is No Conflict between", Matrix[x], "and ", Matrix[z])
                    r += r
                    k += k
                    c = c+1
#8
        if c==0:
            print ("Rule #8 There is No Conflict between", Matrix[x], "and ", Matrix[z])
            r += r
            k += k
        z = z+1
        c =0
    x=x+1
    z=0
```

Figure 5.10: Contextual Rules Part 3

5.1.3. Conflict Detection Verification

For verification of our algorithm, we have devised two-standard testing verification. First we evaluated the use case and manually calculated the conflicts amongst each requirement which are explained in **Section 5.1.3.1**. After the complete of our algorithm was achieved and conflicts detected in an automated manner, we saved those results, which are explained in **Section 5.1.3.2**. The comparison of both our testing methods is further evaluated, compared, and explained in **Section 5.1.3.3**.

5.1.3.1. Automated Results

When the comparison is performed on the final set of processed requirements, we get a direct display of sentences presenting whether there is a conflict between RxR. ***Figure 5.12, Figure 5.13, Figure 5.14, Figure 5.16, Figure 5.16, Figure 5.17, Figure 5.18, Figure 5.19, Figure 5.20, Figure 5.21*** show the automated results that we have achieved for all ten specified requirements as our test case.

```
Requirement# 0
R( 0 ) x R( 0 )
Rule #1 There is No Conflict between ['students', 'view', 'attendance'] and  ['students', 'view', 'attendance']
R( 0 ) x R( 1 )
Rule #3 There is No Conflict between ['students', 'view', 'attendance'] and  ['students', 'view', 'grades']
R( 0 ) x R( 2 )
Rule #3 There is No Conflict between ['students', 'view', 'attendance'] and  ['students', 'view', 'cgpa']
R( 0 ) x R( 3 )
Rule #8 There is No Conflict between ['students', 'view', 'attendance'] and  ['facultymembers', 'add', 'grades']
R( 0 ) x R( 4 )
Rule #8 There is No Conflict between ['students', 'view', 'attendance'] and  ['facultymembers', 'edit', 'grades']
R( 0 ) x R( 5 )
Rule #7 There is No Conflict between ['students', 'view', 'attendance'] and  ['facultymembers', 'mark', 'attendance']
R( 0 ) x R( 6 )
Rule #8 There is No Conflict between ['students', 'view', 'attendance'] and  ['facultymembers', 'submit', 'results']
R( 0 ) x R( 7 )
Rule #8 There is No Conflict between ['students', 'view', 'attendance'] and  ['system', 'estimate', 'cgpa']
R( 0 ) x R( 8 )
Rule #8 There is No Conflict between ['students', 'view', 'attendance'] and  ['users', 'access', 'system']
R( 0 ) x R( 9 )
Rule #8 There is No Conflict between ['students', 'view', 'attendance'] and  ['facultymembers', 'change', 'passwords']
```

Figure 5.12: Automated Conflict Detection for R0

```
Requirement# 1
R( 1 ) x R( 0 )
Rule #3 There is No Conflict between ['students', 'view', 'grades'] and  ['students', 'view', 'attendance']
R( 1 ) x R( 1 )
Rule #1 There is No Conflict between ['students', 'view', 'grades'] and  ['students', 'view', 'grades']
R( 1 ) x R( 2 )
Rule #3 There is No Conflict between ['students', 'view', 'grades'] and  ['students', 'view', 'cgpa']
R( 1 ) x R( 3 )
Rule #7 There is No Conflict between ['students', 'view', 'grades'] and  ['facultymembers', 'add', 'grades']
R( 1 ) x R( 4 )
Rule #7 There is No Conflict between ['students', 'view', 'grades'] and  ['facultymembers', 'edit', 'grades']
R( 1 ) x R( 5 )
Rule #8 There is No Conflict between ['students', 'view', 'grades'] and  ['facultymembers', 'mark', 'attendance']
R( 1 ) x R( 6 )
Rule #8 There is No Conflict between ['students', 'view', 'grades'] and  ['facultymembers', 'submit', 'results']
R( 1 ) x R( 7 )
Rule #8 There is No Conflict between ['students', 'view', 'grades'] and  ['system', 'estimate', 'cgpa']
R( 1 ) x R( 8 )
Rule #8 There is No Conflict between ['students', 'view', 'grades'] and  ['users', 'access', 'system']
R( 1 ) x R( 9 )
Rule #8 There is No Conflict between ['students', 'view', 'grades'] and  ['facultymembers', 'change', 'passwords']
```

Figure 5.13: Automated Conflict Detection for R1

```
Requirement# 2
R( 2 ) x R( 0 )
Rule #3 There is No Conflict between ['students', 'view', 'cgpa'] and  ['students', 'view', 'attendance']
R( 2 ) x R( 1 )
Rule #3 There is No Conflict between ['students', 'view', 'cgpa'] and  ['students', 'view', 'grades']
R( 2 ) x R( 2 )
Rule #1 There is No Conflict between ['students', 'view', 'cgpa'] and  ['students', 'view', 'cgpa']
R( 2 ) x R( 3 )
Rule #8 There is No Conflict between ['students', 'view', 'cgpa'] and  ['facultymembers', 'add', 'grades']
R( 2 ) x R( 4 )
Rule #8 There is No Conflict between ['students', 'view', 'cgpa'] and  ['facultymembers', 'edit', 'grades']
R( 2 ) x R( 5 )
Rule #8 There is No Conflict between ['students', 'view', 'cgpa'] and  ['facultymembers', 'mark', 'attendance']
R( 2 ) x R( 6 )
Rule #8 There is No Conflict between ['students', 'view', 'cgpa'] and  ['facultymembers', 'submit', 'results']
R( 2 ) x R( 7 )
Rule #7 There is No Conflict between ['students', 'view', 'cgpa'] and  ['system', 'estimate', 'cgpa']
R( 2 ) x R( 8 )
Rule #8 There is No Conflict between ['students', 'view', 'cgpa'] and  ['users', 'access', 'system']
R( 2 ) x R( 9 )
Rule #8 There is No Conflict between ['students', 'view', 'cgpa'] and  ['facultymembers', 'change', 'passwords']
```

Figure 5.14: Automated Conflict Detection for R2

```
Requirement# 3
R( 3 ) x R( 0 )
Rule #8 There is No Conflict between ['facultymembers', 'add', 'grades'] and  ['students', 'view', 'attendance']
R( 3 ) x R( 1 )
Rule #7 There is No Conflict between ['facultymembers', 'add', 'grades'] and  ['students', 'view', 'grades']
R( 3 ) x R( 2 )
Rule #8 There is No Conflict between ['facultymembers', 'add', 'grades'] and  ['students', 'view', 'cgpa']
R( 3 ) x R( 3 )
Rule #1 There is No Conflict between ['facultymembers', 'add', 'grades'] and  ['facultymembers', 'add', 'grades']
R( 3 ) x R( 4 )
Rule #2 There is Conflict between ['facultymembers', 'add', 'grades'] and  ['facultymembers', 'edit', 'grades']
R( 3 ) x R( 5 )
Rule #4 There is No Conflict between ['facultymembers', 'add', 'grades'] and  ['facultymembers', 'mark', 'attendance']
R( 3 ) x R( 6 )
Rule #4 There is No Conflict between ['facultymembers', 'add', 'grades'] and  ['facultymembers', 'submit', 'results']
R( 3 ) x R( 7 )
Rule #8 There is No Conflict between ['facultymembers', 'add', 'grades'] and  ['system', 'estimate', 'cgpa']
R( 3 ) x R( 8 )
Rule #8 There is No Conflict between ['facultymembers', 'add', 'grades'] and  ['users', 'access', 'system']
R( 3 ) x R( 9 )
Rule #4 There is No Conflict between ['facultymembers', 'add', 'grades'] and  ['facultymembers', 'change', 'passwords']
```

Figure 5.16: Automated Conflict Detection for R3

```
Requirement# 4
R( 4 ) x R( 0 )
Rule #8 There is No Conflict between ['facultymembers', 'edit', 'grades'] and  ['students', 'view', 'attendance']
R( 4 ) x R( 1 )
Rule #7 There is No Conflict between ['facultymembers', 'edit', 'grades'] and  ['students', 'view', 'grades']
R( 4 ) x R( 2 )
Rule #8 There is No Conflict between ['facultymembers', 'edit', 'grades'] and  ['students', 'view', 'cgpa']
R( 4 ) x R( 3 )
Rule #2 There is Conflict between ['facultymembers', 'edit', 'grades'] and  ['facultymembers', 'add', 'grades']
R( 4 ) x R( 4 )
Rule #1 There is No Conflict between ['facultymembers', 'edit', 'grades'] and  ['facultymembers', 'edit', 'grades']
R( 4 ) x R( 5 )
Rule #4 There is No Conflict between ['facultymembers', 'edit', 'grades'] and  ['facultymembers', 'mark', 'attendance']
R( 4 ) x R( 6 )
Rule #4 There is No Conflict between ['facultymembers', 'edit', 'grades'] and  ['facultymembers', 'submit', 'results']
R( 4 ) x R( 7 )
Rule #8 There is No Conflict between ['facultymembers', 'edit', 'grades'] and  ['system', 'estimate', 'cgpa']
R( 4 ) x R( 8 )
Rule #8 There is No Conflict between ['facultymembers', 'edit', 'grades'] and  ['users', 'access', 'system']
R( 4 ) x R( 9 )
Rule #4 There is No Conflict between ['facultymembers', 'edit', 'grades'] and  ['facultymembers', 'change', 'passwords']
```

Figure 5.15: Automated Conflict Detection for R4

```
Requirement# 5
R( 5 ) x R( 0 )
Rule #7 There is No Conflict between ['facultymembers', 'mark', 'attendance'] and  ['students', 'view', 'attendance']
R( 5 ) x R( 1 )
Rule #8 There is No Conflict between ['facultymembers', 'mark', 'attendance'] and  ['students', 'view', 'grades']
R( 5 ) x R( 2 )
Rule #8 There is No Conflict between ['facultymembers', 'mark', 'attendance'] and  ['students', 'view', 'cgpa']
R( 5 ) x R( 3 )
Rule #4 There is No Conflict between ['facultymembers', 'mark', 'attendance'] and  ['facultymembers', 'add', 'grades']
R( 5 ) x R( 4 )
Rule #4 There is No Conflict between ['facultymembers', 'mark', 'attendance'] and  ['facultymembers', 'edit', 'grades']
R( 5 ) x R( 5 )
Rule #1 There is No Conflict between ['facultymembers', 'mark', 'attendance'] and  ['facultymembers', 'mark', 'attendance']
R( 5 ) x R( 6 )
Rule #4 There is No Conflict between ['facultymembers', 'mark', 'attendance'] and  ['facultymembers', 'submit', 'results']
R( 5 ) x R( 7 )
Rule #8 There is No Conflict between ['facultymembers', 'mark', 'attendance'] and  ['system', 'estimate', 'cgpa']
R( 5 ) x R( 8 )
Rule #8 There is No Conflict between ['facultymembers', 'mark', 'attendance'] and  ['users', 'access', 'system']
R( 5 ) x R( 9 )
Rule #4 There is No Conflict between ['facultymembers', 'mark', 'attendance'] and  ['facultymembers', 'change', 'passwords']
```

Figure 5.17: Automated Conflict Detection for R5

```
Requirement# 6
R( 6 ) x R( 0 )
Rule #8 There is No Conflict between ['facultymembers', 'submit', 'results'] and  ['students', 'view', 'attendance']
R( 6 ) x R( 1 )
Rule #8 There is No Conflict between ['facultymembers', 'submit', 'results'] and  ['students', 'view', 'grades']
R( 6 ) x R( 2 )
Rule #8 There is No Conflict between ['facultymembers', 'submit', 'results'] and  ['students', 'view', 'cgpa']
R( 6 ) x R( 3 )
Rule #4 There is No Conflict between ['facultymembers', 'submit', 'results'] and  ['facultymembers', 'add', 'grades']
R( 6 ) x R( 4 )
Rule #4 There is No Conflict between ['facultymembers', 'submit', 'results'] and  ['facultymembers', 'edit', 'grades']
R( 6 ) x R( 5 )
Rule #4 There is No Conflict between ['facultymembers', 'submit', 'results'] and  ['facultymembers', 'mark', 'attendance']
R( 6 ) x R( 6 )
Rule #1 There is No Conflict between ['facultymembers', 'submit', 'results'] and  ['facultymembers', 'submit', 'results']
R( 6 ) x R( 7 )
Rule #8 There is No Conflict between ['facultymembers', 'submit', 'results'] and  ['system', 'estimate', 'cgpa']
R( 6 ) x R( 8 )
Rule #8 There is No Conflict between ['facultymembers', 'submit', 'results'] and  ['users', 'access', 'system']
R( 6 ) x R( 9 )
Rule #4 There is No Conflict between ['facultymembers', 'submit', 'results'] and  ['facultymembers', 'change', 'passwords']
```

Figure 5.18: Automated Conflict Detection for R6

```
Requirement# 7
R( 7 ) x R( 0 )
Rule #8 There is No Conflict between ['system', 'estimate', 'cgpa'] and  ['students', 'view', 'attendance']
R( 7 ) x R( 1 )
Rule #8 There is No Conflict between ['system', 'estimate', 'cgpa'] and  ['students', 'view', 'grades']
R( 7 ) x R( 2 )
Rule #7 There is No Conflict between ['system', 'estimate', 'cgpa'] and  ['students', 'view', 'cgpa']
R( 7 ) x R( 3 )
Rule #8 There is No Conflict between ['system', 'estimate', 'cgpa'] and  ['facultymembers', 'add', 'grades']
R( 7 ) x R( 4 )
Rule #8 There is No Conflict between ['system', 'estimate', 'cgpa'] and  ['facultymembers', 'edit', 'grades']
R( 7 ) x R( 5 )
Rule #8 There is No Conflict between ['system', 'estimate', 'cgpa'] and  ['facultymembers', 'mark', 'attendance']
R( 7 ) x R( 6 )
Rule #8 There is No Conflict between ['system', 'estimate', 'cgpa'] and  ['facultymembers', 'submit', 'results']
R( 7 ) x R( 7 )
Rule #1 There is No Conflict between ['system', 'estimate', 'cgpa'] and  ['system', 'estimate', 'cgpa']
R( 7 ) x R( 8 )
Rule #8 There is No Conflict between ['system', 'estimate', 'cgpa'] and  ['users', 'access', 'system']
R( 7 ) x R( 9 )
Rule #8 There is No Conflict between ['system', 'estimate', 'cgpa'] and  ['facultymembers', 'change', 'passwords']
```

Figure 5.19: Automated Conflict Detection for R7

```
Requirement# 8
R( 8 ) x R( 0 )
Rule #8 There is No Conflict between ['users', 'access', 'system'] and  ['students', 'view', 'attendance']
R( 8 ) x R( 1 )
Rule #8 There is No Conflict between ['users', 'access', 'system'] and  ['students', 'view', 'grades']
R( 8 ) x R( 2 )
Rule #8 There is No Conflict between ['users', 'access', 'system'] and  ['students', 'view', 'cgpa']
R( 8 ) x R( 3 )
Rule #8 There is No Conflict between ['users', 'access', 'system'] and  ['facultymembers', 'add', 'grades']
R( 8 ) x R( 4 )
Rule #8 There is No Conflict between ['users', 'access', 'system'] and  ['facultymembers', 'edit', 'grades']
R( 8 ) x R( 5 )
Rule #8 There is No Conflict between ['users', 'access', 'system'] and  ['facultymembers', 'mark', 'attendance']
R( 8 ) x R( 6 )
Rule #8 There is No Conflict between ['users', 'access', 'system'] and  ['facultymembers', 'submit', 'results']
R( 8 ) x R( 7 )
Rule #8 There is No Conflict between ['users', 'access', 'system'] and  ['system', 'estimate', 'cgpa']
R( 8 ) x R( 8 )
Rule #1 There is No Conflict between ['users', 'access', 'system'] and  ['users', 'access', 'system']
R( 8 ) x R( 9 )
Rule #8 There is No Conflict between ['users', 'access', 'system'] and  ['facultymembers', 'change', 'passwords']
```

Figure 5.20: Automated Conflict Detection for R8

```
Requirement# 9
R( 9 ) x R( 0 )
Rule #8 There is No Conflict between ['facultymembers', 'change', 'passwords'] and  ['students', 'view', 'attendance']
R( 9 ) x R( 1 )
Rule #8 There is No Conflict between ['facultymembers', 'change', 'passwords'] and  ['students', 'view', 'grades']
R( 9 ) x R( 2 )
Rule #8 There is No Conflict between ['facultymembers', 'change', 'passwords'] and  ['students', 'view', 'cgpa']
R( 9 ) x R( 3 )
Rule #4 There is No Conflict between ['facultymembers', 'change', 'passwords'] and  ['facultymembers', 'add', 'grades']
R( 9 ) x R( 4 )
Rule #4 There is No Conflict between ['facultymembers', 'change', 'passwords'] and  ['facultymembers', 'edit', 'grades']
R( 9 ) x R( 5 )
Rule #4 There is No Conflict between ['facultymembers', 'change', 'passwords'] and  ['facultymembers', 'mark', 'attendance']
R( 9 ) x R( 6 )
Rule #4 There is No Conflict between ['facultymembers', 'change', 'passwords'] and  ['facultymembers', 'submit', 'results']
R( 9 ) x R( 7 )
Rule #8 There is No Conflict between ['facultymembers', 'change', 'passwords'] and  ['system', 'estimate', 'cgpa']
R( 9 ) x R( 8 )
Rule #8 There is No Conflict between ['facultymembers', 'change', 'passwords'] and  ['users', 'access', 'system']
R( 9 ) x R( 9 )
Rule #1 There is No Conflict between ['facultymembers', 'change', 'passwords'] and  ['facultymembers', 'change', 'passwords']
```

Figure 5.21: Automated Conflict Detection for R9

These are automated results, according to which there are 2 conflicts in total out of ten requirements as our test case, the detection based on comparisons is shown in ***Table 5.1***.

Table 5.1: Automated Testing Results for CS1

	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9
R0	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R1	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R2	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R3	NC	NC	NC	NC	C	NC	NC	NC	NC	NC
R4	NC	NC	NC	C	NC	NC	NC	NC	NC	NC
R5	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R6	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R7	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R8	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R9	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC

5.1.3.2. Manual Results

For verification of our proposed automated model for conflict detection, we also worked on manual testing as a way of comparison between the limited parameters of our test cases. ***Table 5.2*** presents the results of manual testing done on the initial requirements chosen as test case for our model.

Table 5.2: Manual Testing Results for CS1

	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9
R0	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R1	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R2	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R3	NC	NC	NC	NC	C	NC	NC	NC	NC	NC
R4	NC	NC	NC	C	NC	NC	NC	NC	NC	NC
R5	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R6	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R7	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R8	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R9	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC

As this is our training set of requirements, the results for automated and manual testing is exactly the same and thus, gives 100% accuracy.

5.1.3.3. Verification and Comparison of Results

When the automated and manual test results are compared, there is a 100% correct match in comparison. Overall, out of a test case of ten requirements, performing RxR comparison against each requirement against every other requirement as shown and highlighted in ***Table 5.2*** we have

found two cases of conflicts. Further details regarding these test results is discussed in **Chapter 6** and **Chapter 7.**

5.2. PMRB Software Development QA Guidance Document

We have used a the PMRB Software Development QA Guidance document develop by the US Environmental Protection Agency [31] as our case study to verify our research study and proposed approach.. In the implementation phase, the pre-processing and application of contextual rules is the same as Section 5.1.2.1, Section 5.1.2.2, and Section 5.1.2.3. The resultant comparison varies on the basis of extracted words and the comparison between requirements in the RxR table.

For this case study, Section 5.2.1 presents the requirements specification of the PMRB Software Development QA Guidance document's sub-part of the "Software Delivery Terms and Conditions" as our second set of requirements test case. Section 5.2.2 presents the set of extracted words after NLP pre-processing and before the application of results. Section 5.2.3 presents the testing methods, results, and the verification of results by comparison. Furthermore, Section 5.2.3.1 presents the automated results performed on the new test cases, and Section 5.2.3.2 presents the manually testing and results on the requirements. Finally, Section 5.2.3.3 gives a comparison analysis of both testing methods to verify the results in our approach.

5.2.1. Requirement Specification

The case study chosen is from the US Environmental Protection Agency's Office of Research and Development (ORD), and furthermore, the specific set of requirements is from the specific part "Software Delivery and Terms" part of the QA document. The requirements chosen correlate in terms of structure to our training requirements data set and are 12 in number with sub-requirements that are taken into consideration for the testing and verification. ***Figure 5.22*** shows the requirements specification from our selected case study.

1. The technical data contained in the delivered software code, and the software code delivered to PMRB shall not be copied, translated, reproduced, distributed or utilized by the contractor in any way outside of PMRB research project activities without EPA officer/work assignment manager's consent displayed in words and no discontent expressed in paper.
2. The contractor shall follow the technical direction of the EPA project officer/work assignment manager to ensure that all software products (that is, source code, executable code, and associated documentation, etc) comply with the provisions of this document.
3. Products delivered to PMRB achieve the objectives of their respective scientific research project(s) should be by the contractor to ensure for all products.
4. As a minimum, the contractor shall provide to the EPA project at the end of the period of performance of their project(s) officer/work assignment manager the following deliverables.
    a) a complete copy of the entire set of PMRB(uncompiled/uninterpreted) code developed under their project(s).
    b) a complete set of executable software code.
    c) c) a complete set of software documentation for all steps during the development process (that is, design manual, user manual, test/verification manual, metadata document, etc).
    d) a complete copy of any additional software and/or documentation resources (that is, code scripts, macros, makefiles, assembly language routines, 'cheat sheets', etc) used to develop the software source code and executable software code.
5. The contractor shall solicit technical direction from the EPA Project Officer/work assignment manager (if not specified in the contract, grant, cooperative agreement, interagency agreement, CRADA, etc) on the format(s) in which a), b), c), and d) are to be delivered (that is, DVD, CD-ROM, via FTP Site/Transfer [for large files], etc).
6. EPA shall retain unlimited rights to publish, translate, reproduce, deliver, distribute and dispose of the technical data, computer software code or computer firmware contained with the software code delivered by the contractor (within the constraints of any applicable third-party license agreements, previously approved, prior to software development by EPA).
7. The technical data contained in the delivered software code, and the software code delivered to PMRB shall not be copied, translated, reproduced, distributed or utilized by the contractor in any way outside of PMRB research project activities without EPA officer/work assignment manager's consent expressed in paper.
8. The technical data contained in the delivered software code, and the software code delivered to PMRB shall not be copied, translated, reproduced, distributed or utilized by the contractor in any way outside of PMRB research project activities without EPA officer/work assignment manager's expressed verbally consent.

Figure 5.22: Requirements Specification

5.2.2. NLP Pre-Processing

Following the application of NLP Pre-processing and list indexing as explained in Section 5.1.2.1 and Section 5.1.2.2, we get a set of extracted words as our processed test case as shown in ***Figure 5.23***.

```
Final Matrix
['discontent', 'expressed', 'paper']
['comply', 'provisions', 'document']
['contractor', 'ensure', 'products']
['manager', 'following', 'deliverables']
['code', 'developed', 'project']
['software', 'code']
['etc']
['software', 'code']
['files']
['software', 'development', 'epa']
['consent', 'expressed', 'paper']
['expressed', 'written', 'consent']
```

Figure 5.23: Processed Requirements List

Following the transformation of initial raw test cases into processed set of extracted words as requirements, we applied a set of contextual rules as shown in Section 5.1.2.3. However, as the case study is in its raw form with conflicts in terms of syntax, structure and logic to the requirements displayed in the specific document, our approach has produced varying results. The case study does not entirely adjust to our approach which has resulted in lacking automated results as shown in the following sections.

5.2.3. Conflict Detection Verification

For verification, we have performed two-step testing: automated testing as done using our proposed approach for conflict detection, and manual testing of the true conflicts in the set of requirements. Section 5.2.3.1 presents the automated testing, Section 5.2.3.2 presents the manual testing method, and Section 5.2.3.3 presents the verification and comparison of both testing methods on the test cases.

5.2.3.1. Automated Results

When the comparison is performed on the final set of processed requirements, we get a direct display of sentences presenting whether there is a conflict between RxR. For the processed test cases where not enough words are classified in labels for our processed list of requirements, the model does not pick them up and the comparison is shown as null, this is done to avoid Mis-identifying conflicts on the basis of their indexing in lists.

Following Figures shows the automated results that we have achieved for all eleven specific requirements.

```
Requirement# 0
R( 0 ) x R( 0 )
Rule #1 There is No Conflict between ['discontent', 'expressed', 'paper'] and  ['discontent', 'expressed', 'paper']
R( 0 ) x R( 1 )
Rule #8 There is No Conflict between ['discontent', 'expressed', 'paper'] and  ['comply', 'provisions', 'document']
R( 0 ) x R( 2 )
Rule #8 There is No Conflict between ['discontent', 'expressed', 'paper'] and  ['contractor', 'ensure', 'products']
R( 0 ) x R( 3 )
Rule #8 There is No Conflict between ['discontent', 'expressed', 'paper'] and  ['manager', 'following', 'deliverables']
R( 0 ) x R( 4 )
Rule #8 There is No Conflict between ['discontent', 'expressed', 'paper'] and  ['code', 'developed', 'project']
R( 0 ) x R( 5 )
R( 0 ) x R( 6 )
R( 0 ) x R( 7 )
R( 0 ) x R( 8 )
R( 0 ) x R( 9 )
Rule #8 There is No Conflict between ['discontent', 'expressed', 'paper'] and  ['software', 'development', 'epa']
R( 0 ) x R( 10 )
Rule #5 There is Conflict between ['discontent', 'expressed', 'paper'] and  ['consent', 'expressed', 'paper']
R( 0 ) x R( 11 )
Rule #8 There is No Conflict between ['discontent', 'expressed', 'paper'] and  ['expressed', 'written', 'consent']
```

Figure 5.24: Automated Conflict Detection for R0

```
Requirement# 1
R( 1 ) x R( 0 )
Rule #8 There is No Conflict between ['comply', 'provisions', 'document'] and  ['discontent', 'expressed', 'paper']
R( 1 ) x R( 1 )
Rule #1 There is No Conflict between ['comply', 'provisions', 'document'] and  ['comply', 'provisions', 'document']
R( 1 ) x R( 2 )
Rule #8 There is No Conflict between ['comply', 'provisions', 'document'] and  ['contractor', 'ensure', 'products']
R( 1 ) x R( 3 )
Rule #8 There is No Conflict between ['comply', 'provisions', 'document'] and  ['manager', 'following', 'deliverables']
R( 1 ) x R( 4 )
Rule #8 There is No Conflict between ['comply', 'provisions', 'document'] and  ['code', 'developed', 'project']
R( 1 ) x R( 5 )
R( 1 ) x R( 6 )
R( 1 ) x R( 7 )
R( 1 ) x R( 8 )
R( 1 ) x R( 9 )
Rule #8 There is No Conflict between ['comply', 'provisions', 'document'] and  ['software', 'development', 'epa']
R( 1 ) x R( 10 )
Rule #8 There is No Conflict between ['comply', 'provisions', 'document'] and  ['consent', 'expressed', 'paper']
R( 1 ) x R( 11 )
Rule #8 There is No Conflict between ['comply', 'provisions', 'document'] and  ['expressed', 'written', 'consent']
```

Figure 5.25: Automated Conflict Detection for R1

```
Requirement# 2
R( 2 ) x R( 0 )
Rule #8 There is No Conflict between ['contractor', 'ensure', 'products'] and  ['discontent', 'expressed', 'paper']
R( 2 ) x R( 1 )
Rule #8 There is No Conflict between ['contractor', 'ensure', 'products'] and  ['comply', 'provisions', 'document']
R( 2 ) x R( 2 )
Rule #1 There is No Conflict between ['contractor', 'ensure', 'products'] and  ['contractor', 'ensure', 'products']
R( 2 ) x R( 3 )
Rule #8 There is No Conflict between ['contractor', 'ensure', 'products'] and  ['manager', 'following', 'deliverables']
R( 2 ) x R( 4 )
Rule #8 There is No Conflict between ['contractor', 'ensure', 'products'] and  ['code', 'developed', 'project']
R( 2 ) x R( 5 )
R( 2 ) x R( 6 )
R( 2 ) x R( 7 )
R( 2 ) x R( 8 )
R( 2 ) x R( 9 )
Rule #8 There is No Conflict between ['contractor', 'ensure', 'products'] and  ['software', 'development', 'epa']
R( 2 ) x R( 10 )
Rule #8 There is No Conflict between ['contractor', 'ensure', 'products'] and  ['consent', 'expressed', 'paper']
R( 2 ) x R( 11 )
Rule #8 There is No Conflict between ['contractor', 'ensure', 'products'] and  ['expressed', 'written', 'consent']
```

Figure 5.26: Automated Conflict Detection for R2

```
Requirement# 3
R( 3 ) x R( 0 )
Rule #8 There is No Conflict between ['manager', 'following', 'deliverables'] and  ['discontent', 'expressed', 'paper']
R( 3 ) x R( 1 )
Rule #8 There is No Conflict between ['manager', 'following', 'deliverables'] and  ['comply', 'provisions', 'document']
R( 3 ) x R( 2 )
Rule #8 There is No Conflict between ['manager', 'following', 'deliverables'] and  ['contractor', 'ensure', 'products']
R( 3 ) x R( 3 )
Rule #1 There is No Conflict between ['manager', 'following', 'deliverables'] and  ['manager', 'following', 'deliverables']
R( 3 ) x R( 4 )
Rule #8 There is No Conflict between ['manager', 'following', 'deliverables'] and  ['code', 'developed', 'project']
R( 3 ) x R( 5 )
R( 3 ) x R( 6 )
R( 3 ) x R( 7 )
R( 3 ) x R( 8 )
R( 3 ) x R( 9 )
Rule #8 There is No Conflict between ['manager', 'following', 'deliverables'] and  ['software', 'development', 'epa']
R( 3 ) x R( 10 )
Rule #8 There is No Conflict between ['manager', 'following', 'deliverables'] and  ['consent', 'expressed', 'paper']
R( 3 ) x R( 11 )
Rule #8 There is No Conflict between ['manager', 'following', 'deliverables'] and  ['expressed', 'written', 'consent']
```

Figure 5.27: Automated Conflict Detection for R3

```
Requirement# 4
R( 4 ) x R( 0 )
Rule #8 There is No Conflict between ['code', 'developed', 'project'] and  ['discontent', 'expressed', 'paper']
R( 4 ) x R( 1 )
Rule #8 There is No Conflict between ['code', 'developed', 'project'] and  ['comply', 'provisions', 'document']
R( 4 ) x R( 2 )
Rule #8 There is No Conflict between ['code', 'developed', 'project'] and  ['contractor', 'ensure', 'products']
R( 4 ) x R( 3 )
Rule #8 There is No Conflict between ['code', 'developed', 'project'] and  ['manager', 'following', 'deliverables']
R( 4 ) x R( 4 )
Rule #1 There is No Conflict between ['code', 'developed', 'project'] and  ['code', 'developed', 'project']
R( 4 ) x R( 5 )
R( 4 ) x R( 6 )
R( 4 ) x R( 7 )
R( 4 ) x R( 8 )
R( 4 ) x R( 9 )
Rule #8 There is No Conflict between ['code', 'developed', 'project'] and  ['software', 'development', 'epa']
R( 4 ) x R( 10 )
Rule #8 There is No Conflict between ['code', 'developed', 'project'] and  ['consent', 'expressed', 'paper']
R( 4 ) x R( 11 )
Rule #8 There is No Conflict between ['code', 'developed', 'project'] and  ['expressed', 'written', 'consent']
```

Figure 5.28: Automated Conflict Detection for R4

```
Requirement# 5


Requirement# 6


Requirement# 7


Requirement# 8
```

Figure 5.29: Automated Conflict Detection for R5, R6, R7, and R8

As shown in ***Figure 5.29***, the comparison matrix for requirements that have not enough contextually extracted words is left to 0. We have set the baseline for extracted words to be three in order to satisfy the classification labelling of Subject, Action, and Object which is the pre-lude to our comparison matrix.

```
Requirement# 9
R( 9 ) x R( 0 )
Rule #8 There is No Conflict between ['software', 'development', 'epa'] and  ['discontent', 'expressed', 'paper']
R( 9 ) x R( 1 )
Rule #8 There is No Conflict between ['software', 'development', 'epa'] and  ['comply', 'provisions', 'document']
R( 9 ) x R( 2 )
Rule #8 There is No Conflict between ['software', 'development', 'epa'] and  ['contractor', 'ensure', 'products']
R( 9 ) x R( 3 )
Rule #8 There is No Conflict between ['software', 'development', 'epa'] and  ['manager', 'following', 'deliverables']
R( 9 ) x R( 4 )
Rule #8 There is No Conflict between ['software', 'development', 'epa'] and  ['code', 'developed', 'project']
R( 9 ) x R( 5 )
R( 9 ) x R( 6 )
R( 9 ) x R( 7 )
R( 9 ) x R( 8 )
R( 9 ) x R( 9 )
Rule #1 There is No Conflict between ['software', 'development', 'epa'] and  ['software', 'development', 'epa']
R( 9 ) x R( 10 )
Rule #8 There is No Conflict between ['software', 'development', 'epa'] and  ['consent', 'expressed', 'paper']
R( 9 ) x R( 11 )
Rule #8 There is No Conflict between ['software', 'development', 'epa'] and  ['expressed', 'written', 'consent']
```

Figure 5.30: Automated Conflict Detection for R9

```
Requirement# 10
R( 10 ) x R( 0 )
Rule #5 There is Conflict between ['consent', 'expressed', 'paper'] and  ['discontent', 'expressed', 'paper']
R( 10 ) x R( 1 )
Rule #8 There is No Conflict between ['consent', 'expressed', 'paper'] and  ['comply', 'provisions', 'document']
R( 10 ) x R( 2 )
Rule #8 There is No Conflict between ['consent', 'expressed', 'paper'] and  ['contractor', 'ensure', 'products']
R( 10 ) x R( 3 )
Rule #8 There is No Conflict between ['consent', 'expressed', 'paper'] and  ['manager', 'following', 'deliverables']
R( 10 ) x R( 4 )
Rule #8 There is No Conflict between ['consent', 'expressed', 'paper'] and  ['code', 'developed', 'project']
R( 10 ) x R( 5 )
R( 10 ) x R( 6 )
R( 10 ) x R( 7 )
R( 10 ) x R( 8 )
R( 10 ) x R( 9 )
Rule #8 There is No Conflict between ['consent', 'expressed', 'paper'] and  ['software', 'development', 'epa']
R( 10 ) x R( 10 )
Rule #1 There is No Conflict between ['consent', 'expressed', 'paper'] and  ['consent', 'expressed', 'paper']
R( 10 ) x R( 11 )
Rule #8 There is No Conflict between ['consent', 'expressed', 'paper'] and  ['expressed', 'written', 'consent']
```

Figure 5.31: Automated Conflict Detection for R10

```
Requirement# 11
R( 11 ) x R( 0 )
Rule #8 There is No Conflict between ['expressed', 'written', 'consent'] and  ['discontent', 'expressed', 'paper']
R( 11 ) x R( 1 )
Rule #8 There is No Conflict between ['expressed', 'written', 'consent'] and  ['comply', 'provisions', 'document']
R( 11 ) x R( 2 )
Rule #8 There is No Conflict between ['expressed', 'written', 'consent'] and  ['contractor', 'ensure', 'products']
R( 11 ) x R( 3 )
Rule #8 There is No Conflict between ['expressed', 'written', 'consent'] and  ['manager', 'following', 'deliverables']
R( 11 ) x R( 4 )
Rule #8 There is No Conflict between ['expressed', 'written', 'consent'] and  ['code', 'developed', 'project']
R( 11 ) x R( 5 )
R( 11 ) x R( 6 )
R( 11 ) x R( 7 )
R( 11 ) x R( 8 )
R( 11 ) x R( 9 )
Rule #8 There is No Conflict between ['expressed', 'written', 'consent'] and  ['software', 'development', 'epa']
R( 11 ) x R( 10 )
Rule #8 There is No Conflict between ['expressed', 'written', 'consent'] and  ['consent', 'expressed', 'paper']
R( 11 ) x R( 11 )
Rule #1 There is No Conflict between ['expressed', 'written', 'consent'] and  ['expressed', 'written', 'consent']
```

Figure 5.32: Automated Conflict Detection for R11

For ***Figure 5.30***, ***Figure 5.31***, and ***Figure 5.32***, the contradiction comparison follows, as shown above. Detected Conflicts found through automated testing are shown in **Table 5.3: Automated Testing Results for CS2.**

Table 5.3: Automated Testing Results for CS2

	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11
R0	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	C	NC
R1	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R2	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R3	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R4	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R5	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R6	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R7	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R8	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC

R9	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R10	C	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R11	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC

5.2.3.2. Manual Results

For verification of our proposed automated model for conflict detection, we also worked on manual testing as a way of comparison between the limited parameters of our test cases. ***Table 5.4*** presents the results of manual testing done on the case study chosen.

Table 5.4: Manual Testing Results for CS2

	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11
R0	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	C	C
R1	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R2	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R3	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R4	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R5	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R6	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R7	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R8	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R9	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R10	C	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	C
R11	C	NC	NC	NC	NC	NC	NC	NC	NC	NC	C	NC

From the manual testing, we have found 6 conflicts based on RxR matrix's 12x12 comparisons which equals to a total of 144 comparisons.

5.2.3.3. Verification and Comparison of Results

In Section 5.2.3.2, the model displayed two correct conflict detections as shown in ***Table 5.4***. After performing a comparison on the two testing methods, we get 2 correctly detected conflicts from our automated approach, out of 6 known conflicts as shown in ***Table 5.5***.

Table 5.5: Comparison of Automated Vs Manual Testing for CS2

	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11
R0	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	C	C
R1	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R2	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R3	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R4	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R5	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R6	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R7	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R8	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R9	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC
R10	C	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	C
R11	C	NC	NC	NC	NC	NC	NC	NC	NC	NC	C	NC

In ***Table 5.5***, green shows the correctly detected conflicts, and red shows the Mis-identified conflicts in automated as compared to the manual testing method. This presents our results in terms of Accuracy, Recall, and Precision [32] as shown in ***Figure 5.33*** as follows:

	Predicted class		
Actual Class		Class = Yes	Class = No
	Class = Yes	True Positive	False Negative
	Class = No	False Positive	True Negative

Figure 5.33: Accuracy, Recall, and Precision Score Interpretation for Results

- **Accuracy:**

  Accuracy = (TP + TN) / (TP + TN + FP + FN)

  Accuracy = (2 + 138) / (2 + 138 + 0 + 4)

  Accuracy = 0.97

  Based on ***Figure 5.33***, the accuracy of our approach based on the given metrics is 0.97 or 97%.

- **Recall**

  Recall = (TP) / (TP + FN)

  Recall = (2) / (2 + 4)

  Recall = 0.33

  Based on ***Figure 5.33***, the recall of our approach based on the given metrics is 0.33 or 33%.

- **Precision**

  Precision = (TP) / (TP + FP)

  Precision = (2) / (2 + 0)

  Precision = 1

  Based on ***Figure 5.33***, the precision of our approach is 1, or 100%. High precision relates to low false positive rate [32], which in our approach comes to 100%. This is mainly because our method is bound by contextual rules and cannot Mis-identify non-conflicts.

Chapter 6

Discussion and Limitation

Chapter 6: Discussion and Limitation

This chapter presents a detailed discussion and the limitations encountered in this research work. Discussions are shown in Section 6.1 and limitations to the research are presented in Section 6.2.

6.1. Discussion

From this research, it has been analyzed that there is a limited approach to automating conflict detection in software requirements during the Requirements Analysis step in Requirements Engineering phase in the SDLC. Most of the work has been proposed with varying data sets and is done manually. Our proposed approach is a first step towards automating conflict detection in software requirements using NLP techniques and contextual rules.

The approach we have proposed examines processed test cases using our classification labels and contextual rules which are based on a result of a technique of NLP, namely – Chunking. The use of NLP was encouraged because of the NLP toolkits available and the flexibility of NLP techniques. This motivated us to study NLP further and include it in our data extraction and synthesis section of research as well. Motivation behind this work is to provide an automated approach to conflict detection by extracting phrases from a sentence that can assist to conflict, inconsistency or doubt. The test cases are thoroughly processed and the flow of the algorithm is transparent so that their movement can be tracked as they are processed. The generated test cases follow a pattern of words so that comparison can be made without any inconsistency.

Our approach is a proposed algorithm and the tool isn't developed yet. But it is a step forward in our research case study of conflict detection.

Our selected requirements for test case are medium complexity sentences in English. The requirements were ten in number with varying degrees of length and structure. The purpose was to determine the exact and concise sequence of phrases that could be extracted for comparison analysis later on.

6.2. Limitations

As we have taken the first step towards automating conflict detection in software requirements using NLP techniques and contextual rules based on NLP techniques, specifically, extracted set of sequenced words from a requirement, there are a few limitation to our work. NLP techniques are quite flexible but its limitations increase with increase in the complexity of sentences. This in turn increases the pre-processing of test cases which becomes a project of its own. This is our limitation in our approach.

Chapter 7

Future Work

Chapter 7: Conclusion and Future Work

The proposed research gives a way in the direction of detailing requirements efficiently during the SDLC of a software project. Automated detection of conflicts in the early stages of Requirements Engineering will allow testers and developers to develop the right project without having to go back and forth to the early stages of SDLC. The proposed research makes use of the existing techniques and toolkit of Natural Language Processing.

Future work of our research includes improving and extending this approach to include a variety of requirements in terms of length and complexity. The applicability of turning this approach into an automated tool is another possibility in our future work set.

REFERENCES

1. Urbieta, M., et al. *Detecting Conflicts and Inconsistencies in Web Application Requirements*. 2012. Berlin, Heidelberg: Springer Berlin Heidelberg.
2. https://distilradar.com/2018/09/24/sentiment-analysis-using-lexalytics/.
3. Kitchenham, B.A. *Systematic reviews*. in *10th International Symposium on Software Metrics, 2004. Proceedings.* 2004.
4. Malhotra, R., et al. *Analyzing and evaluating security features in software requirements*. in *2016 International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH)*. 2016.
5. Nikora, A.P. and G. Balcom. *Automated Identification of LTL Patterns in Natural Language Requirements*. in *2009 20th International Symposium on Software Reliability Engineering*. 2009.
6. Ott, D. *Defects in natural language requirement specifications at Mercedes-Benz: An investigation using a combination of legacy data and expert opinion*. in *2012 20th IEEE International Requirements Engineering Conference (RE)*. 2012.
7. Khtira, A., A. Benlarabi, and B.E. Asri. *Detecting feature duplication in natural language specifications when evolving software product lines*. in *2015 International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*. 2015.
8. Nguyen, T.H., J. Grundy, and M. Almorsy. *GUITAR: An ontology-based automated requirements analysis tool*. in *2014 IEEE 22nd International Requirements Engineering Conference (RE)*. 2014.
9. Mezghani, M., J. Kang, and F. Sèdes. *Industrial Requirements Classification for Redundancy and Inconsistency Detection in SEMIOS*. in *2018 IEEE 26th International Requirements Engineering Conference (RE)*. 2018.
10. Herrera, J., et al. *The Revealing Crosscutting Concerns in Textual Requirements Documents: An Exploratory Study with Industry Systems*. in *2012 26th Brazilian Symposium on Software Engineering*. 2012.
11. Sandhu, G. and S. Sikka. *State-of-art practices to detect inconsistencies and ambiguities from software requirements*. in *International Conference on Computing, Communication & Automation*. 2015.
12. Arunthavanathan, A., et al. *Support for traceability management of software artefacts using Natural Language Processing*. in *2016 Moratuwa Engineering Research Conference (MERCon)*. 2016.
13. Zambrano, A., J. Fabry, and S. Gordillo, *Expressing aspectual interactions in requirements engineering: Experiences, problems and solutions.* Science of Computer Programming, 2012. **78**(1): p. 65-92.
14. Binkley, D., et al., *Increasing diversity: Natural language measures for software fault prediction.* Journal of Systems and Software, 2009. **82**(11): p. 1793-1803.
15. Bateman, J.A., et al., *A linguistic ontology of space for natural language processing.* Artificial Intelligence, 2010. **174**(14): p. 1027-1071.
16. Chentouf, Z., *Managing OAM&P requirement conflicts.* Journal of King Saud University - Computer and Information Sciences, 2014. **26**(3): p. 296-307.
17. Dragos, V., *Detection of contradictions by relation matching and uncertainty assessment.* Procedia Computer Science, 2017. **112**: p. 71-80.
18. Matsumoto, Y., S. Shirai, and A. Ohnishi, *A Method for Verifying Non-Functional Requirements.* Procedia Computer Science, 2017. **112**: p. 157-166.
19. Phalnikar, R. and D. Jinwala, *Analysis of Conflicting User Requirements in Web Applications Using Graph Transformation.* SIGSOFT Softw. Eng. Notes, 2015. **40**(2): p. 1-7.

20. Sousa, T.C.d., et al., *Automatic analysis of requirements consistency with the B method.* SIGSOFT Softw. Eng. Notes, 2010. **35**(2): p. 1-4.
21. Perera, S., et al., *Challenges in understanding clinical notes: why NLP engines fall short and where background knowledge can help*, in *Proceedings of the 2013 international workshop on Data management & analytics for healthcare*. 2013, ACM: San Francisco, California, USA. p. 21-26.
22. Ahsan, I., et al., *A comprehensive investigation of natural language processing techniques and tools to generate automated test cases*. 2017. p. 132:1-132:10.
23. Sharma, V.S., R.R. Ramnani, and S. Sengupta, *A framework for identifying and analyzing non-functional requirements from text*, in *Proceedings of the 4th International Workshop on Twin Peaks of Requirements and Architecture*. 2014, ACM: Hyderabad, India. p. 1-8.
24. Iren, D. and H.A. Reijers, *Leveraging business process improvement with natural language processing and organizational semantic knowledge*, in *Proceedings of the 2017 International Conference on Software and System Process*. 2017, ACM: Paris, France. p. 100-108.
25. Sateli, B., E. Angius, and R. Witte. *ReqWiki Approach for Collaborative Software Requirements Engineering with Integrated Text Analysis Support*. in *2013 IEEE 37th Annual Computer Software and Applications Conference*. 2013.
26. Sengupta, S., et al., *Verb-based Semantic Modelling and Analysis of Textual Requirements*, in *Proceedings of the 8th India Software Engineering Conference*. 2015, ACM: Bangalore, India. p. 30-39.
27. Pivatelli, J. and J.C.S.d.P. Leite, *The clash between requirements volatility and software contracts*, in *Proceedings of the 31st Brazilian Symposium on Software Engineering*. 2017, ACM: Fortaleza, CE, Brazil. p. 144-153.
28. Falessi, D., G. Cantone, and G. Canfora, *A comprehensive characterization of NLP techniques for identifying equivalent requirements*, in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. 2010, ACM: Bolzano-Bozen, Italy. p. 1-10.
29. Sharma, M.K., *A study of SDLC to develop well engineered software.* 2017, 2017. **8**(3): p. 4.
30. Kotonya, G. and I. Sommerville, *Requirements Engineering: Processes and Techniques*. 1998: Wiley Publishing. 294.
31. Hall, E.S., *"PMRB Software Development QA Guidance Document"*.
32. Joshi, R., *"Accuracy, Recall, and Precision Score Interpretation".*