MBA-BA Final Project Report

Titled

Natural Language Processing for Requirement Engineering

Submitted by

Aditya Gurbaxani

Roll No.: BA004-21
MBA-BA (2021-23)

<u>Indian Institute of Management, Ranchi</u>

Table of Contents

List of Figures

1. Introduction

The very beginning of the software development lifecycle comprises of software requirements gathering. This involves extensive discussions between the developers and customers to agree upon the features and functionality of the software to be developed. Since the customers are seldom familiar with the technical terminology related to software development, the requirements are documented in natural language. This brings in many challenges during the subsequent stages of the software development life cycle.

The biggest challenge of all is the inherent ambiguity of the natural language, making it difficult to interpret. During manual examination of large set of software requirements in natural language, it is challenging to develop a big picture of what the software is about, finding inconsistencies, duplicates, and missing requirements. Manual examination is extremely time consuming and prone to human errors.

During the early stages, the requirements are quite abstract and do not provide sufficient details for the developer to work with. What may seem to be simple and straight-forward during discussions often turns out to be quite complex at the time of documentation. As the complexity of the requirements increase, the documentation requires longer descriptions in the natural language. It is even more complex for the developer to understand the requirements correctly and have the required information to proceed forward. It is not uncommon during the initial development phase to have a wide deviation between the requirements and the actual software developed.

While several standard, guidelines and tools have been incorporated to ensure that the software requirements are captured correctly, still there are cases where the requirements captured have faults. The actual reasons behind each such case might be unique, however a common solution here is to provide support in the manual examination process using natural language processing (NLP). Requirement engineering can be considered as a semi-structured problem, where the actual meaning of the statements would still need human interpretation, however the NLP system would be able to detect inconsistencies, duplicates and possible instances of missing requirements. Such a NLP-system could help speed-up the examination process, reduce the number of revisions and ultimately result in better quality software.

2. Related Work

2.1 Two Rule-Based Natural Language Strategies for Requirements Discovery and Classification in Open Source Software Development Projects

Just like any other software project, open source projects too have requirements. The requirements for the open source projects are mostly obtained through informal textual descriptions existing in requests, forums, and other correspondence. The understanding of these requirements provides the insights for the open source projects. Manual analysis of the textual descriptions is time consuming and prone to error for larger projects.

Even though partial, automated analysis of the natural language text can provide high benefit in this situation. The research study done by Radu E. Vlas and William N. Robinson compares two approaches for recognition of requirements. The first strategy explored is the grammar-based strategy, where the text is parsed as per the grammatical format. This approach breaks down each sentence into subject, action and object tags. Each set of subject, action, and object is considered as a requirement. The second approach in the study is the delimiter-based strategy. The text is divided in segments using the delimiters. Each segment may be a keyword or a key phrase. These segments are considered as the individual requirements.

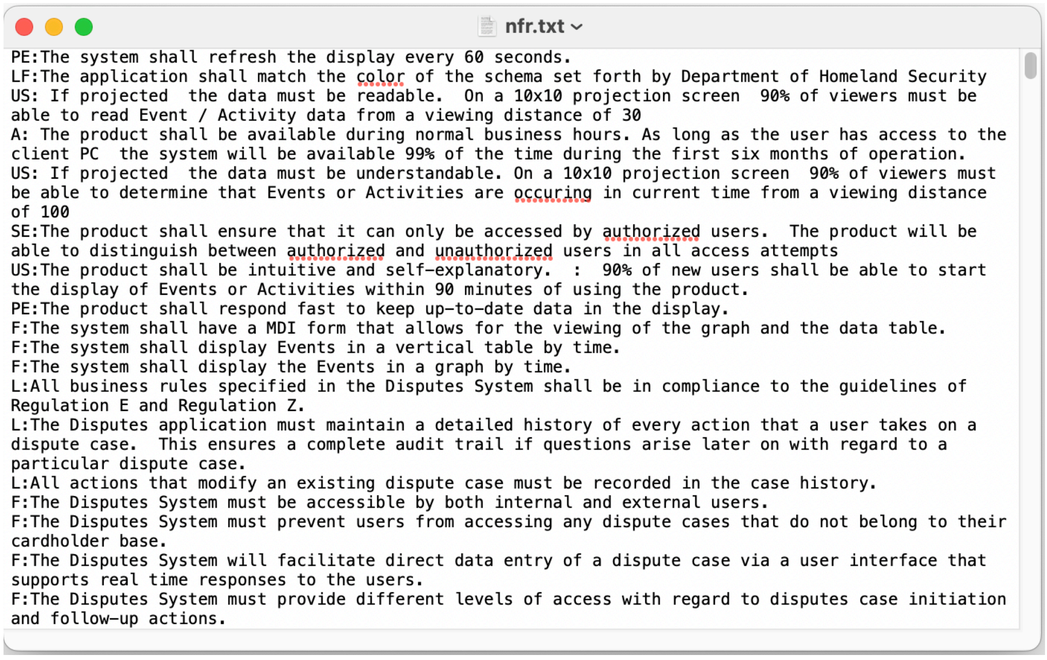
Finally, both approaches have been combined to form a hybrid approach where the text is first parsed through the delimiter-based strategy. Then the segments obtained are parsed using the grammar-based strategy. The hybrid strategy enable the recognition of aggregate requirements and the corresponding sub-requirements. While the strategy is not perfect, it was found to be capable of automating the process up to a large extent.

2.2 Model Based Specifications

Throughout the industry, many firms have approached the problem of requirement errors by employing more formal specification methodologies like Model-Based Systems Engineering (MBSE). Here instead of using natural language to document requirements, domain models are used as the primary mode of communication between developers. This form of specification is mostly mathematical, and it is also possible to test and verify the domain models.

Such models leave very low chances of ambiguity, however they too are derived from the initial top-level requirements in natural language. Since the requirements come from external stake holders, there is always scope for the vagueness and ambiguity to slip into the requirements. Even with the clear natural language requirements, there is a challenge to accurately translate them into the mathematical models.

3. Dataset



```
PE:The system shall refresh the display every 60 seconds.
LF:The application shall match the color of the schema set forth by Department of Homeland Security
US: If projected  the data must be readable.  On a 10x10 projection screen  90% of viewers must be
able to read Event / Activity data from a viewing distance of 30
A: The product shall be available during normal business hours. As long as the user has access to the
client PC  the system will be available 99% of the time during the first six months of operation.
US: If projected  the data must be understandable. On a 10x10 projection screen  90% of viewers must
be able to determine that Events or Activities are occuring in current time from a viewing distance
of 100
SE:The product shall ensure that it can only be accessed by authorized users.  The product will be
able to distinguish between authorized and unauthorized users in all access attempts
US:The product shall be intuitive and self-explanatory.  :  90% of new users shall be able to start
the display of Events or Activities within 90 minutes of using the product.
PE:The product shall respond fast to keep up-to-date data in the display.
F:The system shall have a MDI form that allows for the viewing of the graph and the data table.
F:The system shall display Events in a vertical table by time.
F:The system shall display the Events in a graph by time.
L:All business rules specified in the Disputes System shall be in compliance to the guidelines of
Regulation E and Regulation Z.
L:The Disputes application must maintain a detailed history of every action that a user takes on a
dispute case.  This ensures a complete audit trail if questions arise later on with regard to a
particular dispute case.
L:All actions that modify an existing dispute case must be recorded in the case history.
F:The Disputes System must be accessible by both internal and external users.
F:The Disputes System must prevent users from accessing any dispute cases that do not belong to their
cardholder base.
F:The Disputes System will facilitate direct data entry of a dispute case via a user interface that
supports real time responses to the users.
F:The Disputes System must provide different levels of access with regard to disputes case initiation
and follow-up actions.
```

Figure-1: Dataset Screenshot

Source: https://www.kaggle.com/datasets/iamsouvik/software-requirements-dataset

The dataset contains the requirements for different types of software products. It includes both functional and non-functions requirements. The dataset is in the form of raw text file with each line starting with a 1-2 character label, followed by a semi-colon (:), and then the requirement in natural language. The list of labels used for the requirements and their description has been provided below.

- ***Functional (F)***

  The functional requirements for a software are related to the features and user interface (UI). These requirements are directly linked to the user experience and the implementation directly impacts the software users.

- ***Availability (A)***

  The availability requirement describes when and where the software needs to be available to users. The availability can be in terms of the time, for example business hours, or in terms of the platform, for example mobile application. Additionally the availability of the software can also be in terms of the location. Certain software are only available at business locations for security reasons.

- ***Fault Tolerance (FT)***

  An environment without any faults is not a realistic scenario. Depending upon the type and type of users, software need to capable of operating in presence of faults. The exact degree and type faults under which the software should remain operational are the fault tolerance requirements.

- ***Legal (L)***

  Compliance with government regulations is essential for the software. This can be related to but not restricted to the handling of sensitive data, censorship of the user content and distribution of the software.

- ***Look & Feel (LF)***

  The Look & Feel requirements are an essential part of the user interface and user experience. This include the visual design, content positioning, user controls, etc. In addition to the natural language, the look & feel requirements of a software also need to be described using wireframes.

- ***Maintainability (MN)***

  Refers to requirements and features of the software that come into play post the production and deployment. This includes functionality such as customers being able to integrate the software with another software, customer support, provisions for customer to add new content or modify existing content from their end.

- ***Operational (O)***

  Operational requirements describe the software environment where the software needs to work. For example, for web applications it is important to document which web browsers are supported.

- ***Performance (PE)***

  Under the performance requirements we define how well the software system accomplishes certain functions under specific conditions. Performance requirements may include the software's speed of response, throughput, execution time and storage capacity. The service levels comprising performance requirements are often based on supporting end-user tasks. Performance requirements are key elements in the design of a software.

- ***Portability (PO)***

  A software is said to be portable if there is very low effort required to make it run on different platforms. For the development of a software, often it is seen that it is developed for just one platform where it is tested and refined. Once it reaches a certain level of maturity, the software is then made available on other platforms as well. However, to ensure that the software is portable in the future, it is important to have the requirement during the planning stage so that the architecture of the software is designed accordingly.

- ***Scalability (SC)***

  Software scalability is the ability to grow or shrink a piece of software to meet changing demands on a business. Software scalability is critical to support growth, but also to pivot during times of uncertainty and scale back operations as needed.

- ***Security (SE)***

  Since software often handle the critical information and infrastructure of a business, security also play a key role. Based on the type of software and its role within the business, the level of security for the software needs to documented in the requirements.

- ***Usability (US)***

  Usability is the degree to which a software can be used by specified consumers to achieve quantified objectives with effectiveness, efficiency, and satisfaction in a quantified context of use.

4. Methodology and Techniques

Since the data is unstructured raw text, the python *'Natural Language Toolkit'* or *'NLTK'* has been used to manipulate and analyse the corpus. Various text analysis techniques including *'tokenization'*, *'POS-tagging'*, *'lexicon normalization'* and *'NER'* have been used on the corpus. The overall process followed has been depicted in *Figure-2*.

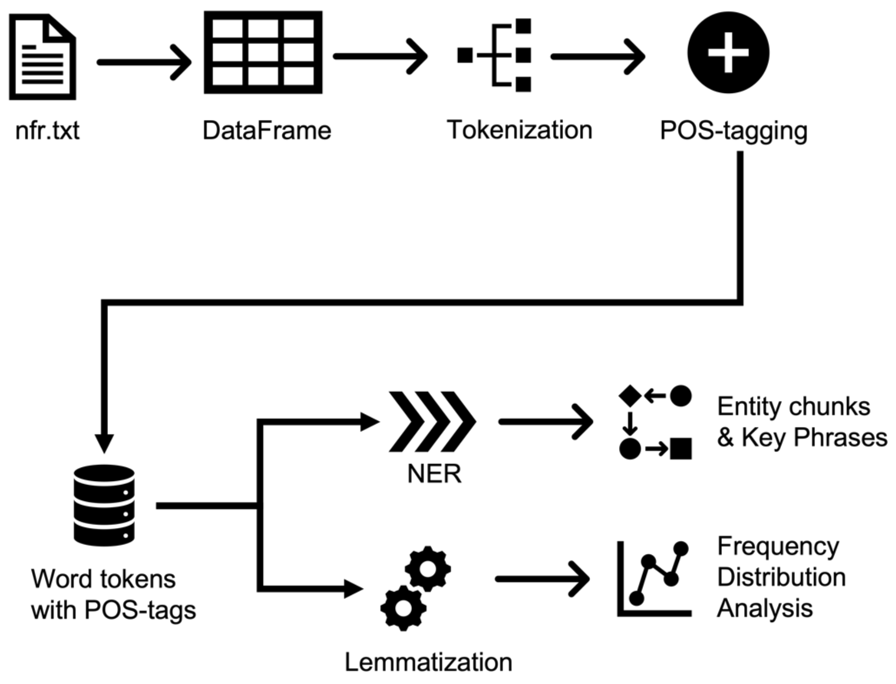

nfr.txt
DataFrame
Tokenization
POS-tagging
Word tokens with POS-tags
NER
Entity chunks & Key Phrases
Lemmatization
Frequency Distribution Analysis

Figure-2: Process *flow*

As shown in the above figure, for each requirement in the corpus, both the entity chunks and lemmatized keywords have been extracted.

4.1 Text Tokenization

The very first step when working with raw text is to split the text into smaller chunks. Since the corpus already exists as sentences, the text in the entire sentences were first converted to lower case and, then the sentences were tokenized into individual words for the analysis.

The primary purpose of considering the individual words with regards to software requirements is to identify the keywords which appear very frequently. While the exact collection of such keywords is expected to be unique for every software development project, however this can help with ensuring uniformity.

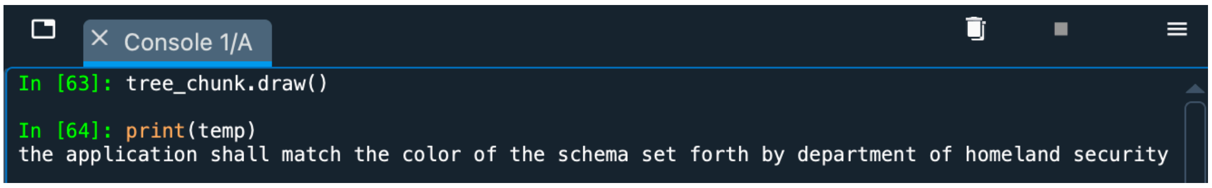
When the requirements are written in natural language, it is possible that synonyms are used instead of the actual keyword. This may require additional efforts for the developer to analyse the requirement and ensure that there is no ambiguity or missing details. Based on the analysis of the initial requirements, a set of keywords can be fixed for the project. This practise would help the tools to better assist with software requirements analysis.

4.2 Parts of Speech (POS) tagging

When we look at the software requirements written in natural language, an individual word by itself does not add any value without the 1-2 words just before or after. In order to actually identify the key phrases within the corpus, we first need the parts of speech tag for every word in the sentence. Largely the words are tagged as *nouns*, *verbs*, *pronouns*, *adverbs*, *articles*, *prepositions*, *conjunctions* or *numeric digits*.

The POS tags also provides support during the later step of lemmatization in the analysis. Use of a different form of the root word can change its meaning. For instance, the word *'application'* is derived from the root word *'apply'*. However, the word *'application'* is considered as a *noun* and not a *verb*, making the meaning different from the root word *'apply'*.

The original requirement text and its tokenized form with POS-tags for one of the requirements has been shown below in *Figure-3* and *Figure-4*.



```
In [63]: tree_chunk.draw()

In [64]: print(temp)
the application shall match the color of the schema set forth by department of homeland security
```

Figure-3: Original requirement text

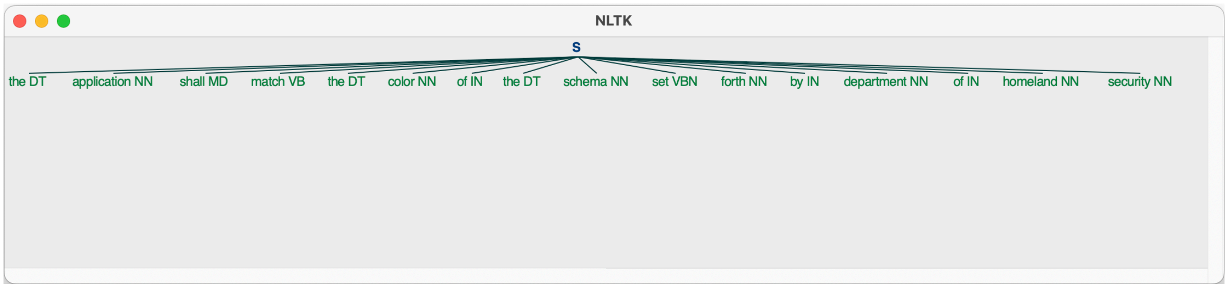


Figure-4: Tokenized form with POS-tags

4.3 Removing the punctuations and stop words

When talking about the keywords, we must understand that not all the individual words in a corpus are worthy of being considered as the keywords. As a tool, the tokenize method within python's *'nltk'* library simply splits the text into words by considering the whitespaces as delimiters. Hence the generated token include the articles, verbs and prepositions along with the in-sentence punctuations marks.

To make the analysis meaningful, we need to remove the words and punctuations since they are simply part of the sentence to fulfil the grammatical construct. This also helps to improve the efficiency as the garbage values get removed during an early stage itself. The initial tokenization of the corpus to words can help identify additional custom stop words.

4.4 Lexicon Normalization

When we write something in the natural language, there are several instances where different forms of the same word are used as per the sentence construction. While in is important to ensure that the requirements written in natural language follow the rules of grammar and are meaningful, the different forms of same word adds redundancy and inaccuracy.

Since it may not be possible to have the uniformity in the natural language, some form of *‘linguistic normalization’* is required. Under the python *‘nltk’* library, there are namely two methods to normalize the text:

a) ***Stemming***- The root words are extracted by reducing the words present in the corpus, i.e. by removing the *‘derivational affixes’*.
b) ***Lemmatization***- Using vocabulary and *‘morphological analysis’*, the words present in the corpus are reduced the their base form of the word.

While both the methods have their own pros and cons, for the software requirements corpus it was observed that the lemmatization method performed better. Here lemmatization was used to make the corpus text more uniform. Post lemmatization, all the occurrences of the base word in different forms can be identified, making the analysis is also more meaningful.

In order to perform a meaningful frequency distribution analysis and identification of the custom stop words for a corpus, lexicon normalization plays a critical role. To further improve the accuracy with lemmatization, the *‘parts of speech’* tags were also used so that the correct base word was extracted.

4.5 Named Entity Recognition

Named Entity Recognition (NER) is an advanced form of natural language processing. NER is used to identify important elements such as *places*, *people*, *organizations* and *languages* within a corpus. As mentioned earlier, the individual keywords are not always significant, hence NER techniques were used to extract entity chunks or key phrases from the corpus.

The below Figure-5, Figure shows the *‘Entity Chunks’* identified in one of the requirement text using NER.

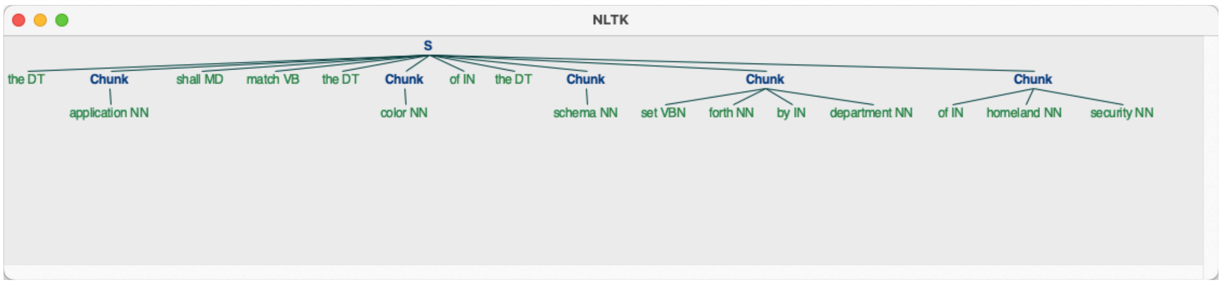
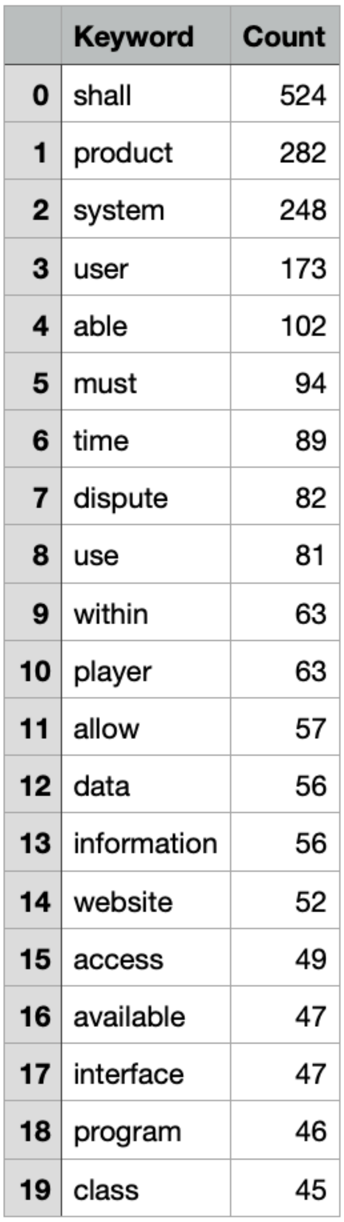


Figure-5: Entity chunks identified within requirement text using NER

5. Results

On analysing the master list of keywords post removal of stop words and lemmatization, the top 20 most frequently appearing words were extracted and the same have been shown in *Figure-6*. The keywords with their corresponding count of occurrence has been visualised using a bar chart and line graph in *Figure-7* and *Figure-8* respectively.



	Keyword	Count
0	shall	524
1	product	282
2	system	248
3	user	173
4	able	102
5	must	94
6	time	89
7	dispute	82
8	use	81
9	within	63
10	player	63
11	allow	57
12	data	56
13	information	56
14	website	52
15	access	49
16	available	47
17	interface	47
18	program	46
19	class	45

Figure-6: List of top 20 most frequently appearing words in the corpus

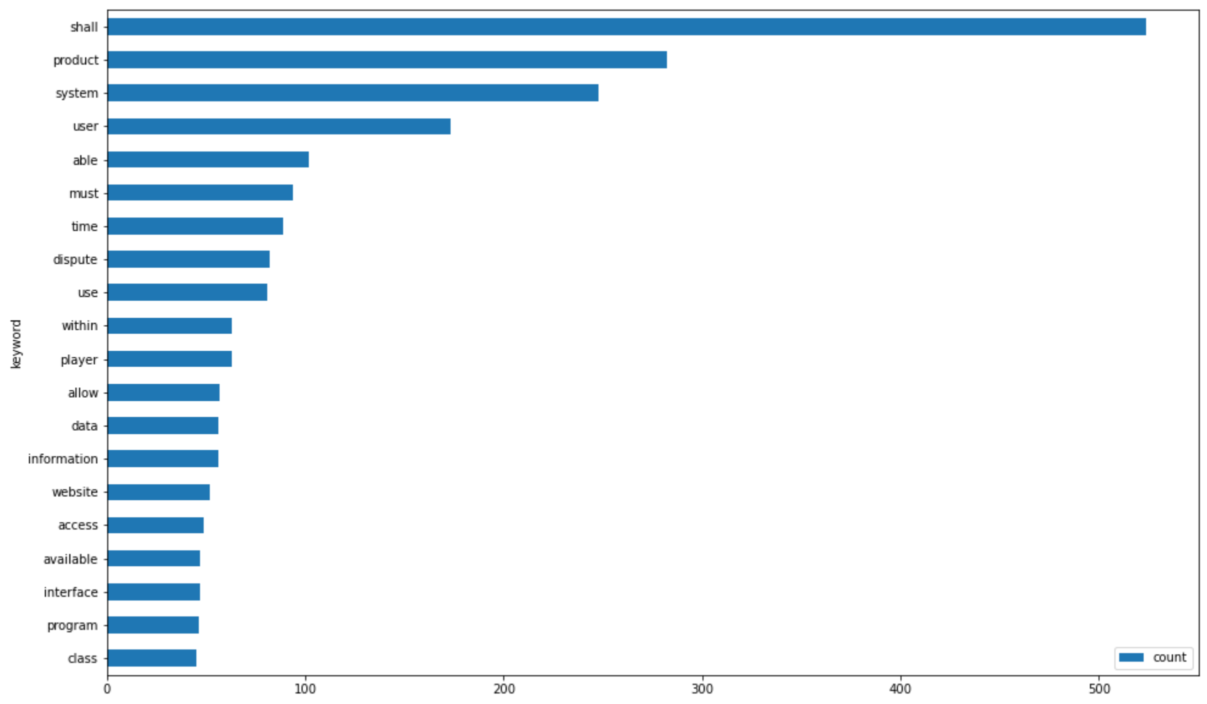


Figure-7: Bar chart showing the top 20 keywords and corresponding count of occurrences

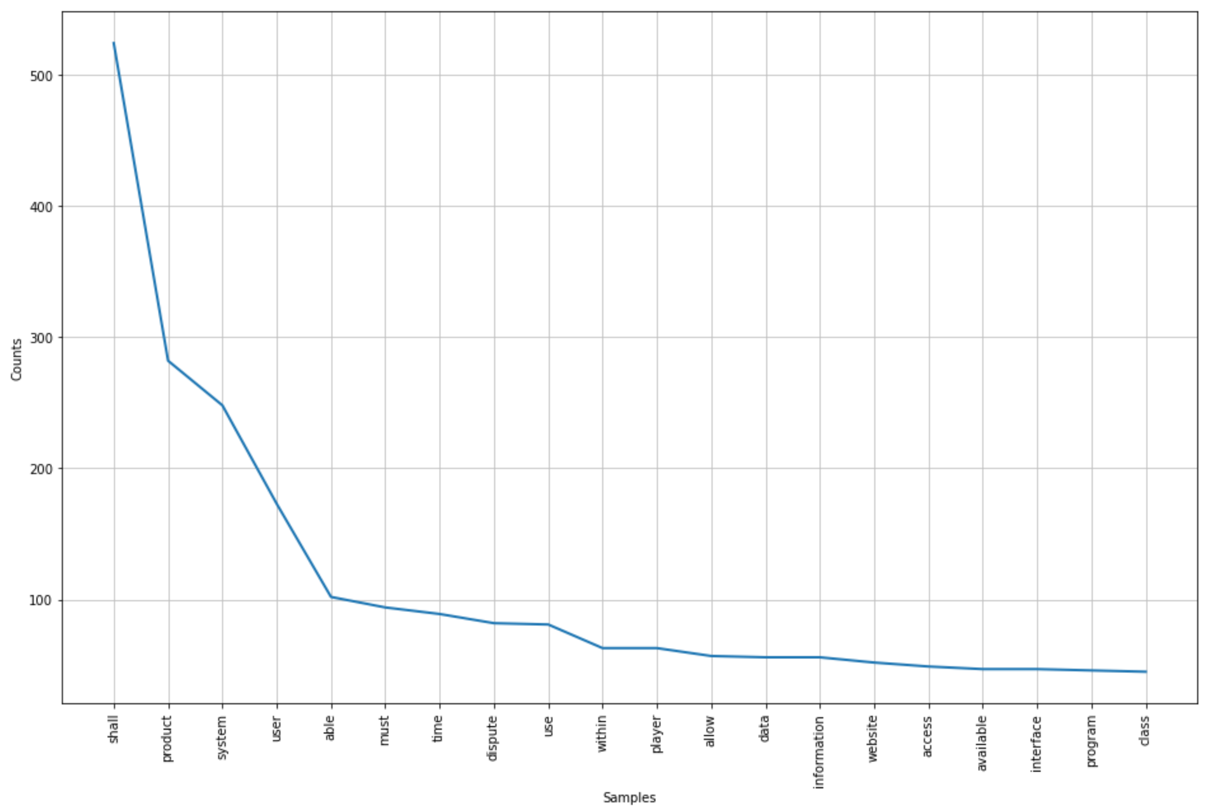


Figure-8: Line graph showing the top 20 keywords and corresponding count of occurrences

Using the master list of keywords from the corpus, an alternate visualisation in the form of a Word Cloud was also generated and the same has been shown in *Figure-9*.

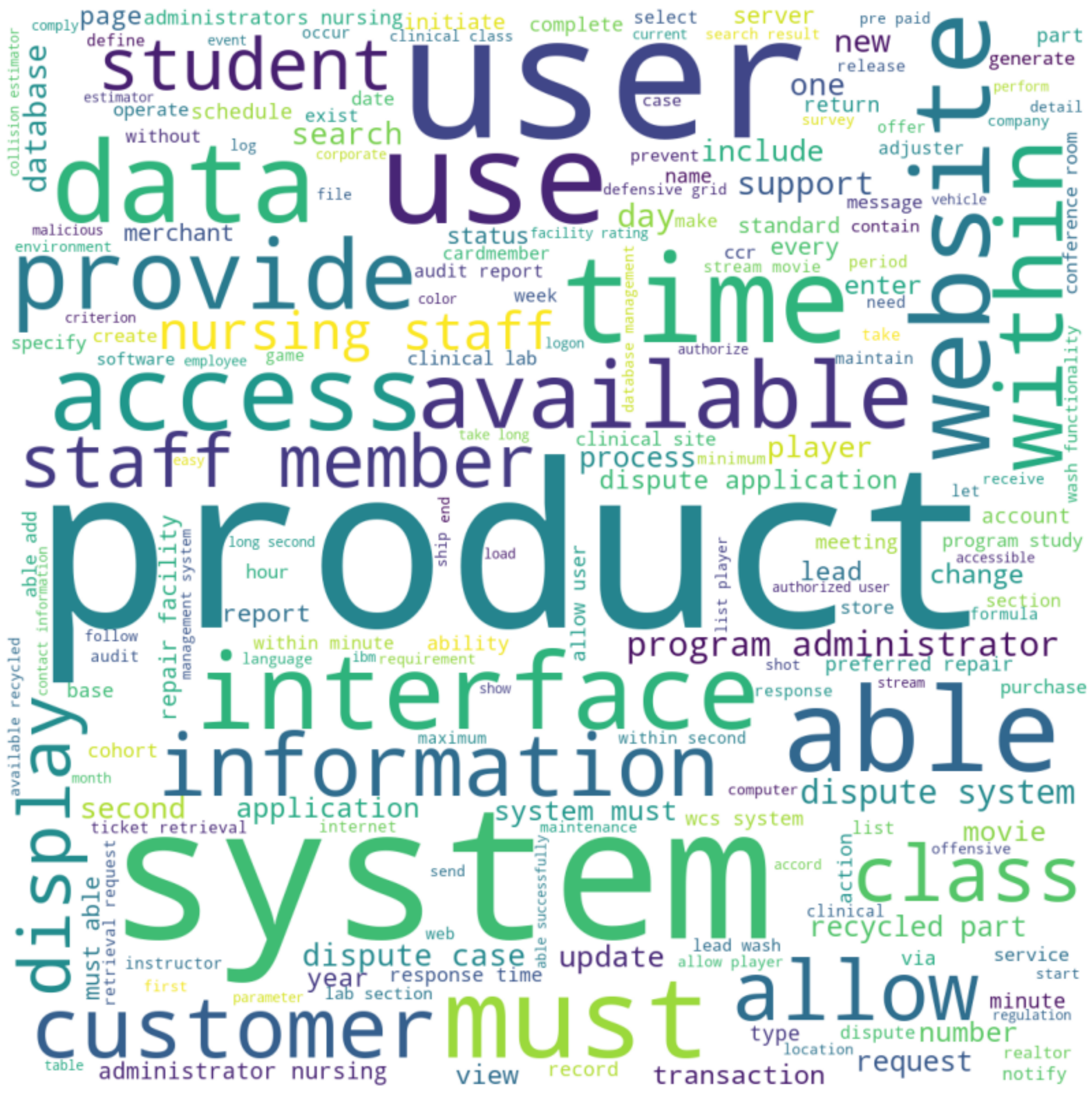


Figure-9: World Cloud visualisation for the given corpus

6. Conclusion

The processing and analysis done on the corpus clearly indicates that NLP techniques can be effective for validation of software requirements written in natural language. Even when the requirements are written in natural language, it is possible to follow certain conventions to improve the performance of the NLP tool. This would include the use of certain standard keywords and sentence structures.

Every corpus corresponding to a different software development project has to be treated as unique. The keywords need to be extracted and the custom stop words for the particular corpus need to be defined. By doing so we can better understand the requirements and accordingly define more specific grammar for extracting the entity chunks using NER.

Given the nature of software development projects, the project deadlines do not leave much time for the developers to analyse the requirements using NLP techniques. Hence, the use of standard keywords of could make this process simpler. A more diverse set of *'Context Free Grammar'* (CFG) specific to the corpus could allow the NER to better extract the entity chunks. The entity chunks are ultimately what would help to verify if the requirement stated in natural language is complete and unambiguous. More advance tools could provide some form of visualization for each requirement based on the entity chunks extracted.

Overall the use of *'Natural Language Processing'* for software requirement engineering can support the developers. At present the developers are expected to read the endless amount of requirements text written in natural language. Having tool which can extract the key phrases and generate some form of visualisation with interface for user to make corrections would be a good place to start.

References

[1] F. Dalpiaz, A. Ferrari, X. Franch and C. Palomares, "Natural Language Processing for Requirements Engineering: The Best Is Yet to Come," in IEEE Software, vol. 35, no. 5, pp. 115-119, September/October 2018, doi: 10.1109/MS.2018.3571242.

[2] Diamantopoulos, T., Roth, M., Symeonidis, A., & Klein, E. (2017). Software requirements as an application domain for natural language processing. *Language Resources and Evaluation*, *51*(2), 495–524. http://www.jstor.org/stable/26649523

[3] *Leveraging Natural Language Processing in Requirements Analysis*. (n.d.). QRA Corp. Retrieved October 31, 2022, from https://qracorp.com/leveraging-natural-language-processing-in-requirements-analysis/

[4] Ahmed, H., Hussain, A., & Baharom, F. (2018). The Role of Natural Language Processing in Requirement Engineering. *International Journal of Engineering & Technology, 7*(4.19), 168-171. doi: 10.14419/ijet.v7i4.19.22041

[5] VLAS, R. E., & ROBINSON, W. N. (2012). Two Rule-Based Natural Language Strategies for Requirements Discovery and Classification in Open Source Software Development Projects. *Journal of Management Information Systems*, *28*(4), 11–38. http://www.jstor.org/stable/41713856