An approach to Verify Compliance of Requirement Template Using Natural Language

Author

Ahmed Hasnat Safder

NUST201464547MCEME35414F

Supervisor

Dr. Usman Akram

DEPARTMENT OF COMPUTER ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY ISLAMABAD

April 2018

An approach to Verify Compliance of Requirement Template Using Natural Language

Author

Ahmed Hasnat Safder

NUST20146454MCEME35414F

A Thesis submitted for the fulfillment of requirements for degree of

MS Computer Software Engineering

Thesis Supervisor

Dr. Usman Akram

Thesis Supervisor's Signature: - ________________________________

DEPARTMENT OF COMPUTER ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY ISLAMABAD

April, 2018

Declaration

I certify that this research work titled “*An approach to Verify Compliance of Requirement Template Using Natural Language*” is my own work. 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

Ahmed Hasnat Safder

NUST20146454MCEME35414F

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 format given by university.

Signature of Student

Ahmed Hasnat Safder

NUST ID

Supervisor Signature

Dr. Usman Akram

Acknowledgments

I am extremely thankful to Allah Almighty for guidance in this work, without Allah’s help I would never have been able to complete my work. Indeed, none be worthy of praise but Allah Subhana-Watala.

I would also like to thank my parents whose enormous help and love has enabled me to reach this point. I would also like to thank them for being the biggest positive influence on my life.

I am extremely grateful to my siblings, who have always helped and encouraged to challenge myself against bigger goals and objectives.

I am especially thankful to **Dr. Usman Akram** whose enormous dedication and broad knowledge was one of the biggest factor for me during my research phase. He serving as my supervisor provided a huge deal of motivation for me in each step of my research thesis.

I am also thankful to friends and colleagues for their step by step support and encouragement throughout this process.

Dedicated to my siblings who have helped and supported me like one of their own child throughout my life. I would never have made this far in life without you love and guidance. I am truly indebted to you all.

Table of Contents

List of Figures

List of Tables

Abstract

Requirement Engineering is the process to elicit stakeholder requirements and developing them in to an agreed requirement document. Requirements should serve as the basis and guideline for all the software development lifecycle. Top Level system requirements are typically written in Natural Language (NL) by individuals who are not requirement experts. During Software Development, problems in these requirements can create unnecessary risks that can impact schedule and cost. Requirement Templates address these issues by providing structural rules that increase the precision of natural language requirements. When applying these templates, it is necessary to verify that these requirements are indeed documented according to the template. Manual inspections for this purpose is time consuming and usually take multiple cycles to complete. In this paper we develop an automated approach to conform requirements to two well-known templates in requirement engineering community. Further we attempt to categorize individual requirements by using guidelines of the two mentioned requirement templates.

Key Words: Natural Language Processing, Requirement Templates, Text Chunking.

Chapter 1: Introduction

Requirements engineering is the process of defining and documenting stakeholder requirements and needs. Requirements are specified in such a way that they can serve as the basis for all other system development activities. Finally they are developed into detailed and agreed requirement documents [1]. Natural Language (NL) is arguably the most common method to specify these requirement. NL is easier to understand by all group of stakeholders as it requires little to no training. Despite these advantages unconstrained use of NL can be unsuitable for requirements definition for a number of reasons [2].

Words or phrases in NL sentences can be ambiguous meaning they can refer to multiple meanings. Vagueness or lack of precision is another major issue with NL. Compound requirements can contain complex subclasses or several interrelated statement which may lead to increase in complexity. There may be missing requirements, particularly to handle unwanted behaviors. Duplication or repetition of requirements is another concern with NL sentences. Wordiness (use of unnecessary number of words) is also common, especially when requirement are written by individuals who are not NL experts. Testability is another problem, requirements are difficult to be proven true or false when the system is implemented.

There are also other problems with the requirements, such as conflicting requirements and lack of tractability links. However these problems are not just unique to NL requirement document.

There have been several proposed approaches to replace the Natural Language for requirements. Most of these alternate methodologies are notation based, such as Petri Nets [3] and Z specification [4]. Notation based approaches remove some of core issues with using Natural Language such as ambiguity and testability. However they present several drawbacks of their own.

Use of these non-textual approaches require a process of complex translation from source requirements which can include further errors. Furthermore they result in language barrier between developers and stakeholders. There is also training overhead which is associated with introduction of such notations.

There are a lot of books on requirement engineering. There are also numerous literature on techniques of writing requirements. These include two well-known research papers “Writing Good Requirements” [9, 10] that focus on best practices for well-formed requirements and attributes that requirement document must include.

There always has been a growing need to apply a set of structures for high level natural languages based stakeholder requirements. The goal of these sets is to reduce the complexity of requirements and simplify their understanding to all stakeholders group. Many approaches were proposed to present these set of rules including Event Condition English[11], Specified Technical English [12] and Attempto Controlled English [13]. The goal of these approaches was to standardize a requirements document in order to reduce problems associated with Natural Language.

Requirement templates were introduced to provide a syntax to write requirement statements. This procedure is much more effective than using analytical methods. Construction of requirements statements according to set of rules and syntax can results in avoiding mistakes from the very beginning to software development lifecycle. It also helps in usage of similar structure for each requirement. During development of system, requirements generally are translated and formalized into objected-Oriented models and processes. Using predefined syntax for requirements can also greatly assists in these translations which are otherwise difficult with using NL.

When applying requirement templates, it is important to verify the correctness of requirements that they are written in accordance to templates. This process if done manually can be time and resource consuming, especially in cases where requirements are constantly changed. To overcome this problem we propose an automated method to verify requirements to templates. Our solution will implement automated checkers for two well reputed requirement templates. Furthermore we will characterize each valid requirement to the type it represents.

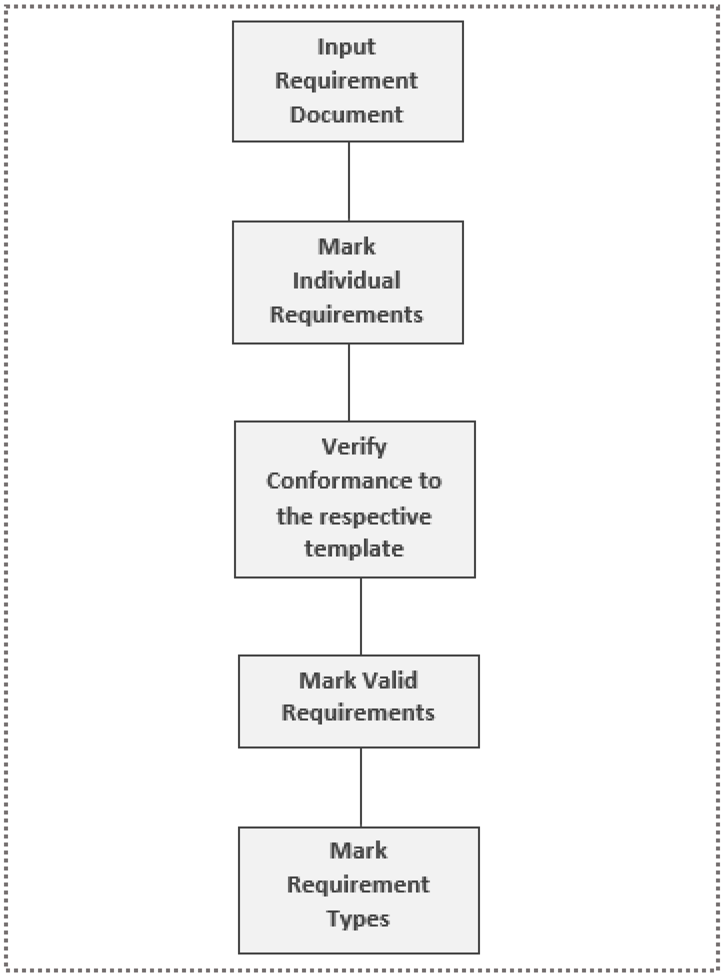

Input Requirement Document
Mark Individual Requirements
Verify Conformance to the respective template
Mark Valid Requirements
Mark Requirement Types

Figure 1. Overview of NL requirement conformance steps

1.1 Objectives

The domain of Natural Language Processing deals with an efficient and automated analysis of both in speech and written format of natural language. Natural language can be processed to several levels, ranging from analysis of each word to processing a document containing several sentences.

Since Natural Language is the most common way to describe and list a set of user requirements, it is especially useful since all stakeholders have a common understanding of natural language as compared to other approaches. Recent advancements in natural language processing in the domain of requirement engineering have enabled us to develop structured requirement documents.

The core objective is to develop a methodology that utilizes Natural Language Processing to enlist requirements that are not overly complex and ambiguous. The introduction of requirements templates in the field of requirement engineering have allowed us to write a well-structured requirement document.

In our study of literature review, we discovered that Natural Language Processing techniques can applied to conform a requirement to the specified template. So this research thesis is concerned with how these techniques can be applied to conform requirement statements to templates in most robust environments. Our objectives are to

- Analyzing natural language against other approaches to enlist requirements in a document.
- Understand benefits of natural language to facilitate all stakeholder concerns.
- To analyze requirement templates and their benefits in developing un-ambiguous requirements.
- Analyzing artificial intelligence based approach to conform requirements to templates using Natural Language Processing.
- Define a robust approach to validate requirement templates.
- Implementation of the defined approach on various case studies.
- To evaluate the accuracy of the approach against human manual conformance.

We emphasize that our main aim of our research is not completely eliminate human analysts, rather assist manual checking especially in most robust environments where requirement changes are expected.

1.2 Motivation

Requirement templates provide an efficient in elicitation of well-formed and unambiguous requirement documents. They are especially in development of safety critical systems where complex and ambiguous requirements can cause a great security risk. Several safety critical projects such as SAREMAN [14] and OPENCROSS [15] recommend the use of an accurate and precise requirement document.

There is an increasing support for requirement templates in commercial requirements [16]. Naturally there is a need for quality assurance activities surrounding these templates, one such example is the RQA tool [17] that provide functionality for verifying the use of these templates.

The effectiveness of such tools require building of a glossary terms. One important issue with dependence of glossary is wasted effort in glossary construction stage since all the terms in glossary are not needed in the actual conformance steps. To overcome this problem we identify and define the glossary terms only when requirements have reached an advance stage and are thus less likely to change [18]. One major drawback with such approach is that glossary is not ready for the start of template conformance stages. Also it is likely that glossaries may not be complete in the development stage [19] and as a result may not provide a full coverage for all the desired terms. Implication of these factors suggest that those tools for automated conformance of requirement templates that heavily rely on glossary may not be fully effective and reliable.

This thesis is motived by the need to develop an automated solution that provide features for conformance without the need of glossary. As a result in this thesis we make the following contributions.

- We propose an automated of requirement template conformance using Natural Language Processing (NLP), the distinguishing feature of our approach is eliminating the reliance on glossary terms.
- We will use text chunking for our goal, in which chunks (segments) of texts are identified. Our approach will not perform a detailed analysis on these chunks internal structure and relationships [20]. These chunks largely composed of verb phrases (VP) and noun phrases (NP) allow us to work on abstraction level over natural language.
- We further provide NLP parsing as text chunking is not sufficient to determine template conformance, especially in cases where requirements phrases contain complex noun phrases that include verb phrases.
- We evaluate our approach by using four cases studies. Two of these case studies have requirements according to RUPS template, while the other two are written using EARS template.

- The results from our case studies will show that our approach provides effective approach for requirement template conformance even for complex requirements and where glossary is undefined.
- We provide a tool for automated conformance, with features for both RUPPS and EARS, as well identifying requirement types according to the two mentioned requirement templates.

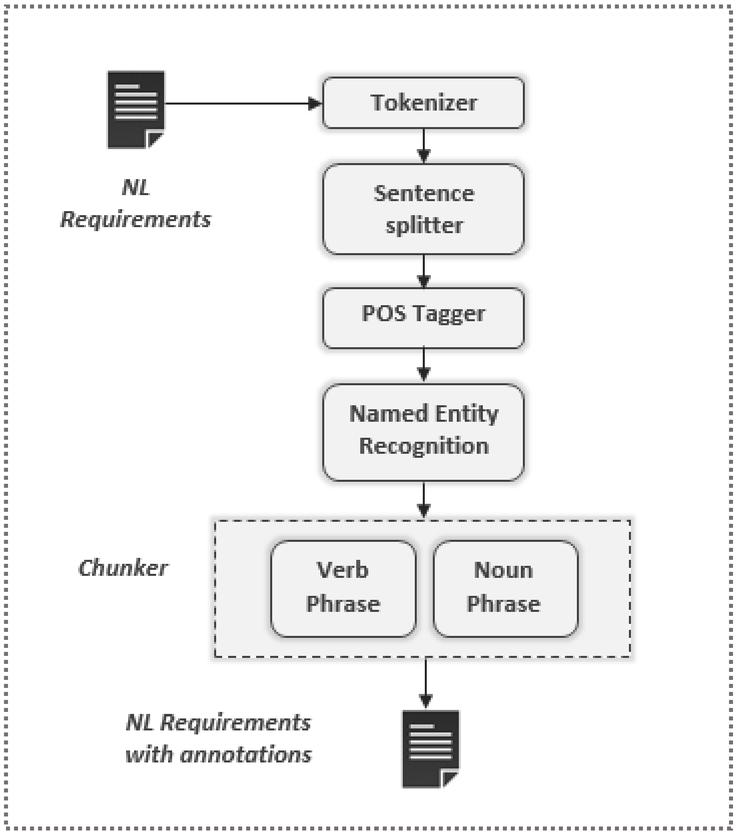


Figure 2. NLP Pipeline for conformance checking

1.3 Structure of Document

The rest of thesis has been organized as follow.

- Chapter 2 – Software Requirements and its templates: In this chapter we will provide a detailed analysis on system requirements and its different types. Furthermore we will explain why it is beneficial to use natural language to document requirements over other techniques. This chapter will explain the need for requirement templates and explain their working

- Chapter 3 – The literature review: This chapter will provide some related work in context of requirement, requirement templates and their automated conformance.
- Chapter 4 – The proposed approach: In this chapter we will provide the detailed approach of our methodology including problems relating to previous approach and how we plan to improve those approaches.
- Chapter 5 – Evaluation: In chapter 5 we will implement our proposed approach on four case studies of real world, and explain the results and evaluation of our approach against those case studies.
- Chapter 6 – Conclusion and future work: In this chapter we explain conclusion based on our thesis and future work in this area.

Chapter 2: Software Requirements and its templates

Software requirements is the field of software engineering that establishes the need of stakeholder that is to be solved using a software system. The IEEE standard glossary of software engineering terminology [21] defines a requirement as following

- A condition or capability needed by a user to solve a problem or achieve an objective.
- A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.
- A documented representation of a condition or capability as in 1 or 2.

The activities relating to software requirements can be broken down to 4 phases namely elicitation, analysis, specification and management.

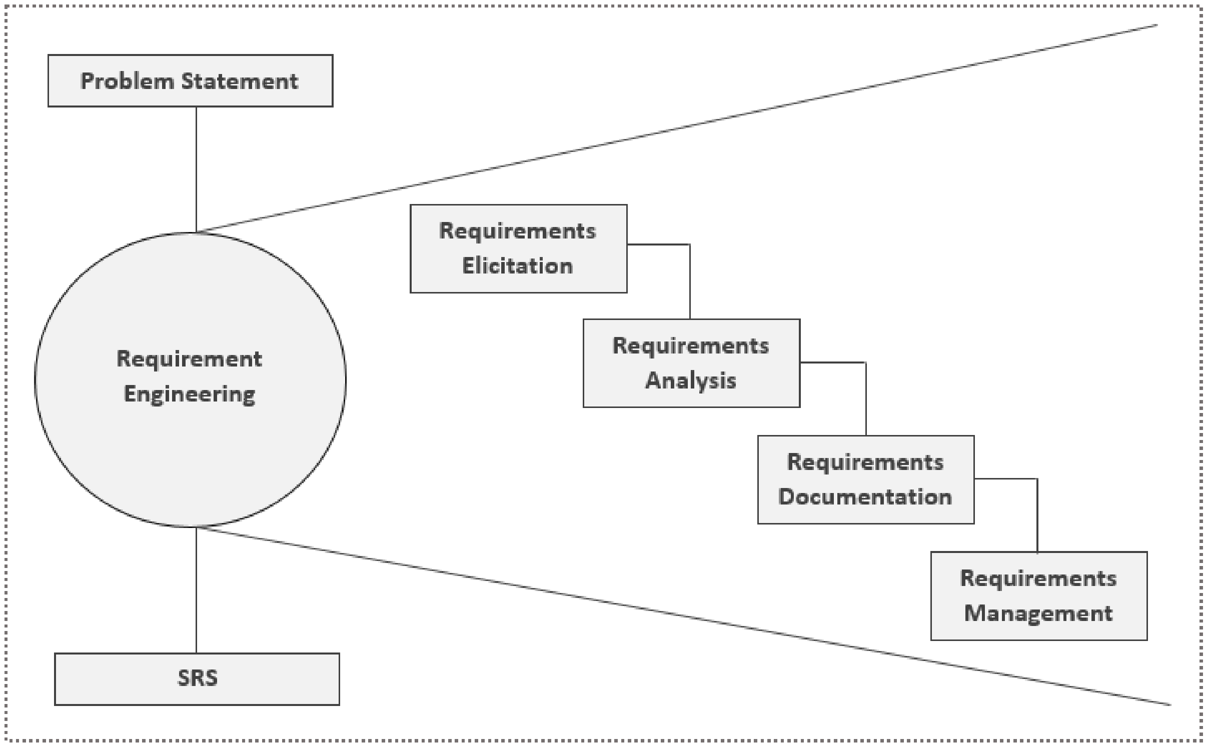


Figure 3. Overview of Requirement engineering and its core steps

1. Elicitation: The process of discovery and gathering of requirements from stakeholders and other sources (such as similar systems) is called requirement elicitation. The common techniques for elicitation are interviews, joint application designs (JAD) sessions, focus groups and document analysis etc.
2. Analysis: The next step is of analysis in which we seek a richer and more precise understanding of requirements. These include needs of requirements, identifying conflicting requirements, analyzing, documenting and validating software requirements.
3. Specification: This step involves representation of requirements in a well-organized fashion. This is most commonly achieved by making a software requirement specification document (SRS), in which we list down the function requirements, non-functional requirements, use cases, user stories etc.

4. Management: In requirements management we deal with changes in requirements during the development of project. The main goal is to ensure development of correct software system with minimum effect on current system.

2.1 Challenges in software requirement specification

One of the major problem in software development lifecycle is considered to be requirement specification [22]. The main hurdle is the fact that ambiguous, inconsistent and incomplete requirements can lead to incorrect features of software and thus project failures. The main cause of the problem is that Practitioners have a poor understanding of requirements.

Another challenge when listing requirements is that organizations tend to think their maturity level in Information Technology and Requirements Engineering is fairly high. As a result it is not deemed necessary to give users and stakeholder's further education and training methods about techniques used in Requirement Engineering [23].

2.2 Different methods for requirements development techniques

To solve problems discussed earlier many techniques for development and documentation of requirements have been devised. The most obvious technique is to use natural language (NL).

2.2.1 Natural Language

The most common technique of defining and documenting requirements is by use of some natural language, such as English. It is a common knowledge that around 72% of software requirement specification are captured using some natural language [24]. The main reason for such high percentage is the fact that a significant number of stakeholders have little to no knowledge of programming and software techniques. So organizations instead of providing training and education of techniques to enlist requirement use natural language which is generally understood by all stakeholders.

Natural language has its own drawbacks, some of the important drawbacks are

- Natural language can be ambiguous, some words and sentences can be understood as different meanings, which will result in incorrect features in developed software system.
- Requirements written in natural language are most unsuited among all techniques when it comes to dynamic translation of requirement in to object oriented specification format.
- Compound requirements can contain complex subclasses or several interrelated statements which may lead to increase in complexity.
- There may be missing requirements, particularly to handle unwanted behaviors.
- Duplication or repetition of requirements is another major issue with NL sentences.
- Wordiness (use of unnecessary number of words) is also common, especially when requirement are written by individuals who are not NL experts.
- Testability is another problem, requirements are difficult to be proven true or false when the system is implemented.

2.2.2 Non Textual Approaches

There are also other problems with the requirements, such as conflicting requirements and lack of tractability links. However these problems are not just unique to NL requirement document.

To overcome these problems associated with the NL, there are many notation based approaches for requirement specifications. Formal notifications include Petri Nets [25] and Z specification [26], Graphical notations such as Unified Modelling Language (UML) [27] and System Modelling language (SysML) [28]. Other non NL approaches include pseudo-code, Table Driven Requirement [29] and Scenario Based Approaches [30].

Some of these techniques claim universal applicability to all system levels. UML and SysML are graph based notations to describe complete functionality of systems, where different views can be created based on user needs. Scenario based Approaches can be different for different system Levels, however they not necessarily work for integrated systems. Using non-textual approaches like notation and scenario based requirement engineering can be beneficial in terms avoiding ambiguity and other common problems faced when using Natural Language. However they have their own drawbacks, some of which are listed below.

- Non-textual approaches require a process of complex translation from source requirements which can include further errors.
- Furthermore they result in language barrier between developers and stakeholders.
- There is also training overhead which is associated with introduction of such notations.

As a result of these important issues, practitioners generally discourage use of non-textual approaches. However there is a general consequences to use natural language for requirement engineering with some defined principles and structure.

2.3 Template based requirement engineering

It was hypothesized that by applying a certain set of a requirement structure will result in a practical and effective method to write high level stakeholder requirements. These small sets must be simplified to reduce complexity. A vast amount of work was performed in this area of constrained natural language. These work included Event Condition English [31], which proposed that event should signal the trigger of rule and condition cause the specified system action. Other works in this area include Specified Technical English [32] and Attempto Controlled English [33].

The purpose of last two approaches is to make technical texts which are easy to understand by all users. They proposed to a generalized and controlled vocabulary which is sufficient to write all technical sentences. In addition to this general vocabulary, these approaches permits the unrestricted use of words that are technical names and technical verbs. In general the work is mainly concerned with requirement syntax. Although steps are taken to improve the sematic part of requirement engineering in Natural Language, but there is no claim made to cover all levels of system.

In Requirement templates also known as boilerplates provide an efficient tool for reducing ambiguity, complex structures and inconsistency in NL requirements. They are principally concerned with requirement syntax and are most suited to definition of high-level stakeholder requirements. A template organizes the syntactic structure of requirements statement to into a number of predefined slots. As a result requirement become analyzable and thus are easy to automatically verifiable.

In requirement engineering literature several templates are proposed. While our approach can be edited to cover majority of those templates, in this thesis we are using two well-known requirement templates namely Rupps (proposed by Chris Rupps) and EARS (Easy Approach to Requirement Syntax). Our choice to select these two templates as the center of our research is motivated by extensive use of these two in industry [34] [35] [36]. Furthermore there are vast amount of guidelines for these templates, which provide several advantages for training and utilizing of templates in varying environments.

2.3.1 Rupps Template

Fig. 4 shows the steps for Rupps template. The template contains six slots: (1) first slot specifies an optional slot that is only needed if the requirement contains a precondition; (2) the name of the system (3) a modal (will/shall/should) to specify the importance of the requirement; (4) the process slot which can accept three different forms. These forms are based on manner in which functionality is to be applied during development; (5) the object for which functionality is required; (6) optional details appear at the end of requirement sentence and provide detail about the object.

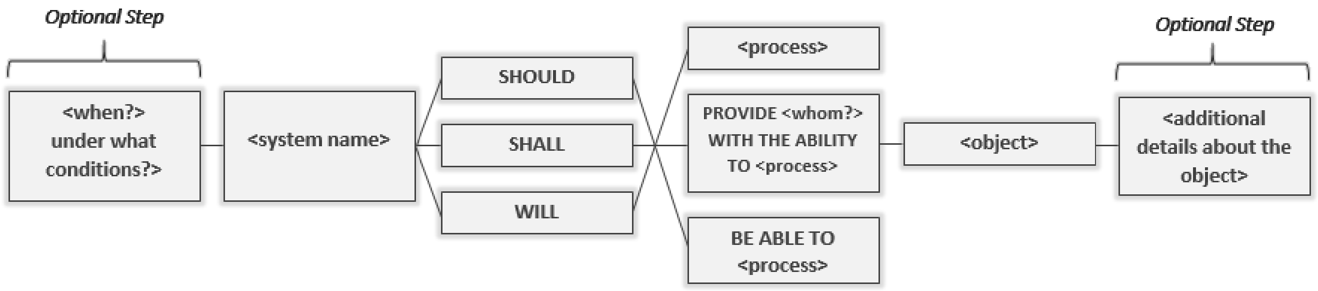

Optional Step
<when?> under what conditions?>
<system name>
SHOULD
SHALL
WILL
<process>
PROVIDE <whom?> WITH THE ABILITY TO <process>
BE ABLE TO <process>
<object>
Optional Step
<additional details about the object>

Figure 4: Sentence architecture according to Rupp's template.

The Rupps template shown in Fig. 4 distinguishes three types of processing functionality

- Autonomous Requirements, states functionalities which are independent from user interaction. These are captured using the "<process"> form.
- User Interaction Requirements, states functionalities that the system provides to user. These are captured using "PROVIDE <whom?> WITH THE ABILITY TO <process>".
- Interface Requirements, states functionalities the system performs in response to trigger events from other systems. These are captured using "BE ABLE TO <process>".

2.3.2 EARS Template

The EARS template is made of four slots and shown fig. 5: (1) an optional condition at the beginning; (2) the name of system; (3) a modal (only SHALL is accepted in EARS template); (4) the system response.

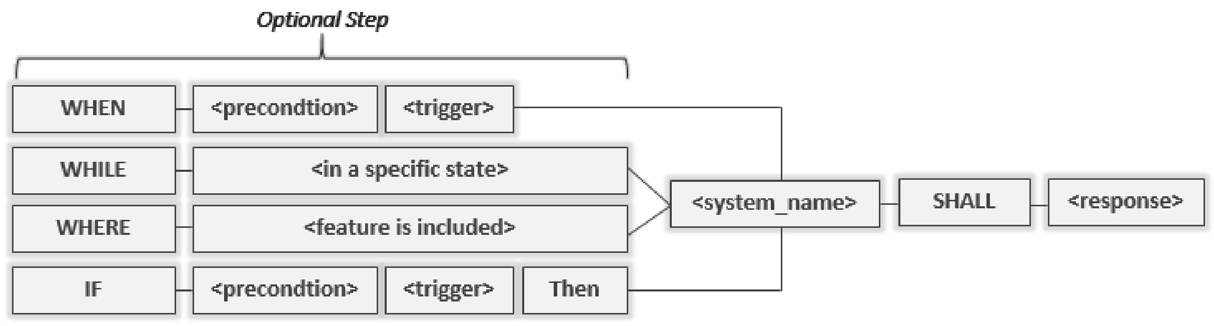


Figure 5: Sentence architecture according to EARS template.

EARS template uses five alternate structures of first slot to distinguish requirement types.

- Ubiquitous requirements, are always active and have no pre-condition.
- Event Driven requirements, are initiated by a trigger event and always begin with WHEN.
- State Driven requirements, are active for a definitive state and begin with WHILE.
- Optional Feature, are fulfilled when certain optional features are present and begin with Where.

- Unwanted behavior requirements, these requirements are used to handle unwanted behaviors including error conditions, failures, faults, disturbances and other undesired events. Use the “If” and “then” keywords.

Observing these two templates, we come to conclusion that, EARS template offers more advanced features for specifying conditions on requirements. In contrast the RUPPS template enforces more structure than EARS when it comes to non-conditional parts of requirement statement.

Chapter 3: The literature review

The selected papers for literature review represent processing on natural language requirements. The selected papers are divided into three categories representing the type of dominant approach or technique they follow. The three approaches are NLP based approaches, Machine learning based approaches and other approaches which include feature based and rule based.

3.1 NLP based approaches

The NLP based approaches for process natural language requirement documents are usually sub-divided to two NLP categories of Syntax and Semantics. For syntax-based approach well know techniques are Lemmatization, POS tagging, stemming etc. For semantic based approaches some of the most common techniques are Word sense disambiguation, Named entity recognition etc. In most of research papers a combination of more than one technique is usually adapted. Following are the papers where the dominant technique for processing is NLP based.

3.1.1 Change Impact Analysis for Natural Language Requirements: An NLP Approach

In this paper [37], the authors propose an NLP based approach to analyze the impact of a change in requirements. The proposed approach automatically detects the phrase structure of requirements. An argument is made that capturing conditions which result in change of requirement is important when analyzing the impact of changes. The proposed process starts with text chunking of requirement, then extraction of Noun and Verb Phrase tokens, this is followed by applying similarity measures where authors use Levenshtein measures for syntactic similarity and Path measures for semantic similarity measures. The evaluation and testing were performed on 14 scenarios from two separate industrial case studies. The results showed correct detection of 99% of impacted requirements.

3.1.2 Can Clone Detection Support Quality Assessments of Requirements Specifications

In this paper [38], the authors propose a methodology to identify and reduce the redundancy that arises from cloning (copy & paste) during requirement specification process. Their methodology to detect cloning is a three steps process: *Input and Preprocessing* in which text is split into single words from which whitespace and punctuation is discarded, followed by stemming (reducing words to its stems) using the Porter Stemming algorithm; *Detection* in which token-based approach is used to construct a suffix tree in which, those branches that reach at least two leaves correspond to a clone and are reported; *Post-processing* in which all overlapping clone groups are removed and rest are reported as a report. The proposed methodology is applied to 28 requirement specifications with the results show precision of up-to 99% for each alphabet, which letter 'U' with the lowest precision of 85%.

3.1.3 Documenting requirements specifications using natural language requirements boilerplates by Noraini Ibrahim and Wan M. N. Wan Kadir

In this paper [39], the authors present impact and benefits of using predefined boilerplates or templates to requirement specifications. The study was motivated by the need to find a better way of eliciting requirement from the novice stakeholders point of view and understanding. The templates were divided into the broad category of requirements, namely functional and non-functional. The evaluation was performed on an industrial health care application case study, the Meidnet System. The results indicated that boilerplates resulted in a reduced impact of unclear requirements that affect other artifacts and hence minimize the risk of volatility issues. Furthermore, the cost and impact of requirements was also greatly reduced.

3.1.4 Rapid Requirements Checks with Requirements Smells: Two Case Studies

In this paper [40], they propose an approach to detect issues in requirements based on defected phrases which this paper labels as bad smells. They apply a four-step NLP based light weight technique for instant checks as soon as the requirement is written down. These four steps include *Parsing* of requirements statements; *Annotation* in which POS tagging, Morphological Analysis and Lemmatization is performed; *Smell identification* the algorithm for find defected phrases; *Presentation* of findings with explanations. They applied the approach on two cases studies with a total of 336 requirements and 53 use cases that are taken from 9 specifications. The results when discussed with industry experts concluded that the approach can be used to detect relevant defects in requirements. Since it cannot find all the possible defects, thus is most suited as valuable input for requirement reviewers.

3.1.5 Natural Language Requirements Specification Analysis Using Part-of-Speech Tagging

In this research [41], the author proposed a methodology to first perform a grammatical analysis on software requirements and then transform those sentences into formal models. The motivation for this paper is that natural language provides advantage for requirement documentation since they are easily understandable to all stakeholder types, however they are prone to ambiguousness and inconsistency. On the other hand, the formal models provide better requirement specification in terms of lower ambiguity and consistency but are not easily understandable to some groups of stakeholders, especially the non-technical ones. For formal models this paper uses the Concern-Aware Requirement Engineering (CARE). The five steps for above mentioned conversions are; (1) Breaking each requirement sentence to its composing words; (2) part of speech (POS) tagging, where each word is defined according to its English corpus; (3) Sentence pattern matching, where each sentence is classified into its grammatical elements; (4) Match each sentence from step 3 to CARE sentence pattern; (5) Creation of CARE scenario formats (formal models). The resulting formal models can further be converted into object-oriented models.

3.2 Machine Learning based approaches

Machine learning based coupled with NLP in requirement documents is predominantly used to identify ambiguities inside the document based on previous metrices. It’s also used to classify or categorize individual requirements to a particular set of requirements. Some of the most commonly machine learning techniques in NLP are Decision Trees and KNN classifiers.

3.2.1 Identifying Nocuous Ambiguities in Natural Language Requirements

In this paper [42], the authors present a technique to automatically alert the writers of requirements to a presence of potentially dangerous ambiguities. The authors start the paper by defining ambiguity in Requirement Engineering and presenting different type of ambiguities including acknowledged vs unacknowledged ambiguity. This paper suggests the idea to use a morphological analysis draw conclusions from three methods. First the *Weighted method of interpretation* in which unacknowledged type of ambiguity is given high weight since its more likely to cause nocuous ambiguity in requirement document; secondly using a flexible method for thresholds to classify nocuous and innocuous ambiguity; In the third method authors suggest an approach to deal only with unacknowledged ambiguity can be extremely effective and efficient. For execution and testing the authors suggest the use of a machine learning approach to attain external linguistic data for requirement ambiguities. The results show that morphological heuristics can be more effective for reading ambiguities and suggesting cut of points.

3.2.2 Analyzing Anaphoric ambiguity in NLP

In this research [43], the authors present an automated system for identifying potential harmful ambiguities in requirements that result from anaphoric ambiguities. The anaphoric ambiguity arises when the readers of requirement document may disagree on how pronouns should be interpreted. This paper provides a machine learning based classifier, with a given wide range of requirement documents a set of anaphoric ambiguities are extracted and associated human judgement were collected based on interpretation of individuals. The classifier hence was trained based on both predetermined heuristics and collected human judgements. This developed tool is used to highlight both actual and potential ambiguities in a requirement. Results from this paper show that the classifier achieves a high recall and precision which varies with respect to change in threshold for ambiguity.

3.2.3 Hidden in Plain Sight: Automatically Identifying Security Requirements from Natural Language Artifacts

In this research [44] the authors present a machine learning approach to identify security related requirements in a document. The developed tool takes inputs of natural language document and attempts to identify security related sentences and classify them on the basis of security objectives. The tool is composed of four steps; (1) *Pre-Process Artifacts* where a document is processed to identify requirement sentences and titles; (2) *Classify for security objectives*, here using a

KNN classifier a sentence is classified to having one or more security objectives; (3) *Select context specific Templates* where tool identifies the associated template to requirement having security objectives (e.g. accountability) and possible action based associated with it; (4) *Generate requirement sentences,* in this step sentences are modified to reflect a security related requirement e.g. introduction of explicitness to a sentence. For the evaluation almost 11K sentences were classified and with 46% of those sentences having security objectives. Of those identified to possess a security objective 22% were found to explicitly security related while the other 72% had security implications associated with them. The two metrics for testing are; (1) *Precision* where the tool classified 82% of security objectives of all requirements; (2) *Recall* where of all security objectives 79% were correctly classified.

3.2.4 Non-functional Requirements to Architectural Concerns: ML and NLP at Crossroads

In this paper [45] the authors present a machine learning based approach to automatically detect architectural aspects (e.g. Presentation aspect, data aspect, development aspect etc.) and quality attributes of non-functional requirements (NFRs). Firstly, all the non-functional requirements are converted to plain text. Then using Support vector machine, each NFR is related to an architectural pattern and quality attribute using knowledge base of previous data. To further facilitate the approach the authors used NFR2AC toolset. The execution was performed on an automotive case study, even though the knowledge base was built from a diverse domain of case studies. The results indicate that around 80% of NFRs were accurately matched to its correct architectural pattern. It is the author's belief that the accuracy of results will be significantly improved by using either a larger or matching domain knowledge base.

3.3 Other approaches including feature based and rule based

Feature and rule-based approaches in NLP for requirement specification are mostly used to check the quality of requirements against certain features or pre-defined rules. To pre-process documents for feature extraction, mostly the conventional NLP techniques like POS tagging and parsing etc. are used.

3.3.1 Natural Language Requirements Quality Analysis Based on Business Domain Models

In this research [46] the authors presented an approach for quality analysis of natural language requirement using ontology-based domain. Authors also make use of domain guided parsing for requirements mapping. The type of quality analysis discussed in this paper are divided into two groups; (1) *Comprehension enablers* including Fit-gap view, Dependency view and Clustering view; (2) *Violation Detection* including Missing attributes detection, Missing business rules identification, under specified entity identification, Undefined entities, Unspecified wrong business rule identification, unspecified case identification and identifying conflicting requirements. The methodology proposed is a two-step process; (1) *Matching terms to domain elements* using Jaccard Similarity; (2) *Matching requirements* using Domain guided parsing, a graph-based approach using calculated chunkers. For testing and execution, a dynamic of 38 requirements document was used. The results showed that a precision of 0.84 was achieved for quality-based violations in these requirements.

3.3.2 A Rule-Based Natural Language Technique for Requirements Discovery and Classification in Open-Source Software Development Projects

In this paper [47] the authors follow a rule-based approach to discover and subsequently classify open source requirements. The main problem for open source requirements is that they are informal and found in forums, comments, requests and emails. Manual Analysis for these mediums is not possible or feasible and can be error prone. The proposed approach has 6 levels for classification. The first 5 levels are not necessarily interlinked but provide necessary detail for level 6 to qualify a potential requirement. These 6 levels are; (1) *Tokenization* which defines basic element of text included in all types of communication; (2) *Parts of speech (POS) tagging*; (3) *Qualification* which identifies an expression which might indicate a requirement; (4) *Entities* which identifies three basic requirement elements (Subject(Actor), Action(Verb) and Object); (5) *Requirement* discover parts of text which may be identified as a requirement; (6) *Classification* on text identified in previous 5 levels for classification of type of requirement. This paper uses 23 quality metrices for requirements as defined by McCall. Evaluation were performed on more than 20 open source forums. And results were divided into two categories of requirement and annotation types. The precision, recall and F-measure for requirements were

0.94, 0.64 and 0.76 respectively, while same metrices for annotation types were 0.58, 0.70 and 0.46.

3.3.3 Entity Disambiguation in Natural Language Text Requirements

In this research [48] the two authors proposed a solution for terminological disambiguates problems in requirement specification. The main cause of the problem is term-aliasing where multiple terms may refer to same entity. The paper discusses two types of aliasing; (1) Syntactic aliasing which includes introduced-aliasing and abbreviation; (2) Semantic variance includes multidimensional features like location, statistics and linguistics. The main approach consists of 12 steps; (1) *Requirement Labelling* in which we label each requirement; (2) *POS Tagging*; (3) *Entity Term Extraction*; (4) *Corpus Generation* which collects all terms in a requirement; (5) *Misspelling Identification* using Levenstein distance; (6) *Abbreviation Identification* finding short forms and acronyms; (7) *Explicit Aliasing* identifying entity term pairs; (8) *Generation of Latent Semantic Model*; (9) *Similarity Computation*; (10) *Generating Alias Clusters*; (11) *User Generated alias building*; (12) *Alias Search*. The results of study indicate a Precision, Recall and F-Measure of 0.86, 0.46 and 0.60 respectively.

3.3.4 Semi-automatic Checklist Quality Assessment of Natural Language requirements for Space Applications

In this paper [49] the authors propose a methodology for detecting problems with requirement specification. The underlying case study is related to a space application where authors argue that defects are potentially fatal. A checklist is used for quality assessment of the SRS document. The methodology uses a feature-based approach to verify 22 questions in an SRS document. The 22 questions are divided into four categories; (1) *Trackability* has 5 questions, where it is verified that all requirements are traced to at least a system or an interface; (2) *Incompleteness* has 4 questions, where indications of incompleteness is detected e.g. terms like ‘To be defined (TBD)’ or ‘To be Confirmed (TBC)’; (3) *Incorrectness* has 10 questions to find terms like ‘might’ or ‘in preference’; (4) Consistency has been assigned 3 questions where attempt to detect conflicting information is made.

3.4 Conclusion

The majority of researches for processing on Natural Language Requirements are either NLP based or rely on some core NLP techniques like POS tagging, parsing and tokenization etc. In majority of the research papers, a common observation can be made that none of the researcher claim 100% results on processing, rather rely on practitioners to use the developed tools and techniques to assist them. The practitioners and document writers can use these tools to check quality of requirements or extract meaningful information from requirements. They can also use some tools to convert natural language requirements to more technical representation e.g. object-oriented flows etc. All papers claim natural language to be most widely acknowledged source of requirement documentation which is acceptable and understandable to all stakeholders.

Chapter 4: The proposed approach

Our main NLP approach for automated verification of requirement templates is text chunking. In general text chunking is the process of decomposition of a sentence into smaller, non-overlapping segments know as chunks of a sentence [50]. The two main chuncks of sentence are noun phrases and verb phrases. Fig. 6 shows a requirement statement, followed by a segments of text chunking of the statement, while Fig. 7 shows its parse. A noun phrase is the chunk which can be object or subject of a verb and a verb phrase is the chunk that contains verb with an associated adverb or modal.

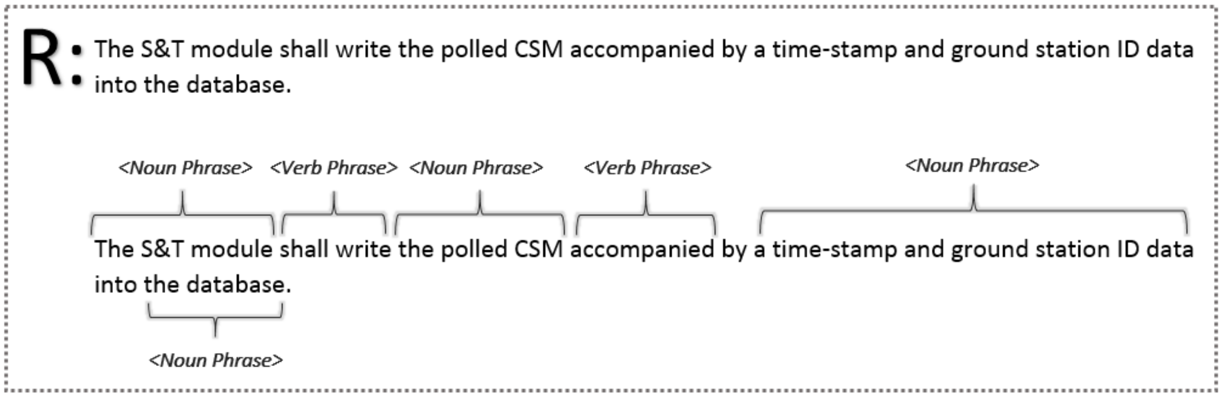

R: The S&T module shall write the polled CSM accompanied by a time-stamp and ground station ID data into the database.
<Noun Phrase>
<Verb Phrase>
<Noun Phrase>
<Verb Phrase>
<Noun Phrase>
The S&T module shall write the polled CSM accompanied by a time-stamp and ground station ID data into the database.
<Noun Phrase>

Figure6. Sentence chunks for a sample requirement

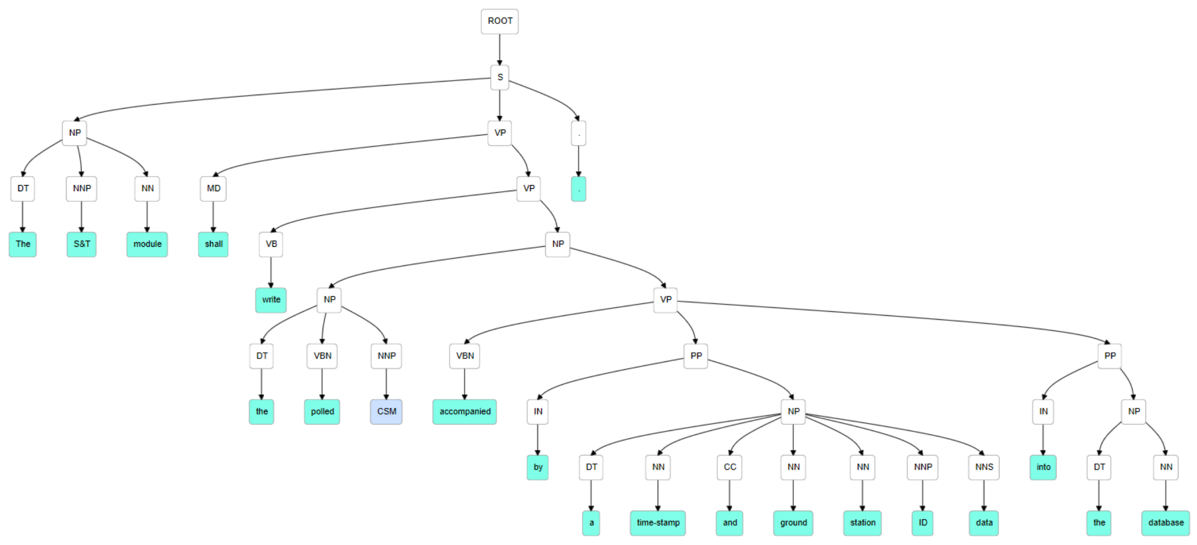


Figure 7. Parse Tree for a sample requirement

A segment generated by text chunking differs from segments in parse tree generated by a natural language parsing library which can have arbitrary depth. Stanford parser is one most well-known parsing library [51]. In NLP when a parse tree is not required we can use text chunking for it can provide two important advantages [52].

- First the text chunking is computationally less expensive, having a complexity of O(n) as compared to O(n^3) for parsing, where n denotes the length of the sentence. This makes text chunking more scalable than parsing which is extremely important consideration when we deal with large requirement documents.
- The second advantage that text chunking offers is robustness [53]. Hence it can offer results in majority of cases where parsing may fail e.g. when we face unfamiliar input. This is another important consideration because more technical requirement documents can deviate from common texts.

4.1 The NLP pipeline

As a result of considerations described in above segment, we conclude that text chunking is better suited to our approach than Parsing. Our approach will thus utilize these chunks of texts to validate a requirement statement. To prepare a requirement document to perform text chunking, the document passes through several phases. We call these phases our NLP pipeline. The basic flow of NLP pipeline is already described in Fig. 2. Next, we will explain each major part or phase of this pipeline.

4.1.1 The Tokenizer

In first phase we break the input document into a list of tokens. A token can be a word, symbol or a number. For our implementation we are going to use OpenNLP Tokenizer [55].

4.1.2 The Sentence Splitter

After Tokenizer, the next phase is to divide each requirement statement into an individual sentence. Marking Sentence is important since we are essentially marking a valid or invalid requirement sentence. For our approach we will use both ANNIE sentence splitter [54] and OpenNLP sentence splitter [55].

4.1.3 The POS Tagger

The next step is POS (part of speech) tagger. This tags each token to a part of speech which includes Nouns, Verbs, Adverbs and Adjectives among others. Most POS taggers use a Penn Tree-Bank tag set [56]. For our approach we are using OPEN POS Tagger [55] and Stanford POS Tagger [57].

4.1.4 The Named Entity Recognizer

After POS tagger the next step is to identify named entities e.g. Locations, Persons and Organizations. This step helps us identify individual elements such as system name etc. In terms of requirement document, we can also include component names and domain keywords.

4.1.5 The Text Chunker

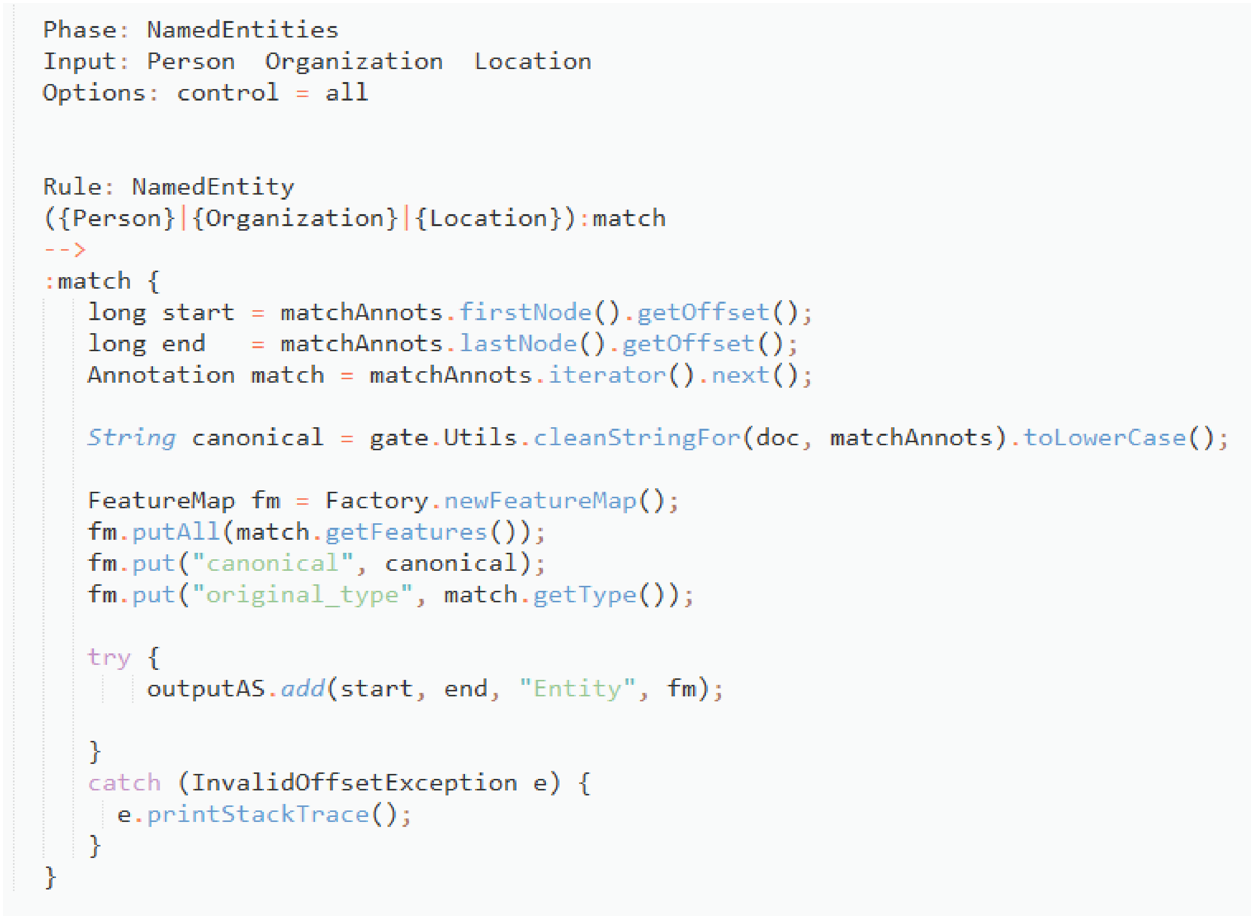
The final step in our approach is the text chunker. As described earlier the most important is chunking of noun phrases and verb phrases. We will handle both noun and verb phrases in a separate modules. We can also use glossary for Noun phrases in named entity recognition to minimize errors of NLP chunker.

These steps can be applied in different order as well. E.g. we can use both phases of POS tagging and named entity recognition before the step of sentence splitting.

4.2 Pattern matching in Requirement document

Once a pipeline is processed we will use annotations for tokens, parts of speech, named entity recognition, sentences, verb phrases and noun phrases. Next task is to represent template. We will use BNF (backus naur form) for representation. This will enable us to write rules for pattern matching over requirement statements. For this we will use JAPE (Java Annotation Pattern Engine) which is a regular expression based pattern matching language. It’s available as a part of GATE (General Architecture for text engineering) NLP workbench.

Each JAPE script has a set of phases and each phase has a set of rules. In Fig. 8, we show phase “NamedEntities” which has a single rule “NamedEntity” to mark named entities.



```
Phase: NamedEntities
Input: Person  Organization  Location
Options: control = all


Rule: NamedEntity
({Person}|{Organization}|{Location}):match
-->
:match {
   long start = matchAnnots.firstNode().getOffset();
   long end   = matchAnnots.lastNode().getOffset();
   Annotation match = matchAnnots.iterator().next();

   String canonical = gate.Utils.cleanStringFor(doc, matchAnnots).toLowerCase();

   FeatureMap fm = Factory.newFeatureMap();
   fm.putAll(match.getFeatures());
   fm.put("canonical", canonical);
   fm.put("original_type", match.getType());

   try {
       outputAS.add(start, end, "Entity", fm);

   }
   catch (InvalidOffsetException e) {
     e.printStackTrace();
   }
}
```

Figure 8. Showing a JAPE script to mark named entities.

In Jape each rule consists of a LHS (left hand side) and RHS (right hand side) which are separate by '->'. The LHS of a rule represent the annotation pattern to be matched and RHS defines the action to be taken on the specified LHS annotation pattern. As displayed in fig. 8 the RHS can also contain Java code to manipulate annotations.

When multiple rules match on a segment, JAPE provides options for controlling the results of annotations. These options are following

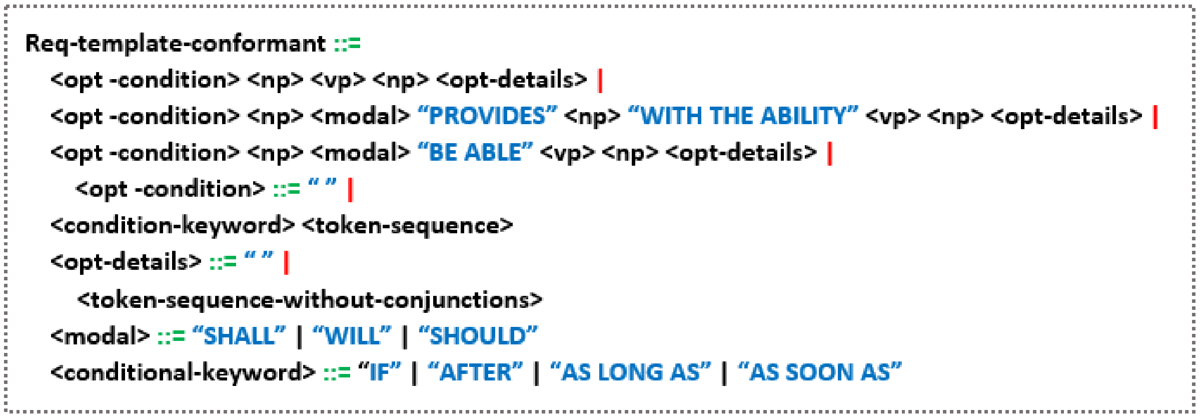
- Brill: This option marks that for a given text region when multiple rules are matched then all rules are fired. Brill is especially useful to detect ambiguities in a requirement sentence, if more than one ambiguities are present then all ambiguities will be annotated.
- First: This marks that when more than one matching rules are present for a segment, the first shortest rule is fired. This is used as an example to detect

a sentence, where we need to detect a sequence of tokens followed by a full stop to mark a sentence.

- Appelt: This option specifies when multiple rules are present the long possible segment of document rule is fired. This is useful to mark paragraphs in a document.

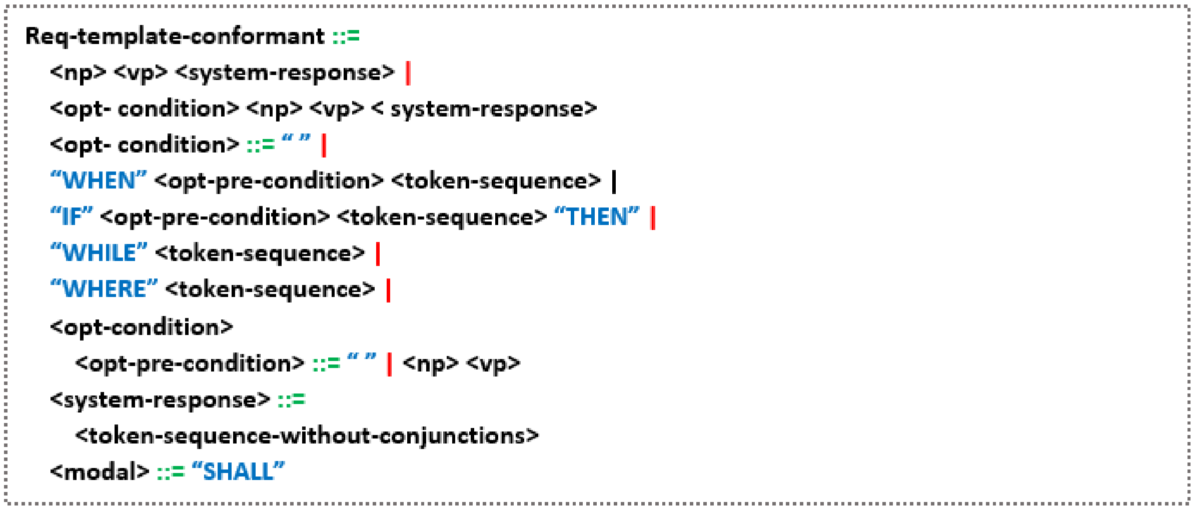
4.3 Expressing the template as BNF grammar

In figs. 9 and 10 we show how BNF grammar is constructed for Rupp's and EARS template respectively.



```
Req-template-conformant ::=
  <opt -condition> <np> <vp> <np> <opt-details> |
  <opt -condition> <np> <modal> "PROVIDES" <np> "WITH THE ABILITY" <vp> <np> <opt-details> |
  <opt -condition> <np> <modal> "BE ABLE" <vp> <np> <opt-details> |
    <opt -condition> ::= " " |
  <condition-keyword> <token-sequence>
  <opt-details> ::= " " |
    <token-sequence-without-conjunctions>
  <modal> ::= "SHALL" | "WILL" | "SHOULD"
  <conditional-keyword> ::= "IF" | "AFTER" | "AS LONG AS" | "AS SOON AS"
```

Figure 9. BNF grammar for Rupps Template



```
Req-template-conformant ::=
  <np> <vp> <system-response> |
  <opt- condition> <np> <vp> < system-response>
  <opt- condition> ::= " " |
  "WHEN" <opt-pre-condition> <token-sequence> |
  "IF" <opt-pre-condition> <token-sequence> "THEN" |
  "WHILE" <token-sequence> |
  "WHERE" <token-sequence> |
  <opt-condition>
    <opt-pre-condition> ::= " " | <np> <vp>
  <system-response> ::=
    <token-sequence-without-conjunctions>
  <modal> ::= "SHALL"
```

Figure 10. BNF grammar for EARS template

In both Rupp's and EARS a requirement can start with an optional condition, where Rupp's template does not provide syntax for that pre condition, recommending only use of following phases.

- For logical conditions use IF.
- For temporal conditions use AFTER, AS SOON AS and AS LONG AS.

EARS in contrast differentiates the optional conditions in a requirement. Also using recursion to mark complex requirement, where more than one pre-condition is marked.

We can use a hard rule to enforce the end of conditional segment to always end with a comma. This maybe too constraining because using commas can be forgot or may depend on a person's individual choice of punctuation. We can use heuristics to avoid reliance on use of comma to identify the conditional segment in a requirement statement. For example we can use an NP system name followed by modal such as SHALL to identify a conditional part.

4.3.1 Gazetteers

Both Rupps and EARS have template specific keywords, these may include modals (e.g. SHALL, SHOULD, WILL) or conditional keywords (e.g. WHEN, IF) etc. We group them in a separate list of keywords. These lists are called gazetteers in Natural language Processing [58]. These help our automated tools to be generic for both templates thus decoupling them from template specific keywords.

4.3.2 System Response

For system response in EARS and optional details in Rupps, we accept any sequence of token. One condition for these tokens is that they must not contain a subordinate conjunction (such as after, unless, before etc.). The reasoning behind this logic is that any subordinate conjunction may enforce an additional condition on requirement and both Rups and EARS state that all conditions must be defined at the beginning of the requirement rather than at the end.

4.4 Conformance Checking Steps

Template conformance starts with text chunking shown in fig. 2. It identifies tokens, sentences, parts of speech, named entities, noun phrases and verb phrases. After that another text processing pipeline is executed. The overview of this pipeline is show in fig. 11. Again JAPE is used to write rules and scripts for this pipeline. Below we explain each step of this second pipeline followed by a sample JAPE script to mark requirement type.

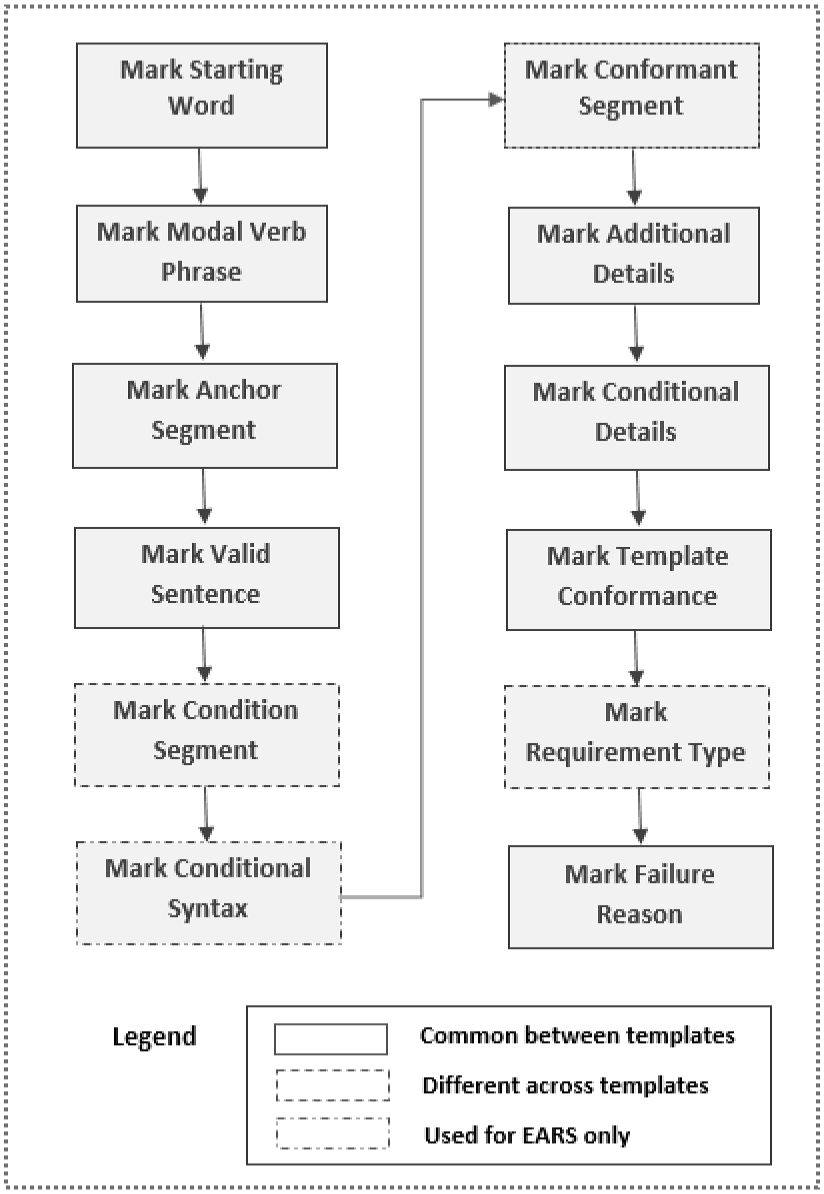


Figure 11. Pipeline for template conformance

4.4.1 Mark starting word

First step is to mark the first word of the requirement sentence. This word may start a condition segment or anchor segment. In fig. 11 we mark this step as mark start.

4.4.2 Mark modal verb phrase

Mark the sequence of a modal followed by a VP (e.g. SHALL PROVIDE). A requirement sentence normally has only one modal, if more than one modal is found in a sentence we generate a warning. The resulting annotation from this step is represented as Modal_VP in fig. 11.

4.4.3 Mark Anchor segment

The first system_name is that is followed by Modal_VP in step 2. The absence of a system_name that does not preceded the modal results in error generation. In Fig. 12, the requirement 3 doesn't contain an anchor tags. Normally an anchor segment is at the start of the statement or preceded by the condition segment.

4.4.4 Mark Valid Sentence

In this we mark a sentence valid or un-valid based on condition that it must contain an anchor tag at the beginning of a sentence or immediately followed by the condition segment. Sentence not having an anchor or don't meet some additional constraint are marked invalid. In example shown in fig. 12 Req-1 and Req-2 are valid sentences but Req-3 is an invalid sentence.

4.4.5 Mark condition segment

Here we mark the optional condition in the valid sentences. The condition segment is all the text from the beginning of the sentence to the anchor segment. Since the structure for conditions in both Rupps and EARS are not same, the scripts written in JAPE for this step are different.

4.4.6 Mark Conditional syntax

This step is exclusive to EARS template since it defines a certain syntax for conditions of requirement. For example conditional keyword IF is followed by an optional precondition, trigger and keyword THEN .Later on we will mark requirement type for EARS based on condition keyword used.

4.4.7 Mark Conformant Segment

The conformant segment is a valid sentence requirement until the additional details segment. Like conformant segment the rules for this also vary in the two templates. Thus the scripts will also be different for the two defined templates.

4.4.8 Mark additional details

Additional details are optional in both templates. This step is only valid for sentences that contain a conformant segment. Every token after conformant segment is considered a part of additional details. In EARS this segment is also called system response.

4.4.9 Mark Conditional details

As described earlier that both templates force to define conditions in start of a requirement sentence. In this step we check for details which contain conditions, most notable the subordinate conjunction. If found we trigger non-conformance. We normally use gazetteers to define conditional keywords that must not be present in the detail of requirement.

4.4.10 Mark template conformance

All requirement with a valid sentence, a conformant segment and non-presence of conditional details are marked as Template_Conformant. Any requirement without this segment will be marked Template_Non_Conformant. In the example of fig 12, req-1 and req-2 conform to templates while req-3 is marked as non-conforming requirement.

4.4.11 Mark Requirement type

As shown earlier for Rupps we have three requirement types while for EARS we have six. In this segment we mark the requirement types of a valid requirement based on presence of keywords and their sequence described earlier.

4.4.12 Mark non-conformant reason

All those requirements which do not conform are assigned a reason for the non-conformant. We trigger the first reason found for non-conformant. For example a reason lack of anchor segment may or may not have a conditional detail segment.

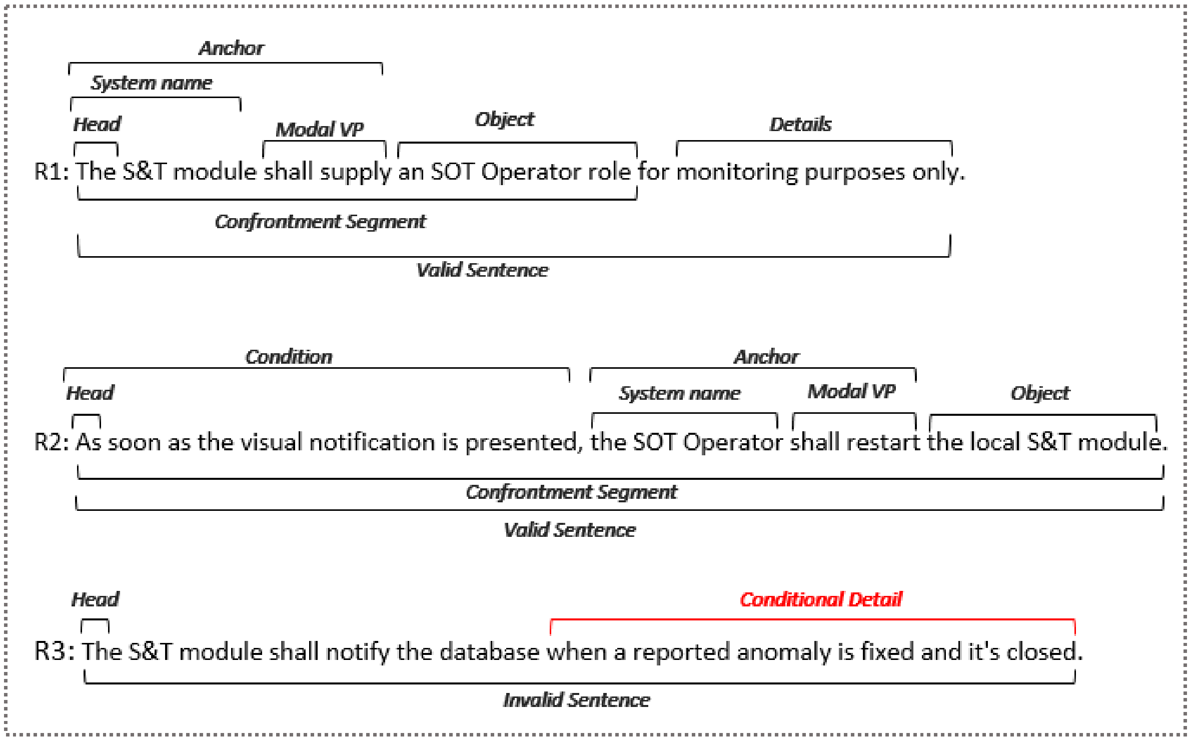

Anchor
System name
Head
Modal VP
Object
Details
R1: The S&T module shall supply an SOT Operator role for monitoring purposes only.
Confrontment Segment
Valid Sentence
Condition
Anchor
Head
System name
Modal VP
Object
R2: As soon as the visual notification is presented, the SOT Operator shall restart the local S&T module.
Confrontment Segment
Valid Sentence
Head
Conditional Detail
R3: The S&T module shall notify the database when a reported anomaly is fixed and it's closed.
Invalid Sentence

Figure 12. Annotations generated on example requirements

4.5 Identifying and warn about Complex Phrases

Apart from template conformance, we also provide cautions and warning by detecting complex and vague segments in a sentence. NLP can be used to detect these segments. In Table 1. we list the potential ambiguous annotations that represent the presence of these segments. The identification of these segments will also provide help to requirement practitioners and will thus lead to writing of clean and unambiguous statements.

Annotation	Example	Potential reason for caution
Caution_And	The M&C shall provide user with the ability to remotely monitor **and** control via SNMPv3.	The conjunction 'and' provides many problem such as multiple condition to be fulfilled.
Caution_Or	The M&C shall visually distinguish between planned unavailability **or** M&C detected only outage.	The conjunction 'or' may suggest use of an 'inclusive or' or an 'exclusive or'.
Caution_Quantifier	The M&C component supplier shall deliver **all** licenses required to operator the M&C component.	Terms such as 'all' that are used for quantification can lead to ambiguousness in a requirement.
Caution_Plural_ Noun	The M&C **components** shall be designed to allow a 24/7 without interruption during routine operations with changing operator personnel.	The use of plural nouns can lead to ambiguousness.
Caution_Pronoun	The M&C components shall be implemented in a way that **they** can be deployed on both PMOC and BMOC.	Can lead to referential ambiguousness.
Caution_Complex_ Sentence	The M&C shall support the modification of **any** of **its** configuration parameters upon request.	Complex sentence such as the one which contains both quantifiers and pronouns, can imply multiple meanings.
Caution_ VagueTerms	The M&C shall **periodically** poll the DBMS for the availability of LEO orbit S/C files.	Vague terms such as periodically or acceptable should be avoided in a requirement sentence.
Caution_ PassiveVoice	The M&C shall **be developed** under TechCom's ISO 9001 quality management system.	Use of passive voice can lead to confusion for developers and thus must be avoided.
Caution_Adverb_ in_Verb	The M&C shall provide user with the ability **to remotely monitor and control** via SNMPv3.	Using adverbs in a verb phrase is discouraged.
Caution_Adjective_ FollowedBy_ Conjunction	The M&C shall visually distinguish between **planned unavailability or M&C** detected only outage.	If adjective is followed by two nouns then it's ambiguous since it may apply to only first or both nouns.

Table 1. Potentially ambiguous annotations, their examples and possible reasons.

In this section we presented our approach to template conformance including pipeline to identify conformance, requirement types and possible reasons for failure to comply. We also presented a list ambiguous terms that may help requirement practitioners to avoid problematic statements. In the next section we will apply our approach on case studies for both Rupps and EARS template and attempt to verify our results.

Chapter 5: Evaluation

For evaluation we are going to consider four case studies. For selection of our case studies we addressed many considerations. Most important consideration was coverage of industry domains that are different from each other. Also these case studies present real life scenarios. Furthermore the number of requirements in each study is realistic and enough to evaluate our methodology.

In the remainder of this chapter we will discuss, the research questions and criteria, design, execution and results we attain from the execution.

5.1 The research questions

We have several consideration in our case studies which must be handled for our evaluation purpose. Following are the research questions to handle these considerations.

Research Question – 1: *What are ideal configurations for our Pipeline?*

There are several different libraries and implementations for each stage of our NLP pipeline. Our goal is to identify a combination that produces best results for our goal based on several metrics for assessing the accuracy of results. These metric include F-measure, precision and recall.

Research Question – 2: *Measure the effectiveness for non-confrontment defects and reasons?*

As described in chapter 4, our goal is not only to identify the requirement conformance but also to provide reasoning for why a requirement is deemed non-conforming to the respective template. This will assist the practitioners to not only identify non-conforming requirements, but also to improve those requirements according to the template.

Research Question – 3: *Is the accuracy of our approach compromised by the lack of glossary?*

As described earlier, the process of collecting glossary terms can be time consuming. Furthermore, most of these terms are unused within a given case study. In this research question we attempt to identify whether lack of glossary terms in our approach will affect our accuracy.

Research Question – 4: *How scalable is our approach?*

In real life environment, the number of requirements can be hundreds and sometimes thousands. In this question we attempt to explore whether our approach can conform requirements in a reasonable time.

Research Question – 5: *How Flexible is our approach?*

In real life, the requirements are written by different practitioners and for different domains. For our case studies we have different domains, in this question we will explore whether our approach is consistent in all domains.

Research Question – 6: *Is our approach equally effective across different requirement templates?*
We are exploring two requirement templates for our research, in this research question we explore whether our approach is equally effective for both templates.

In coming sections, we will attempt to answer these questions based on the results from case studies, and provide guidelines and insight for practitioners.

5.2 The selected case studies

In this section we will introduce our selected case studies. In the table 2, we provide the basic information about our case studies, including description, domain, conformant template and the number of requirements in each case study.

Case	Domain	Description	Template	Requirements
Case-A	Satellites	Requirement for software components in a satellite ground station	Rupps	380
Case-B	Safety certification of embedded systems	Requirements for safety evidence management tools	Rupps	110
Case-C	Satellites	Requirements written in EARS template for Case-A	EARS	380
Case-D	Nuclear Energy	Regulatory requirements for nuclear safety by the Finnish Radiation and Nuclear Safety Authority	EARS	890

Table 2. Case Studies used for evaluation

5.2.1 Case-A

Case-A [59] contains requirements for a software that concerns about a satellite ground station. These requirements are written by specialists in the satellite industry. The selected requirement template is Rupps and contains a total of 380 requirements.

5.2.2 Case-B

This case study explores a tool for managing safety evidence and information for embedded systems. This case study is developed under a European project named OPENCOSS. The requirements written are in accordance to Rupps template and contain a total of 110 statements.

5.2.3 Case-C

Case-C is the transformation of Case-A to EARS template. In further sections we will explore the details of this transformation.

5.2.4 Case-D

In Case-D we explore safety requirements for nuclear facilities [60]. These requirements are written by requirement experts with nuclear safety engineers. A total of 890 requirements statement represent the full system, and are developed under EARS template.

5.3 Data Collection Process

The collection of data for our research was a two phase process.

1. Identify requirement statement and glossary terms in a requirement document.
2. Inspect the documents from the first phase and separate the conformant and non-conformant requirements.

In phase-1 we identified that the glossary was not used in Case-D. The requirements in Case-A were rephrased in Case-C using EARS and glossary from Case-A was reused. In phase-2 all the case studies were analyzed after the completion of Phase-1. Below we describe these two phases.

5.3.1 Phase-1

The requirements from Case-A and Case-B were extracted. Case-C Requirements were extracted using transformation of Case-A. This transformation process included 3 steps

1. All non-conformant were written without being rephrased.
2. All those requirements which did not contain a conditional segment were marked as ubiquitous type under EARS template.
3. All the conditional requirements were mapped to the different requirement types in EARS template.

For Case-C it is important to consider that since non-conformant requirements were not rephrased, hence some requirements that are non-conforming in Rupps may be deemed as conformant in EARS template. For Case-D requirements were written with advanced training in EARS template.

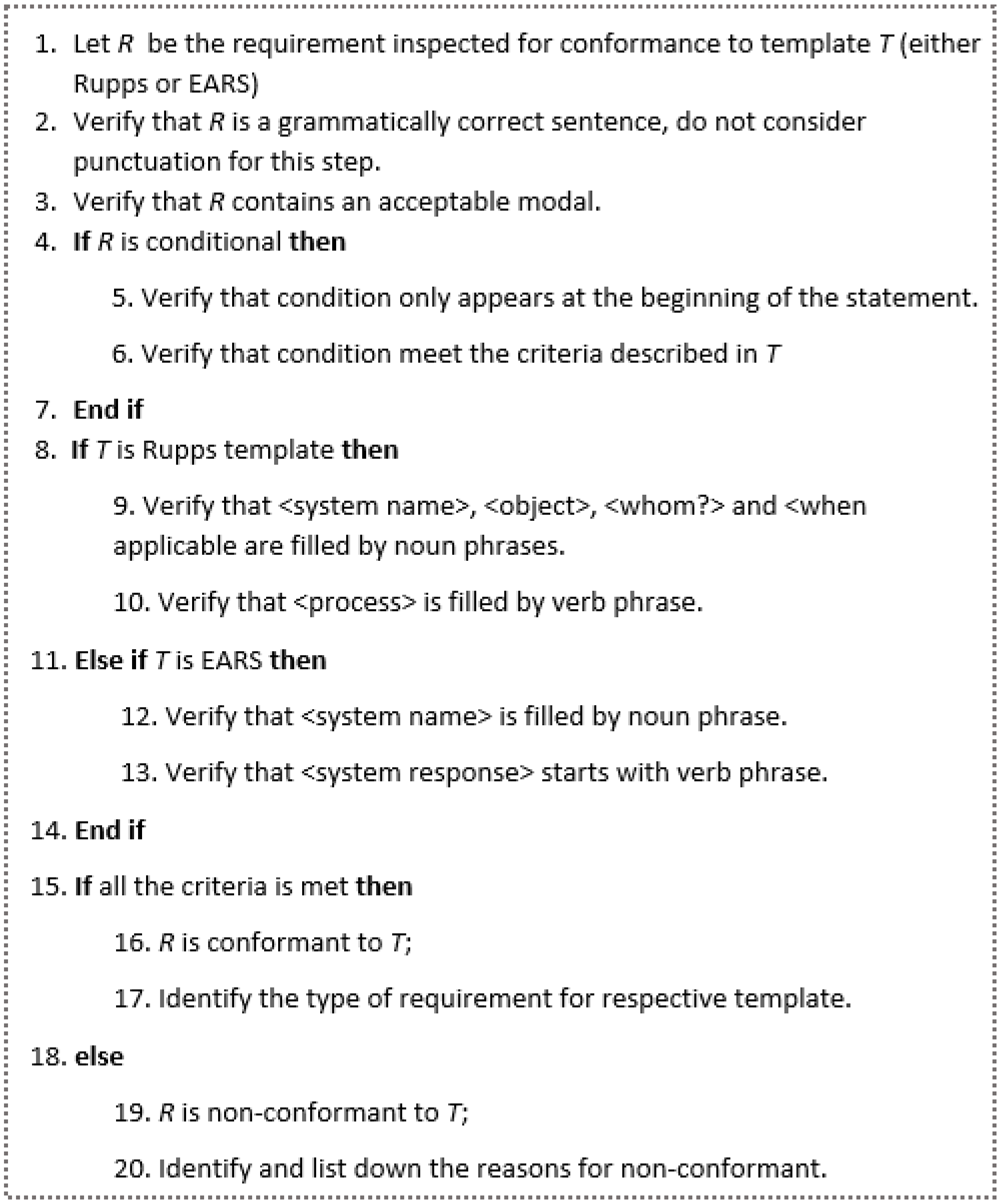
In Case-A and Case-B the glossary terms are provided, with two important consideration for collecting them.

1. We only need to identify glossary terms as compared to define them.
2. When in doubt whether a term may be included or excluded from glossary, we should favor inclusion.

These two considerations are important for our 3rd Research Question where we monitor the effect of using glossary terms. As described earlier Case-3 has same glossary terms as Case-A. For Case-D we do not have any glossary terms.

5.3.2 Phase-2

In Phase-2 we perform a manual conformance inspection of requirements from Phase-1. This will help us to calculate the accuracy of our tool by comparing results of automated conformance to manual conformance. In Fig. 13 we enlist the steps and considerations for this manual inspection.



```
1.  Let R  be the requirement inspected for conformance to template T (either
    Rupps or EARS)
2.  Verify that R is a grammatically correct sentence, do not consider
    punctuation for this step.
3.  Verify that R contains an acceptable modal.
4.  If R is conditional then
        5. Verify that condition only appears at the beginning of the statement.
        6. Verify that condition meet the criteria described in T
7.  End if
8.  If T is Rupps template then
        9. Verify that <system name>, <object>, <whom?> and <when
        applicable are filled by noun phrases.
        10. Verify that <process> is filled by verb phrase.
11. Else if T is EARS then
        12. Verify that <system name> is filled by noun phrase.
        13. Verify that <system response> starts with verb phrase.
14. End if
15. If all the criteria is met then
        16. R is conformant to T;
        17. Identify the type of requirement for respective template.
18. else
        19. R is non-conformant to T;
        20. Identify and list down the reasons for non-conformant.
```

Figure 13. Steps for manual template conformance.

While applying these steps for manual inspection, one important consideration was whether the team who wrote the requirements would like use only atomic nouns or they would also use complex noun phrases in noun phrase slots. During our manual inspection we observed that all four case studies were adapting complex noun phrases as well atomic nouns in noun phrase slots.

5.4 Analysis Process

The analysis phase is going to utilize different configurations for each module of our NLP pipeline. We will attempt to answer our first Research Question about how each configuration is effective for our requirements and furthermore which combination of configurations gives the most accurate results in a meaningful amount of execution time.

5.4.1 Different NLP pipeline configurations

For our NLP pipeline we need to choose a specific implementation for each step. Our NLP tool GATE allows us several different mature libraries for each step. This will help us experiment with different implementations. Also using a single NLP tool (GATE) allows us to access the most accurate combination of configurations, without having to consider the compatibility issues across different tools.
Our approach mainly focuses on the 5th and 6th step of NLP pipeline i.e. text chunking. However these steps rely on annotations produced by earlier steps (1-4). Hence we should not only experiment with different configurations of step 5 and 6, but also for four earlier steps. In table 3. we list down each step with number of different alternatives for the implementation of that step.

Step no.	Step Name	Alternative 1	Alternative-2	Alternative-3
1	Tokenizer	ANNIE English Tokenizer	OpenNLP Tokenizer	-
2	Sentence Splitter	ANNIE Sentence Splitter	OpenNLP Sentence Splitter	-
3	POS Tagger	OpenNLP POS Tagger	Stanford POS Tagger	ANNIE POS Tagger
4	Named Entity Recognizer	ANNIE Named Entity Transducer	OpenNLP Name Finder	-
5	Noun Phrase Chunker	OpenNLP NP Chunker	Multi-lingual NP Chunker	Noun Phrase Chunker
6	Verb Phrase Chunker	ANNIE Verb Group Chunker	OpenNLP VP Chunker	-

Table 3. NLP configuration steps and their different combinations

Many of the implementations involve machine learning. GATE allows us train data for English, hence we are going rely on that for machine learning purpose [61]

As described earlier the steps 5 and 6 are basis for conformance testing. For step 5 we have a choice whether to include the glossary terms or not. For Case 1, 2, 3 glossary is available hence, we are going to implement all these steps with and without glossary for all configurations for these case studies. The total different configurations for first 3 case studies 2 x (2 x 2 x 3 x 2 x 3 x2) = 288. Four Case-4 since the glossary is unavailable so the total steps are half as much as the first 3 i.e. 144.

5.4.2 Metrics for Accuracy

The two main metrics for measuring our accuracy are precision and recall. These two metrics are most widely used metrics. One such example for their use is in information retrieval [62]. In information retrieval we measure the accuracy of classification of objects into classes.

For our case studies we need two such class: (1) The template conformance class and (2) the template non-conformant class.

In table 4. we represent the confusion matrix for our classes which represent the possible errors an automated checker can make.

	Conformant	Non-Conformant
Conformant	True Negative (TN)	False Positive (FP)
Non-Conformant	False Negative (FN)	True Positive (TP)

Table 4. The confusion matrix for our classes

Precision is a metric for quality (low number of False Positives. The formula to define precision is

Precision = True Positive (TP) / (True Positives (TP) + False Positives (FP))

The second main metrics is Recall, it represents coverage of the solution (low number of False Negatives). The formula to define recall is

Recall = True Positive (TP) / (True Positives (TP) + False Negatives (FN))

In majority of classifications (including ours), the increase in Precision results in a decrease of Recall [63]. To solve this problem we will use a third metric F-measure [62] for accounting both Precision and Recall in measuring accuracy. This metric calculates the weighted harmonic mean of Precision and Recall. The weights allow us to emphasize more on either Precision or Recall in a given approach. In our research Recall is easier to calculate since we can rule a lesser amount of False Positives than consider a larger amount of False Negatives. As a result we will emphasize more on Precision when assigning weights to our F-measure formula. So our F-measure formula will be defined as

F-measure = 3 x Precision + Recall / (2 x Precision + Recall)

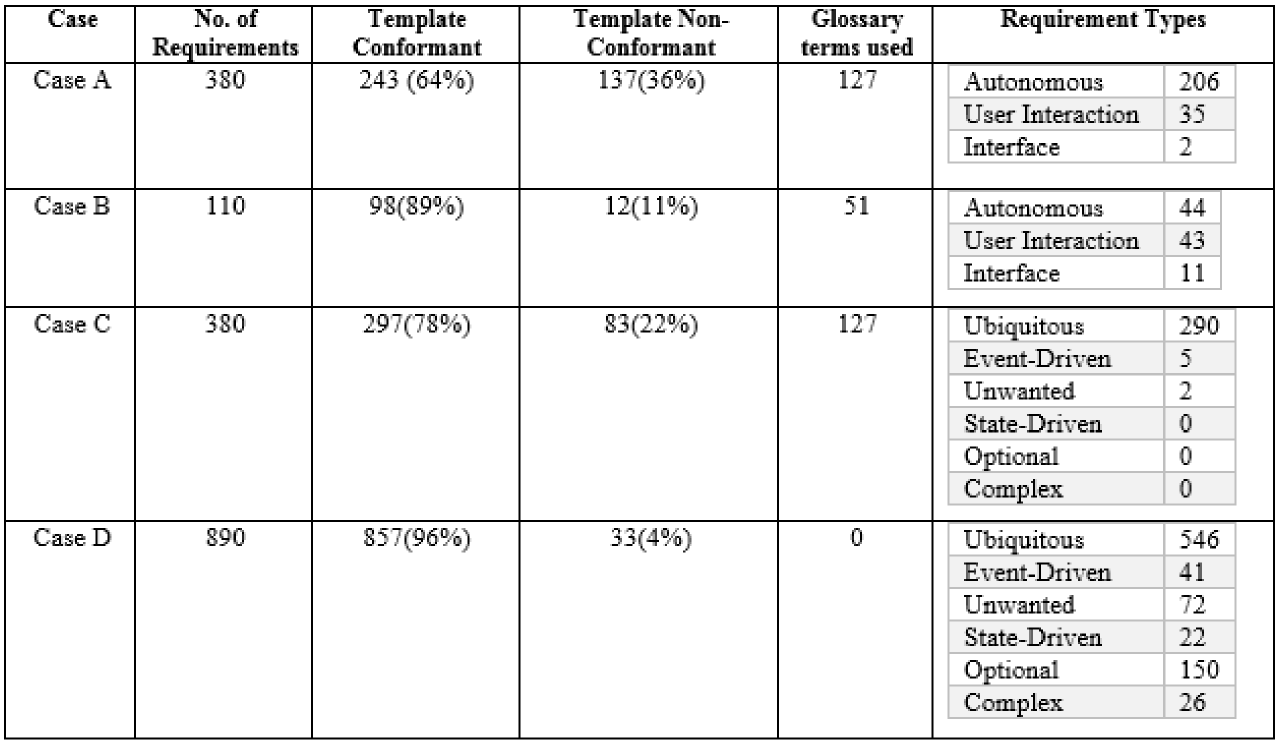
Other metrics or weights can been used, the important factor when using other formula of F-measure is to assign more emphasis on Precision as compared to Recall.

5.5 Results

In this section we will provide results from manual inspection and compare them with our suggestion for NLP pipeline configuration that we deem best based on our analysis.

5.5.1 Manual Inspection of Requirements

In table 5. we provide statistics about manual inspection for conformance on case studies. These statistics include the number of requirement that are correct, the number of requirements that do not conform to their respective template, glossary terms used for the case study and the type of requirement as defined by the template.
As mentioned previously the Case-C of our case study is extracted from Case-A. An observation can be made that in Case-C most of requirements fall in category of ubiquitous category. As described earlier since it is the simplest type of EARS requirement, hence majority of Rupps requirement fall in this category during our conversion. Furthermore the reason for high conformance rate in Case-C is also the manual conversion of requirements from Case-A and some requirements that are considered non-conformant in Rupps are valid according to EARS template.



Case	No. of Requirements	Template Conformant	Template Non-Conformant	Glossary terms used	Requirement Types
Case A	380	243 (64%)	137(36%)	127	Autonomous 206 User Interaction 35 Interface 2
Case B	110	98(89%)	12(11%)	51	Autonomous 44 User Interaction 43 Interface 11
Case C	380	297(78%)	83(22%)	127	Ubiquitous 290 Event-Driven 5 Unwanted 2 State-Driven 0 Optional 0 Complex 0
Case D	890	857(96%)	33(4%)	0	Ubiquitous 546 Event-Driven 41 Unwanted 72 State-Driven 22 Optional 150 Complex 26

Table 5. Statistic for manual inspection of requirements

In table 6. we list the reasons for non-conformance during manual inspection on requirement. A common observation can be made that most common issues for Rupps template is missing or misplaced objects, while the for EARS template the most common mistake is the misplaced condition. These two factor account for more than 75% of the total mistakes. The other minor issues include incorrect modals in verb phrases, presence of more than one object in the requirement, incorrect or unknown conditional keyword within the scope of the template, minor deviations from the template structure and incorrect or ill-formed sentence structure. Also misplaced object issue is only specific to the Rupps model as EARS provides no such restriction on the requirement statement.

In the section 5.5.2 we will provide accuracy results based on our manual inspection shown in table 6.

Type	Description	Cases	Percentage
Enumerations	Presence of more than one Object in the requirement	Case-A	14/137 = 10.2%
		Case-B	2/12 = 16.7%
		Case-C	14/83 = 16.9%
		Case-D	5/33 = 15.2%
Minor Deviations	Requirements deviate slightly from the respective template	Case-A	2/137 = 1.5%
		Case-B	0%
		Case-C	0%
		Case-D	0%
Incorrect VP Modal	Requirement has Incorrect or misplaced in its verb phrase	Case-A	0%
		Case-B	0%
		Case-C	2/83 = 2.4%
		Case-D	8/33 = 24.2%
Misplaced Condition	The conditional keyword are misplaced or multiple in a requirement sentence	Case-A	61/137 = 44.5%
		Case-B	4/12 = 33.3%
		Case-C	61/83 = 73.5%
		Case-D	18/33 = 54.5%
Incorrect Condition	The conditional keyword used is not among the ones that are prescribed by the template	Case-A	2/137 = 1.5%
		Case-B	0%
		Case-C	2/83 = 2.4%
		Case-D	0%
Ill-formed Requirement	Requirement is not a sentence or is grammatically incorrect etc.	Case-A	4/137 = 44.5%
		Case-B	0%
		Case-C	4/83 = 4.8%
		Case-D	8/33 = 24.2%
Missing or misplaced Object	Exclusive to Rupps: Requirement object slot is missing or misplaced according to template.	Case-A	61/187 = 44.5%
		Case-B	4/12 = 33.3%
		Case-C	61/83 = 73.5%
		Case-D	18/33 = 54.5%

Table 6. Percentage of reasons for non-conformance in requirements

5.5.2 Accuracy Results and Execution Time

The NLP Pipeline Configuration mentioned earlier was applied on all four cases. For each configuration we calculate the accuracy based on F-measure. In table 7. we show the most accurate NLP configurations for each case study with statistics show precision, recall, F-measure, execution time and correctly identified types.

Case	Tokenizer	Splitter	POS Tagger	Glossary	NP Chunker	VP Chunker	Time (secs)	Precision	Recall	F-measure	Correctly identified types
A	OpenNLP OpenNLP	ANNIE ANNIE	Stanford Stanford	Yes No	MUNPEX MUNPEX	ANNIE ANNIE	4s 3.5s	0.93	0.99	0.97	100%
B	OpenNLP OpenNLP	ANNIE ANNIE	OpenNLP OpenNLP	- -	ANNIE OpenNLP	- -	2s 2.5s	1 1	1 1	1 1	100% 100%
C	OpenNLP	ANNIE	OpenNLP	-	ANNIE	ANNIE	4.5s	0.96	0.98	0.98	92%
D	OpenNLP	ANNIE	OpenNLP	No	OpenNLP	-	56s	0.95	1	0.99	96%

Table 7. Showing accuracy of best NLP pipeline configuration for each case study

These experiments were conducted on fairly average computation power of Intel Core i-5 and 4 GB of RAM. In Case-A and Case-C there are 23 outliers in precision. One of the common reason in majority of these outliers is the use of MUNPEX noun phrase chunker used alongside OpenNLP Tokenizer, where in some cases MUNPEX NP chunker was misled by the Tokenizer to identify system name slot. In Case-B and Case-C there are no outliers, but F-measure is negatively affected by lower precision. The core reason for this the lower number of non-conformant requirements as shown in table 5. So even a small number of FP (False Positive) can have a significant impact on our precision and subsequently our F-measure.

5.6 Discussion on results and answers to our research questions

For our RQ-1, where we sought to find out the most accurate NLP pipeline configuration. In table 7. we show the most accurate configuration for each case study. We have different configuration for each case study due to the fact that requirements can be new or unknown to the tool. Thus we only recommend use of certain modules for each phase, but not a specific NLP module for each phase. To recommend a general NLP pipeline we can monitor the impact of a particular module on all pipelines e.g. which module causes the most amount of variation across the pipeline configuration. However such analysis is beyond the scope of this research.

In RQ-2, we question about effectiveness of non-conformant defects and correctly identifying reason for a requirement to be deemed as non-conformant. To calculate this we need two measure from our confusion metrics, false positive and false negative.

Case	No. of requirements	False Positives	False Negatives
Case-A	380	8	12
Case-B	110	1	0
Case-C	380	8	2
Case-D	890	6	1

Table 8. The number of false negatives and false positives based on accuracy level in table 7

In table 8. we show the number of false negatives and the number of based on accuracy levels in table 7. The number of false negatives is low in percentage and absolute numbers. In total the number of false positives in each Case-A are 8/137, for Case-B 1/12, for Case-C 8/83 and 6/33 for Case-D.

As evident form table 8, the number of false negatives are also negligible, 12/320 for Case-A, for Case-B 0/110, for Case-C 2/380 and for Case-D 1/890. Ideally we would like to eliminate the number of false negatives and false positives from our results, however this not practically achievable. But our approach tends to minimize their occurrence to negligible. As a result we increase our chances to correctly identify the cause of non-conformant, which can be extremely useful for practitioners writing requirements.

In RQ-3, we question about the lack of glossary terms in our approach. As evident from table 7, the inclusion of glossary terms have no effect on accuracy of our results. Hence our approach can achieve high results without glossary terms.

In RQ-4, we question about the scalability of our approach. The main aspect of scalability in our context is that our approach should be able to cover even a large set of requirement statements in a reasonable amount of time. As shown in table 7. the execution time in all approaches is fairly reasonable. Furthermore our execution time is improved by the lack of indexing through glossary terms for each requirement.

In RQ-5, we question about the flexibility of our solution. In our research we have covered two major requirement templates with minor differences in execution steps. Our approach can further be extended to work with other Natural language templates for requirement engineering.

In RQ-6, we question about our approach for different requirement templates. As evident from table 7, the results of our accuracy were similar for both EARS and Rupps template, the deviation from a perfect F-measure was irrespective of the template used for requirement conformance.

Chapter 6: Conclusion and Future work

In this research we presented an automated technique for requirement template conformance. The dominant NLP technique utilized by us was text-chunking. We used well-known requirement templates for conformance checking; Rupps and EARS. The implementation was performed on four case studies. We evaluated our approach with combination of several NLP libraries with and without presence of a glossary terms, with the goal of finding the most optimal approach, both in terms of accuracy and execution performance.
For Future work, we would like to develop an automated analysis technique of templates for consistency checking in requirements. Furthermore, we would like to automatically correct the non-conformant requirements with minor deviations without losing the true functionality those requirements represent. Also providing suggestions to the practitioners to assist them correct the requirement themselves.

References

[1] Requirements Engineering Fundamentals, Principles, and Techniques http://www.springer.com/gp/book/9783642125775)

[2] Alistair Marvin, Philip Wilkinson and Adrian Harwood “Easy approach to requirements syntax (EARS)”

[3] Peterson, J. “Petri Nets”, ACM Computing Surveys 9 (1977), 223- 252.

[4] Woodcock, J. and Davies, J., “Using Z-Specification, Refinement and Proof”, Prentice Hall, 1996.

[5] Object Management Group, UML Resource Page, http://www.uml.org/

[6] Holt, J., “UML for Systems Engineering: Watching The Wheels” (2nd edition), IEE, 2004.

[7] Alexander, I.F. & Beus-Dukic, L., “Discovering Requirements”, John Wiley, 2009.

[8] Alexander, I.F. & Maiden, N.A.M. (eds), “Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle” Wiley, 2004.

[9] Hooks, I., “Writing Good Requirements”, Proceedings of Third International Symposium of INCOSE Volume 2, INCOSE, 1993.

[10] Wiegers, K., “Writing Good Requirements”, Software Development Magazine, May 1999.

[11] Dittrich, K. R., Gatziu, S. and Geppert, A., “The Active Database Management System Manifesto: A Rulebase of ADBMS Features.”, Lecture Notes in Computer Science 985, Springer, 1995, pages 3-20.

[12] “ASD Simplified Technical English: Specification ASD-STE100. International specification for the preparation of maintenance documentation in a controlled language”, Simplified Technical English Maintenance Group (STEMG), 2005.

[13] Fuchs, N. E., Kaljurand, K. and Schneider, G., "Attempto Controlled English Meets the Challenges of Knowledge Representation, Reasoning, Interoperability and User Interfaces", FLAIRS, 2006.

[14] The SAREMAN Project: Controlled natural language requirements in the design and analysis of safety critical I&C systems. [Online]. Available: http://www2.vtt.fi/inf/julkaisut/ muut/2014/VTT-R-01067-14.pdf, 2014.

[15] OPENCOSS: Open Platform for EvolutioNary Certification Of Safety-critical Systems. [Online]. Available: http://www. opencoss-project.eu, 2012

[16] J. Carrillo-de-Gea, J. Nicolas, J. F. Aleman, A. Toval, C. Ebert, and A. Vizcaıno, “Requirements engineering tools: Capabilities, survey and assessment,” Inform. Softw. Technol., vol. 54, no. 10, pp. 1142–1157, 2012.

[17] RQA: The Requirements Quality Analyzer Tool. [Online]. Available: http://www.reusecompany.com/rqa, 2012.

[18] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer, “Requirement boilerplates: Transition from manually-enforced to automatically verifiable natural language patterns,” in Proc. 4th Int. Workshop Requirements Patterns, 2014, pp. 1–8.

[19] X. Zou, R. Settimi, and J. Cleland-Huang, “Improving automated requirements trace retrieval: a study of term-based enhancement methods,” Empirical Softw. Eng., vol. 15, no. 2, pp. 119–146, 2010.

[20] D. Jurafsky and J. Martin, Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, 1st ed. Englewood Cliffs, NJ, USA: Prentice-Hall, 2000.

[21] IEEE Computer Society (1990). "IEEE Standard Glossary of Software Engineering Terminology". IEEE Standard.

[22] “Specifying Translatable Software Requirements Using Constrained Natural Language”by Agung Fatwanto
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6295244

[23] “Challenges in Requirements Engineering” by Janis A. Bubenko jr
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=512557

[24] Mich Luisa , Franch Mariangela , Inverardi Pierluigi, Market research for requirements analysis using linguistic tools, Requirements Engineering, v.9 n. 1, p.40-56, February 2004

[25] Peterson, J. “Petri Nets”, ACM Computing Surveys 9 (1977), 223- 252.

[26] Woodcock, J. and Davies, J., “Using Z-Specification, Refinement and Proof”, Prentice Hall, 1996.

[27] Object Management Group, UML Resource Page, http://www.uml.org/

[28] Holt, J., “UML for Systems Engineering: Watching The Wheels” (2nd edition), IEE, 2004.

[29] Alexander, I.F. & Beus-Dukic, L., “Discovering Requirements”, John Wiley, 2009.

[30] Alexander, I.F. & Maiden, N.A.M. (eds), “Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle” Wiley, 2004.

[31] Dittrich, K. R., Gatziu, S. and Geppert, A., “The Active Database Management System Manifesto: A Rulebase of ADBMS Features.”, Lecture Notes in Computer Science 985, Springer, 1995, pages 3-20.

[32] “ASD Simplified Technical English: Specification ASD-STE100. International specification for the preparation of maintenance documentation in a controlled language”, Simplified Technical English Maintenance Group (STEMG), 2005.

[33] Fuchs, N. E., Kaljurand, K. and Schneider, G., "Attempto Controlled English Meets the Challenges of Knowledge Representation, Reasoning, Interoperability and User Interfaces", FLAIRS, 2006.

[34] C. Rupp and die SOPHISTen, Requirements-Engineering undManagement: professionelle, iterative Anforderungsanalyse fur die € Praxis, Hanser Verlag, Munchen, D-81631, pp. 225–251, 2009.

[35] S. Gregory, “Easy EARS: Rapid application of the easy approach to requirements syntax,” in Proc. 19th IEEE Int. Requirements Eng. Conf., 2011, pp. 1–2.

[36] J. Terzakis, “Reducing requirements defect density by using mentoring to supplement training,” Int. J. Adv. Intell. Syst., vol. 6, no. 1-2, pp. 102–111, 2013.

[37] Change impact analysis for Natural Language requirements: An NLP approach, http://ieeexplore.ieee.org/document/7320403

[38] Can Clone Detection Support Quality Assessments of Requirements Specifications, https://dl.acm.org/citation.cfm?id=1810308

[39] Documenting requirements specifications using natural language requirements boilerplates, http://ieeexplore.ieee.org/document/6985983

[40] Rapid requirements checks with requirements smells: two case studies, https://dl.acm.org/citation.cfm?id=2593817

[41] Natural Language Requirements Specification Analysis Using Part-of-Speech Tagging, http://ieeexplore.ieee.org/document/6767215

[42] Identifying Nocuous Ambiguities in Natural Language Requirements, http://ieeexplore.ieee.org/document/1704049/

[43] Analysing anaphoric ambiguity in natural language requirements, https://link.springer.com/article/10.1007/s00766-011-0119-y

[44] Hidden in plain sight: Automatically identifying security requirements from natural language artifacts, http://ieeexplore.ieee.org/document/6912260/

[45] Non-functional Requirements to Architectural Concerns: ML and NLP at Crossroads, http://ieeexplore.ieee.org/document/4668138

[46] Natural language requirements quality analysis based on business domain models, http://ieeexplore.ieee.org/document/6693132

[47] A Rule-Based Natural Language Technique for Requirements Discovery and Classification in Open-Source Software Development Projects, http://ieeexplore.ieee.org/abstract/document/5719011

[48] Entity Disambiguation in Natural Language Text Requirements, http://ieeexplore.ieee.org/document/6805412/

[49] Semi-Automatic Checklist Quality Assessment of Natural Language Requirements for Space Applications, http://ieeexplore.ieee.org/document/7781844/

[50] D. Jurafsky and J. Martin, Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics,and Speech recognition, 1st ed. Englewood Cliffs, NJ, USA: Prentice-Hall, 2000.

[51] The Stanford Parser: A statistical parser. Online].Available:http://nlp.stanford.edu/software/lex-parser.shtml, 2014.

[52] S. Bird, E. Klein, and E. Loper, Natural Language Processing with Python, O'Reilly, Sebastopol, CA 95472, 2009.

[53] M. Song, I. Song, and K. Lee, "Automatic extraction for creating a lexical repository of abbreviations in the biomedical literature," in Proc. 8th Int. Conf.

[54] GATE ANNIE: A Nearly-New Information Extraction System. [Online]. Available: http://gate.ac.uk/sale/tao/splitch6.html, 2014.

[55] Apache OpenNLP. [Online]. Available: http://opennlp.apache. org, 2013.

[56] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of English: The Penn Treebank," Comput. Linguistics, vol. 19, no. 2, pp. 313–330, 1993.

[57] Stanford Log-linear Part-Of-Speech Tagger. [Online]. Available: http://nlp.stanford.edu/software/tagger.shtml, 2003.

[58] H. Cunningham, D. Maynard, K. Bontcheva , V. Tablan, C. Ursu, M. Dimitrov, M. Dowman, and N. Aswani. (2014). Developing Language Processing Components with GATE Version 8 (a User Guide). [Online]. Available: http://gate.ac.uk/sale/tao/tao.pdf

[59] C. Arora, M. Sabetzadeh, L. Briand, F. Zimmer, and R. Gnaga, "Automatic checking of conformance to requirement boilerplates via text chunking: An industrial case study," in Proc. 7th ACM/ IEEE Int. Symp. Empirical Softw. Eng. Meas., 2013, pp. 35–44.

[60] List of regulatory guides on nuclear safety (YVL). [Online]. Available: http://plus.edilex.fi/stuklex/en/lainsaadanto/ luettelo/ydinvoimalaitoso hjeet/, 2013.

[61] H. Cunningham, D. Maynard, K. Bontcheva , V. Tablan, C. Ursu, M. Dimitrov, M. Dowman, and N. Aswani. (2014). Developing Language Processing Components with GATE Version 8 (a User Guide). [Online]. Available: http://gate.ac.uk/sale/tao/tao.pdf

[62] M. McGill and G. Salton, Introduction to Modern Information Retrieval, New York, NY, USA: McGraw-Hill, 1983.

[63] M. K. Buckland and F. C. Gey, "The relationship between recall and precision," J. Am. Soc. Inf. Sci., vol. 45, no. 1, pp. 12–19, 1994.