A Comparative Analysis: Spelling Checker Methods for Syntactic Ambiguity Detection in Software Requirements Statements Using SMART Rules Between TextBlob and CyHunspell

Esti Mulyani, Fachrul Pralienka Bani Muhamad, Mohammad Yani and Muhamad Alfarizi
Informatic Engineering, Politeknik Negeri Indramayu, Lohbener, Indramayu, Indonesia

Keywords: Syntactic Ambiguity Detection, SMART Rules, Software Requirement Statements, Spelling Checkers.

Abstract: Software requirements tend to be presented by software analysts based on their academic background, experience limitation, and structural differences. All those things might cause errors in the software requirement creation process. These faults can lead to misinterpretation and even incorrect implementation of program code. Structural error is one of the software requirement error types. This type can potentially create ambiguity that has many meanings and confuses the reader. In previous studies, SMART rules have been proposed to detect ambiguity that may appear or be contained in the software requirements statement. The SMART rules refer to several criteria for the order of word categories (POS tags) which are considered to have no clear clue or potential to obscure meaning. Before the SMART rule is implemented, it is necessary to check the spelling of each word in a sentence to get the correct spelling. There are several methods to correct the spelling, i.e., TextBlob and CyHunSpell. In this study, an evaluation of the ambiguity detection performance of software requirements statements using SMART rules was carried out by comparing two spell-checking methods, TextBlob and CyHunspell. The test scenario in this study was carried out by taking twenty-five syntactically ambiguous and unambiguous statements from the PURE dataset. Determination of ambiguity of statements in this scenario is annotated manually based on matching SMART rules. The experiment demonstrates that CyHunspell has a better performance than TextBlob.

1 INTRODUCTION

The Software Requirements Statement is the initial stage in the software development process, describing the needs needed by customers and software developers and how the system should run (Nurfauziah, 2017). Software Requirements are also needed in the production of Software Requirements Specification (SRS) documents. Therefore, the statement of software requirements is very important in making the SRS document so that the quality of the resulting document will provide results that affect the software development process in the future.

In requirements engineering, the process of searching and searching for requirements is a complicated thing to do. There is always a misinterpretation of needs between customers and developers (Enda, 2018). Errors that often arise are the use of words that are ambiguous, unclear, inconsistent, spelling errors, and incompleteness. This can lead to misunderstandings or interpretations that will be difficult for those involved in the development process. If ambiguous statements about software requirements are not detected and corrected immediately in the early stages of development, misinterpretation will cause problems in the future. Therefore, it is necessary to deal with ambiguity issues to avoid negative impacts such as repair costs, lower software quality, delayed software release times, and software failure can be avoided. Therefore, a method is needed to identify the use of potentially ambiguous words in the statement of software requirements. There exist some types of ambiguity in software requirement statements, namely, syntactic, lexical, and pragmatic ambiguity. However, this paper only focuses on discussing the proposed approaches used to detect sintactic ambiguity in the statement of software requirements. To achieve this goal, a review of previous research has been carried out which has discussed several methods to detect ambiguity in software requirements statements. Note that, in the experiment section, we only use English as the input of both TextBlob and CyHunspell.

Mulyani, E., Muhamad, F., Yani, M. and Alfarizi, M.
A Comparative Analysis: Spelling Checker Methods for Syntactic Ambiguity Detection in Software Requirements Statements Using SMART Rules Between TextBlob and CyHunspell.
DOI: 10.5220/0011723600003575
In *Proceedings of the 5th International Conference on Applied Science and Technology on Engineering Science (iCAST-ES 2022)*, pages 120-126
ISBN: 978-989-758-619-4; ISSN: 2975-8246

The rest of this paper is structured as follows. In Section 1, we introduce the issue of ambiguity in software requirement statements. In Section 2, we brief related articles to this topic. Section 3 presents the result and analysis of our experiment on CyHunspell and TextBlob. Finally, Section 4 concludes the result of experiments.

2 LITERATURE REVIEW

2.1 Related Reviews

Many studies have been conducted regarding the ambiguity in software requirements. To detect ambiguity in the statement of software requirements, a literature study was carried out as a tool for applying research methods. The following is a literature study on the approach used to detect ambiguous or ambiguous requirements statements in software requirements specification documents to provide recommendations for improving grammar in English:

1. This research was conducted by Erik Kamsties, Daniel M. Berry, and Barbara Paech in 2001 entitled “Detecting Ambiguities in Requirements Documents Using Inspections”. The Study uses an approach with inspections carried out manually in detecting ambiguity. In this study, the approach used is still ineffective because the needs analysis is done manually and relies heavily on subjectivity and the ability of an expert to analyze ambiguity.
2. Research by Hussain, H. M in 2007 entitled "Using text classification to automate ambiguity detection in SRS documents". The study attempts to detect ambiguity by utilizing a specially created repository for listing unwanted words. Focusing on detecting ambiguity that is done automatically but has drawbacks because the repository used is still static and the scope of the problem is also limited, this will certainly affect the performance of the method performance.

2.2 Software Requirements

Software requirements are specific attributes that are specifications of functional requirements and non-functional requirements of a software system (Enda, 2018). Based on this statement, software requirements are conditions, criteria, requirements, or capabilities that must be possessed by software to meet what is required or desired by the user. Software that is good and follows the needs of users is very dependent on the success of conducting a needs analysis. When identifying software requirements, the information obtained has not been structured. The user will express what he needs in everyday language that is used by the user.

2.3 SRS Documents

Software Requirements Statements that are functional and non-functional are written in the software requirements specification or SRS documents. The SRS document describes a set of services required by users and used by system developers. SRS can be written in several methods, namely natural language, structured natural language, semi-formal language, and formal language. Natural language is easier for stakeholders to understand. Natural language is more advantageous because it is more flexible to accommodate the description of changing needs. However, it also has a weakness, which is easy to confuse and difficult to analyze automatically (Nurfauziah, 2016).

2.4 Ambiguity of Software Requirements Statement

The chain from a human error to fault to failure is not unbroken and inevitable. Generally, developers either detect and correct faults without investigating the underlying errors or use software testing to reveal system failures that they then repair (Anu., et al., 2018). SRS Documents must be of good quality to avoid possible damage or failure of the software in the planning and testing stages. Software quality at the requirements engineering stage (Carlson, et al., 2014). There are several types of ambiguity in software requirements, lexical ambiguity, syntactic ambiguity, semantic ambiguity, pragmatic ambiguity, and generality.

2.5 Identification Based on SMART Rules

Software requirements can be said to be of quality if they are Specific, Measurable, Attainable, Realisable, and Traceable (SMART). This SMART rule is focused on a Specific pattern of rules, namely the recommended guideline is "Avoid ambiguities such as some, several, many" (Mannion et al., 1995). Based on the statement above, the SMART rules that are built are:

1. If there is some/JJ + NNS pattern, then the pattern is categorized as ambiguous.

2. If there is several/JJ + NNS patterns, then the pattern is categorized as ambiguous.
3. If there is many/JJ + NNS patterns, then the pattern is categorized as ambiguous.

This method works based on a set of rules in the form of a phrase pattern for each part of the word or POS (Part of Speech) along with ambiguous words. The advantage of this method is that the pattern matching technique used can be faster and more efficient in detecting ambiguous words by comparing the patterns of the rules that have been made.

2.6 Text Preprocessing

Text Preprocessing is an important task and step in Text Mining, Natural language Processing (NLP), and Information Retrieval (IR) (Hermawan and Ismiati, 2020). In the field of text mining, data preprocessing is used to extract important results from unstructured data. Meanwhile, information retrieval is used to decide which documents in the collection should be retrieved to meet the user's need for information.

Familiar text processing such as tokenization, case folding, stopword removal, and stemming (Rahimi., et al., 2020). Below is an explanation of each stage of text preprocessing.

- Remove Punctuation
  Remove punctuation is a step of text preprocessing that is used to remove unnecessary punctuation marks or symbols. Punctuation marks or symbols are removed because they do not affect the text preprocessing process. The decision of whether to include or remove punctuation is the first preprocessing choice (Denny and Spirling, 2018).
- Case Folding
  The initial stage of text preprocessing is the case folding stage. Case folding is a method for turning all the letters in a dataset into capital or all small (Mustaqim., et al., 2020)
- Tokenization
  The tokenization stage is the stage of separating or cutting data in the form of phrases, clauses, or sentences into words. This process segments large texts into sentences that are then tokenized into words (Hassani., et al., 2021).
- Filtering
  After the tokenization stage is filtering. Filtering is used to retrieve important words from the tokenization stage (Ratniasih., et al., 2017). Words that have no meaning are called stopwords, like conjunctions and, and, after, and so on. Eliminating stopwords is very useful in the text preprocessing process because it can speed up processing which means maximizing processing time.

2.7 Part of Speech Tagging

Part of speech (POS) tagging is the process of determining the correct POS of each word in the text (Thavareesan and Mahesan, 2020). In language processing, one of the concepts that need to be understood to process and use good and correct grammar is to determine the word class. Word class or word type is a grouping of words to find a system in the language. Words are complex forms composed of several elements. Word classes are divided into five categories based on syntactic, function, and meaning categories, namely nouns, verbs, adjectives, adverbs, and task words. Methodology

2.8 Existing Methods

At this research stage, an auxiliary application will be built for ambiguity detection on software requirements. Application development is done by developing the SMART rule method to detect ambiguous words in the statement of software requirements. The solution flow proposed in this study can be seen in the flowchart of the proposed method in figure 1.

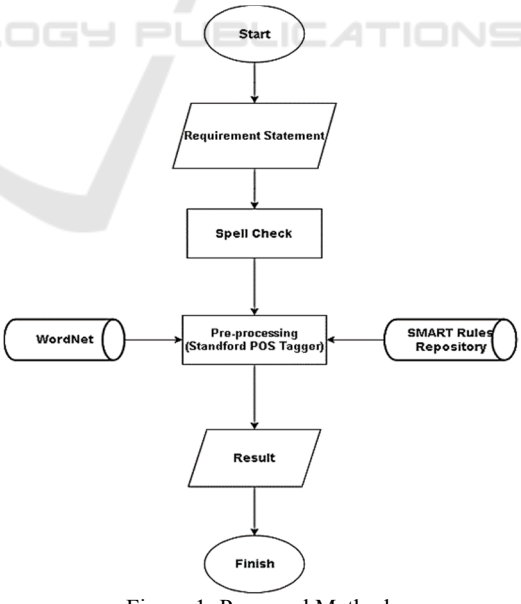

Figure 1: Proposed Method.

The following is an explanation of the components of the proposed method flowchart from Figure 1 as follows:

- Spell Check

Spell checker is a technique of checking every spelling in a sentence, using tools to check the spelling of a sentence namely Textblob and Cyhunspell. There is a method to retrieve the text or sentence to check. The method used by Textblob is spellcheck() whose output will display word recommendations that are written correctly and Cyhunspell uses the suggest() method to display word recommendations that are written correctly.

- Preprocessing
  Preprocessing is the process of converting raw data into a form that is easier to understand. This process is important because raw data often do not have a regular format. In addition, data mining also cannot process raw data, so this process is very important to do to simplify the next process.
- SMART Rules Repository
  SMART Rules Repository is storage containing ambiguous words and part of speech of the words that will be stored in a table which will be used in the SMART module. The repository will be used by the SMART module in the process of selecting the correct rules to analyze the ambiguity of the requirements stated.
- WordNet
  WordNet is used to recommend synonyms for ambiguous words in the SMART rule table. Synonym searches are only performed on words that have the same POS as the word being checked. If synonyms are found, then the sentence will be declared ambiguous.
- SMART Module
  The SMART module is used to analyze the ambiguity of each requirement statement that has been processed from the previous steps. The analysis process is assisted by other components, namely the SMART rule repository and WordNet.

2.9 Method Development

2.9.1 Requirements Statement Collection

The Collection of statements of software requirements from the Public Requirements Dataset (PURE) is a public dataset. The data set contains 79 documents and includes 34,268 sentences that can be used for natural language processing tasks. PURE can be used as a benchmark in tasks, such as detection of ambiguity, categorization of requirements, and identification of equivalence requirements.

2.9.2 Spell Check Process

Drafting documents is an implied part of most positions of employment. Most employees lose command over their writing skills. This disengagement from writing, can lead to grammatical flaws and/or spelling errors and also lead to the use of stagnant vocabulary (Pisat, et al., 2020).

The initial process before entering the preprocessing process is a spell check first. In this process, the sentence will be checked for spelling errors. If there is a spelling error, it will result in where the spelling is wrong and will be recommended the correct spelling of the wrong word. If no errors are detected in the spelling of each word, it will be used as input for the next process.

2.9.3 Preprocessing

After the spell check process is done, the next step is pre-processing. In pre-processing, it is used to prepare text before it is used in testing with the aim of parsing noise in the text or data so that it can improve performance. The pre-processing techniques used can be seen in the points below:

a. Remove Punctuation
   In this stage, the text or data is carried out in the process of removing punctuation. This will remove symbols or punctuation that have no effect, such as punctuation marks, exclamation points, and question marks. This process is done at the beginning to get the maximum text or data.
b. Case Folding
   In this process, uppercase or capital letters will be changed to lowercase letters (lowercase), and only letters a to z are accepted in the case folding process. Characters other than the letters 'a' to 'z' (punctuation marks and numbers) will be removed from the data.
c. Tokenization
   The tokenization process will break the sentence into words or tokens. Tokenization uses a library from the Natural Language Toolkit (NLTK), namely word_tokenize.

2.9.4 POS Tagging

Part of speech is commonly known as the type of word in a sentence such as a verb, adjective, or noun. Part of Speech tagging is a process of marking the word class or part of speech on each word in a sentence. Stanford POS Tagger is used to give a tagset or a mark on every word that has gone through pre-processing. Stanford POS Tagger is a tool for reading text and determining the language word class

of each word, such as nouns, verbs, adjectives, etc. The library used in this stage comes from NLTK by importing StanfordTagger.

2.9.5 SMART Rules Repository Creation

Before being used by the SMART module, it is necessary to create a SMART rule repository that contains words that contain more or have multiple meanings. The words are stored and then matched with the word class or POS of the word and POS2 shows the POS pairs that make a sentence ambiguous. As in the following example, the multiple with POS is JJ and the pair POS2 request is NNS, so the sentence is ambiguous.

There are 28 of words identified as ambiguous with POS and POS2 which have the possibility of ambiguity. Below is a sample of some ambiguous words with POS and POS2 which can be seen in Table 1

Table 1: Sample SMART Rules Repository.

ID	Word	POS	POS2
1.	Several	JJ	NNS
2.	Many	JJ	NNS
3.	Some	JJ	NNS
4.	Obviously	RB	JJ
5.	Who	WP	MD

2.9.6 SMART Module

After the previous stages have been completed, the next step is to build the main module whose task is to analyze the ambiguity of each statement of software requirements that has been processed from the initial stage. At this stage using syntactic ambiguity which uses rules based on matching between POS and POS2. The analysis process is assisted by a previously created SMART rules repository containing words identified as having the possibility of ambiguity along with the POS and POS2 of the words in the SMART rule repository. In this SMART module, matching is done based on words that have been tagged in the POS tagging process using a Stanford POS tagger, then the analysis process begins by matching each word in the sentence with ambiguous words in the rule dictionary. If there is a match between one word and the word stored in the rule dictionary, then the sentence is considered ambiguous.

3 RESULT AND ANALYSIS

3.1 System Implementation

This research consists of three parts, checking the spelling of the word, the class of the word in the sentence, and seeing whether the sentence contains ambiguous words. The system implementation process for building software will be explained in this section.

3.1.1 Obtain Software Requirement Statements

The collection of software requirements statements was obtained from the Public Requirements Dataset. 50 software requirements statements are used to match SMART rules. Table 2 is a sample obtained to be used as a statement of software requirements.

Table 2: Software Requirement Statements.

No.	Statement
1.	This database will be built for a particular system and may not be portable but results to queries will be portable between many environments.
2.	A bulk entry can be used to add many assets
3.	The RLCS shall support multiple users logged on, up to the limit of the number of users defined in the database.
4.	The user interfaces should be designed to make them user intuitive.
5.	The system should be available 24 hours a day, 7 days a week.

3.1.2 Initial Stage

Statements that have been obtained previously will be checked first against the writing of words or the spelling of words in the statement sentences that are entered for processing to the next process. The initial stage is to check the sentence whether there are words that are wrong in writing or spelling can be seen in Figure 2.

```
# Spell Checker
for word in pernyataan.words:
  print(word,":",word.spellcheck())
```

```
This : [('His', 0.7117826487905228), ('This', 0.2882173512094772)]
database : [('database', 1.0)]
will : [('will', 1.0)]
be : [('be', 1.0)]
built : [('built', 1.0)]
for : [('for', 1.0)]
a : [('a', 1.0)]
particular : [('particular', 1.0)]
```

Figure 2: Spell Check.

3.1.3 Pre-Process Stage

If the results of the spell checker produce perfect sentences, then the pre-processing stage is carried out first so that the data is cleaner and avoids punctuation that can block the next process. The pre-processing stages consist of case folding, remove punctuation, and tokenizing. The process of case folding and remove punctuation become one method in the program code. The lower() method is used to generalize the first letter of a word to lowercase (a.k.a. text normalization). Meanwhile, re.sub("[^a-zA-Z]") is used so that the system only accepts input in the form of letters of the alphabet, which can be seen in Figure 3.

```
letters_only = re.sub("[^a-zA-Z]",
                      " ",
                      str(pernyataan).lower())
letters_only
```

Figure 3: Case Folding and Remove Punctuation.

The next pre-process stage is tokenization. Tokenization uses a library from NLTK with the type word_tokenize. Tokenization will generate tokens from a statement of needs in the form of a sentence. Previously a complete statement sentence became a separate word, which can be seen in Figure 4.

```
textblob_kalimat_tokenize = nltk.word_tokenize(letters_only)
textblob_kalimat_tokenize
```

['this', 'database', 'will', 'be', 'build', 'for', 'a', 'particular', 'system', 'and', 'may', 'not', 'be', 'notrable', 'but', 'results', 'to', 'queries', 'will', 'be', 'portable', 'between', 'many', 'environments']

Figure 4: Tokenization.

3.1.4 POS Tagging Stage

After the pre-processing stage is complete, the next step is the POS tagging stage, which is to give a word class to each word in the sentence that is entered in the process of collecting software requirements statements. This stage uses a Stanford POS Tagger library provided by NLTK to assign a word class to each word. When executing the pos_tag() method, the tokenized sentence will be assigned a word class for each word that has become a token. The implementation of POS tagging can be seen in Figure 5.

```
# POS TAGGING

pos_tagged = nltk.pos_tag(kalimat_tokenize)
print(pos_tagged)

for word, word_class in pos_tagged:
    print(word + "_" + word_class)
```

```
this_DT
database_NN
will_MD
be_VB
built_VBN
for_IN
a_DT
particular_JJ
system_NN
```

Figure 5: POS Tagging Stage.

3.1.5 Checking Sentence with SMART Module

After getting the word class on each word that has been tagged POS. Furthermore, it can check whether the entered sentence contains ambiguous words. Matching is done based on words that have been tagged in the POS tagging process using a Stanford POS tagger, then the analysis process begins by matching each word in the sentence with ambiguous words in the rule dictionary. If there is a match between one of the words with the word stored in the rule dictionary, then the sentence is considered ambiguous. The results of the analysis are in the form of a conclusion if the statement is ambiguous or unambiguous. At this stage using the help of list(nltk.bigrams()) to be able to match each word class with two adjacent items can be seen in Figure 6.

```
['many JJ -> environments  NNS | Recommendation many : two, three, four']
```

Figure 6: Result Checking Sentence.

3.2 Match Results

This section is the result of matching with the SMART rule repository which contains words that may contain multiple meanings matched with word classes that have been tagged in the POS tagging process. If there is a word match in the input sentence with a word in the SMART rule repository, then the sentence is declared ambiguous. Table 3 is the result of matching the spell check method.

Table 3: Match with SMART Rules Repository.

Spell Check Methods	Total Ambiguous Word Detect
Textblob	15
Cyhunspell	25

CyHunspell provides recommendations for correcting words that have been registered in the SMART repository, so that detection of ambiguous sentence structures is better than TextBlob.

4 CONCLUSIONS

In this research, a comparative analysis of the spell checker method has been carried out to be integrated with SMART rules. Based on the results of the study, it was found that CyHunspell gave better detection results than TextBlob. Detection of ambiguous statements with the SMART Requirements approach which refers to specific criteria is a suitable criterion because it explains that it is necessary to avoid words that contain ambiguity, for example some, several, and many with pairs of word classes being nouns. If there is a word match in the input sentence with a word in the SMART rule repository, then the sentence is declared ambiguous.

A spell check is performed at the beginning before performing the rule-matching step. The statement sentence will be checked first against the writing of the word or the spelling of the word in the statement sentence that is entered for processing to the next process. Pre-processing is an important stage because if you don't do pre-processing, then the data is declared unclean.

REFERENCES

Anu, V., Hu, W., Carver, J. C., Walia, G. S., & Bradshaw, G. (2018). Development of a human error taxonomy for software requirements: A systematic literature review. *Information and Software Technology*, *103*, 112-124.

Carlson, N., & Laplante, P. (2014). The NASA automated requirements measurement tool: a reconstruction. *Innovations in Systems and Software Engineering*, *10*(2), 77-91.essbout *Analy*it. *Politicalsis*, *26*(2), 168-189.

Denny, M. J., & Spirling, A. (2018). Text preprocing for unsupervised learning: Why it matters, when it misleads, and what to do a

Enda, D. (2018). *Rekomendasi perbaikan pernyataan kebutuhan yang rancu dalam spesifikasi kebutuhan perangkat lunak menggunakan teknik berbasis aturan* (Doctoral dissertation, Institut Teknologi Sepuluh Nopember).

Hassani, A., Iranmanesh, A., & Mansouri, N. (2021). Text mining using nonnegative matrix factorization and latent semantic analysis. *Neural Computing and Applications*, *33*(20), 13745-13766.

Hermawan, L., & Ismiati, M. B. (2020). Pembelajaran text preprocessing berbasis simulator untuk mata kuliah information retrieval. *Jurnal Transformatika*, *17*(2), 188-199.

Hussain, H. M. (2007). *Using text classification to automate ambiguity detection in SRS documents* (Doctoral dissertation, Concordia University).

Kamsties, E., Berry, D. M., & Paech, B. (2001). Detecting ambiguities in requirements documents using inspections. In *Proceedings of the first workshop on inspection in software engineering (WISE'01)* (Vol. 13).

Mannion, M., & Keepence, B. (1995). SMART requirements. *ACM SIGSOFT Software Engineering Notes*, *20*(2), 42-47.

Mustaqim, T., Umam, K., & Muslim, M. A. (2020). Twitter text mining for sentiment analysis on government's response to forest fires with vader lexicon polarity detection and k-nearest neighbor algorithm. In *Journal of Physics: Conference Series* (Vol. 1567, No. 3, p. 032024). IOP Publishing.

Nurfauziah, S. (2016). *Pendeteksian Ketidaklengkapan Kebutuhan Dengan Teknik Klasifikasi Pada Dokumen Spesifikasi Kebutuhan Perangkat Lunak* (Doctoral dissertation, Institut Teknologi Sepuluh Nopember).

Pisat, T., Bartakke, M., & Patil, H. (2020). Synonym Suggestion System using Word Embeddings. In *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184)* (pp. 505-508). IEEE.

Rahimi, N., Eassa, F., & Elrefaei, L. (2020). *An ensemble machine learning technique for functional requirement classification*. *symmetry*, *12*(10), 1601.

Ratniasih, N. L., Sudarma, M., & Gunantara, N. (2017). Penerapan text mining dalam spam filtering untuk APLIKASI chat. *Majalah Ilmiah Teknologi Elektro*, *16*(3), 13. https://doi.org/10.24843/mite.2017.v16i03p03

Thavareesan, S., & Mahesan, S. (2020). Word embedding-based Part of Speech tagging in Tamil texts. In *2020 IEEE 15th International Conference on Industrial and Information Systems (ICIIS)* (pp. 478-482). IEEE.