Research Article

AmbiDetect: An Ambiguous Software Requirements Specification Detection Tool

Osman, M.H. [1], Zahrin, M.F[2]

[1]Faculty of Computer Science and Information Technology, University Putra Malaysia, Malaysia
[2]Ministry of Education, Malaysia
hafeez@upm.edu.my[1]

Article History: Received: 10 November 2020; Revised: 12 January 2021; Accepted: 27 January 2021; Published online: 05 April 2021

Abstract: Software Requirements Specification (SRS) is considered a highly critical artifact in the software development. All phases in software development are influenced by this artifact. Defects in software requirements may higher the risk of project overschedule that contributes to cost overrun of the project. Researchers have shown that finding defects in the initial software development phase is important because the cost of the bug is cheaper if it is fixed early. Hence, our main goal is to provide a platform for requirement engineers to produce better requirement specifications. We propose AmbiDetect, a (prototype) tool to automatically classify ambiguous software requirements. AmbiDetect combines text mining and machine learning for ambiguous requirement specification detection. The text mining technique is used to extract classification features as well as generating the training set. AmbiDetect uses a machine learning technique to perform the ambiguous requirement specification detection. From an initial user study to validate the tool, the result indicates that the accuracy of detection is reasonably acceptable. Although AmbiDetect is an early experimental tool, we optimist that this tool can be a platform to improve SRS quality.

Keywords: Requirement Engineering, Text Mining, Machine Learning

1. Introduction

Software requirements specification (SRS) is the foundation of software and considered a highly critical document in software development. It records all stakeholders' requests of the system to be developed. This document is normally produced in the early phase of software development and influences all subsequent phases in the software development lifecycle. Therefore, we strongly believe that a high-quality software requirement increases the quality of software. It is similar to the term "*Garbage in, Garbage out*" that has been commonly used in computer programming which denotes "*If there is a logical error in software, or incorrect data are entered, the result will probably be either a wrong answer or a system crash*", Dictionary.com, 2018. For instance, if we look at the V-Model, high-level testing is often depicted as the Verification and Validation activity to requirements elicitation, analysis and specification. This means the relationship between requirements engineering and testing is an important part of our software engineering knowledge, Unterkalmsteiner et al. (2014). A key characteristic of software requirements is that it should be possible to verify that the finished product meets the requirements, Bourque and Fairley (2014).

{*Problem statement*}. Incomplete, unverifiable, incorrect, and ambiguous are several examples of issues in software requirements that influence the acceptance testing of software. This may lead to overruns in project time and costs, Fernández and Wagner, (2015). The cost of bugs that found late is more expensive than if it found early. Hence, software defects need to be detected as early as possible, Berry et al. (2006). A method that provides a quick detection to several defects in software requirement could give useful feedback to the requirement engineer, Femmer et al., (2014), Hussain et al., (2017).

As recommended by ISO/IEC/IEEE 29148, ISO (2011), an individual software requirement should have the following characteristics: (i) unambiguous, (ii) consistent, (iii) complete, (iv) singular, (v) feasible, (vi) traceable and (vii) verifiable. To this end, most of the work used to classify defects in the requirements specification refers to several requirements templates, such as Pohl and Rupp (2011), Mudavath et al. (2020), Mavin et al. (2009), IEEE (1998) and ISO (2011). Based on our observations and experiences, not all requirements specifications follow the requirements template, but they can still be understood or the quality is acceptable. It is necessary to have a technique for classifying defects based on the truly unstructured text in requirements specifications.

{Goal}. Our main goal is to provide a platform for requirement engineers to produce better SRSs. As an initial step, we propose AmbiDetect, a (prototype) tool to automatically detect ambiguous software requirements specification. AmbiDetect combines supervised machine learning techniques and text mining to provide defect classification model. The text mining technique is used for feature extraction, and the tools 'learns' to detect the ambiguous SRS by using the supervised machine learning technique. We apply the ambiguous SRS detection proposed by Osman and Zaharin (2018) for developing the classification model.

To this end, the tool capable to perform ambiguity detection only for SRS that is written in Malay (language) because the generated dataset is originated from Malaysian industrial software developments which are only written in Malay. However, the tool is developed in such a way that it can use any dataset from any language. Even though this tool is still a prototype, we received good feedback after conducting an initial user study. Hence, we optimist that the ambiguous SRS detection technique and tool may contribute to enhance the quality of SRS. The tool's demonstration can be found at Osman (2018).

2. System Overview

The development of AmbiDetect is based on the architecture in **Figure 1**. We applied a simple two-tier architecture that is easy to scale for future expansion. The description of two-tier architecture is (i) First Tier: Contain web browser which covers front end-users to provide a graphical user interface and interface windows; and (ii) Second Tier: Consist of Web Server (Apache) and Application Server (PHP) that execute (business) logic of this tool.

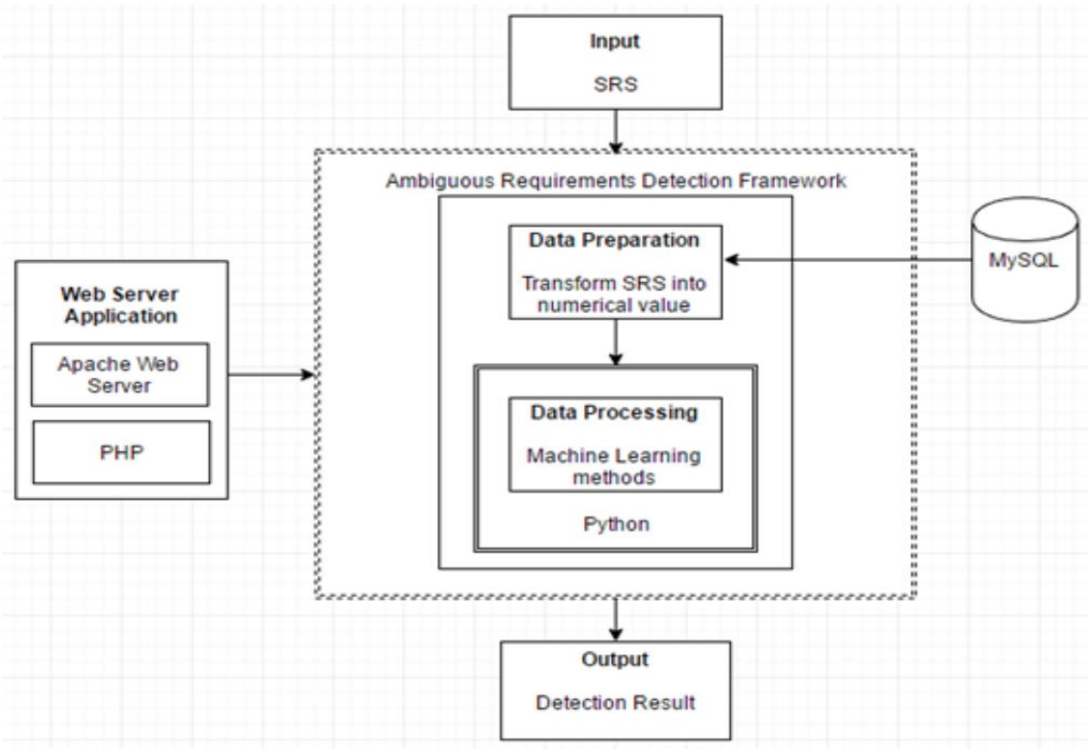

Figure 1. AmbiDetect System Architecture

The description of several elements in the system architecture is the following:
a) Input - The user interface to collect the SRS statement from the user.
b) Data Preparation - Responsible to (i) chunk the text input (i.e. SRS statement) into words, (ii) map each word with the word that is stored in the MySQL database (derived from training data), (iii) calculate the word occurrence and (iv) transform SRS into data for classification.
c) Data Processing - Execute the machine learning component to classify the SRS. The SRS is classified as ambiguous if the classification value = 1 and classified as unambiguous if the classification value = 0. Python scikit-learn library is used for machine learning classification purposes.
d) Output - The user interface to present the output (i.e. ambiguous/unambiguous and the words that possibly contribute the classification)

3. AmbiDetect Features

There are three (3) main features of the AmbiDetect tool: (i) Information of Software Requirement & Dataset, (ii) Requirement Validation, and (iii) Classification Features.

A. Information on Requirement Specification & Dataset

As mentioned in section 1, the main purpose of our work is to provide a platform for software engineers to

produce a better SRS. Hence, in this tool, we present the following information at the tool's landing page (as shown in **Figure 2**):

- *How to construct a good requirement?* This information shows the step of constructing software requirements suggested by the International Requirement Engineering Board (IREB).
- *How to avoid ambiguity?* The tool describes the information on how a user can avoid ambiguity. This information is gathered from ISO/IEC/IEEE 29148.
- *List of the ambiguous word.* The information about the possible ambiguous words is derived from Haron et al. (2015) and Kamsties et al. (2001).
- *Top words in the dataset.* The tool lists (top) ten words that are extracted from the training data.

B. Ambiguous Requirement Detection

The ambiguous requirement detection is the primary feature of this tool. This feature detects the ambiguity of an individual requirement specification that is entered by the user (**Figure 3**). The tool shows the result (i.e. ambiguous or non-ambiguous requirement specification) and also lists the words are possibly influence the classification.

C. Configure Classification Features

This feature allows the user to manually configure the classification features (i.e. the feature-words) for classifying ambiguous and non-ambiguous requirements. The tool listed all the feature-words that are used for machine learning classification (**Figure 4**). In this way, the user may exclude the words that are not influential in classifying the requirements.

4. Building The Classification Model

The classification model was built based on the framework illustrated in **Figure 5**. The explanation of the phases in building the classification model are the following:

A. Data Collection

We collected four (4) SRS documents that derived from four (4) separated Malaysian Industrial Software Development Projects for this tool. In total, 180 software requirements specifications were extracted from the SRS documents to generate the dataset.

B. Document Preprocessing

Supervised machine learning needs labeled data. In this phase, we conduct a data labeling task and perform data cleansing.

- Data Labeling. This tool aims at providing the information of ambiguous and unambiguous requirements based on the classification model. The training dan testing dataset should consist of labeled data which is ambiguous 'Y' and Unambiguous 'N'. Each of the extracted requirements is manually labeled. Two (2) requirement engineers performed the labeling activity. Both engineers have more than three (3) years in the field of requirements engineering. As a reference, the following definition of unambiguous (defined by the ISO/IEC/IEEE 29148) is used for this task: "*The requirement is stated in such a way so that it can be interpreted in only one way. The requirement is stated simply and is easy to understand*".

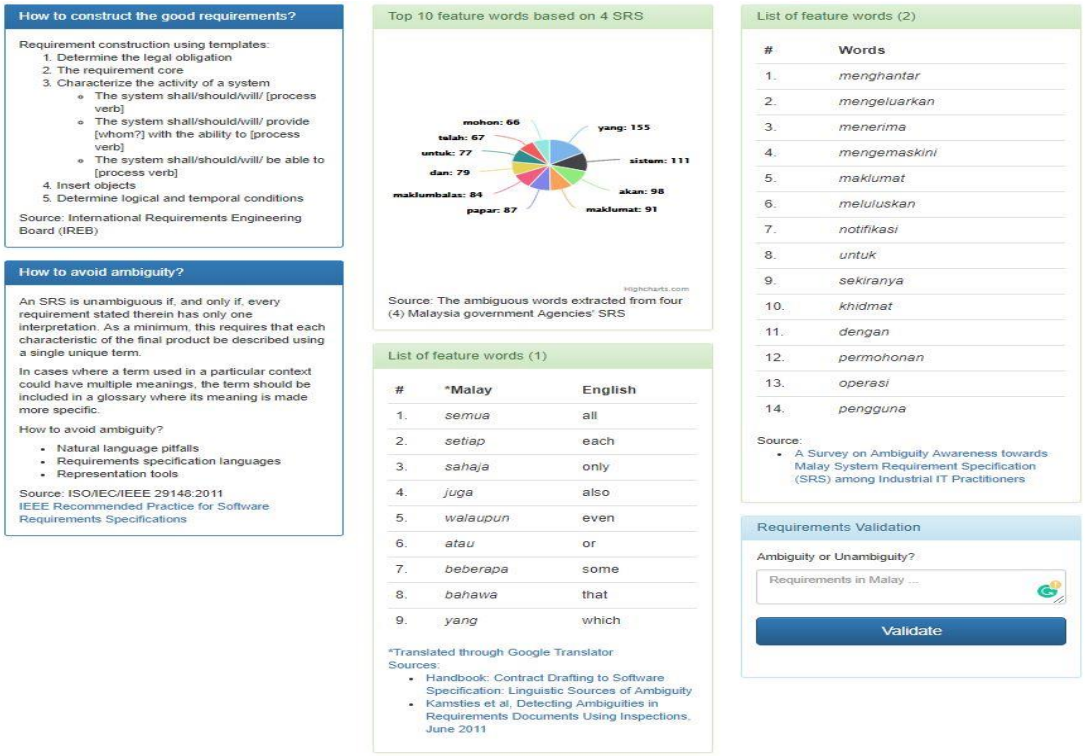

Figure 2. AmbiDetect Landing Page

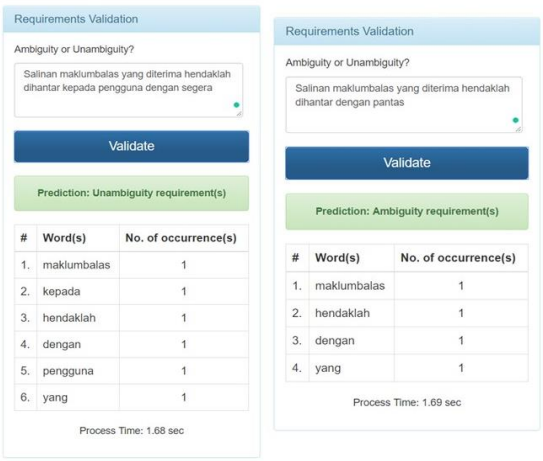

Figure 3. Ambiguity Detection Feature

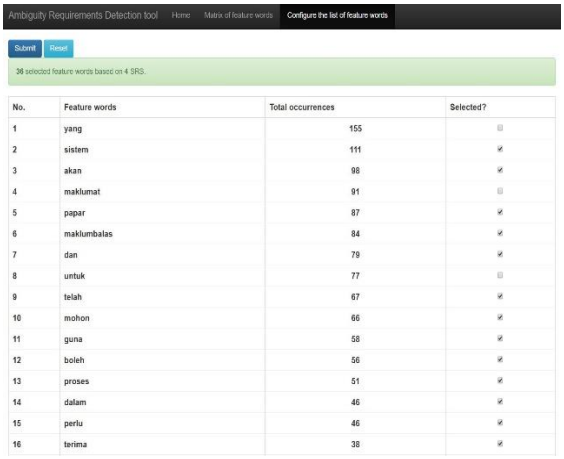

Figure 4. Configure Feature-words

• Data Cleansing. In this activity, we cleaned or scrubbed software requirement specifications (in unstructured text form). The task consists of (i) word filtering, (ii) stop word removal, and (iii) stemming. This task was conducted manually since the Malay language currently does not have a library or tool to perform stop word removal and stemming.

C. Text Processing

From 108 requirement specifications, we gathered 405 feature-words. We formulate a text dictionary by calculating the number of occurrences for each feature-words (in each requirement specification) and integrate with the requirement labels. The text dictionary will be used as the training set.

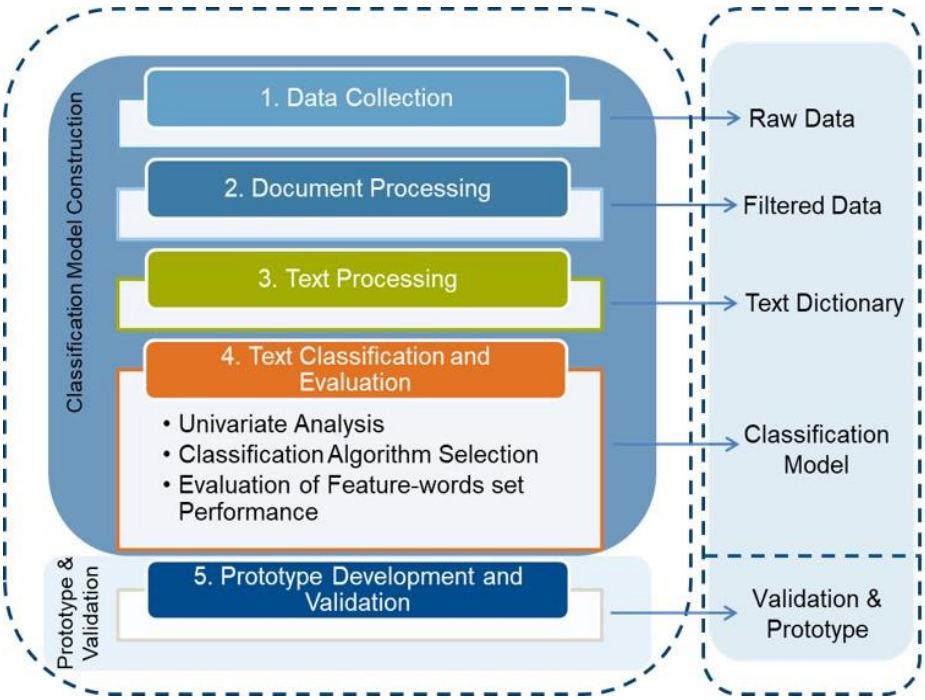

Figure 5. Requirement Specification Detection Framework

D. Text Classification

The Random Forests classification algorithm (Breiman (2001)) was used to classify the ambiguous and unambiguous software requirements. This classification algorithm was selected since it performed well in our dataset based on the work by Osman and Zahrin (2018).

5. User Study

We conducted a user study to evaluate our initial approach to detecting an ambiguous requirement specification. Ten (10) respondents (i.e. working as System Analysts and/or Requirement Engineers) were selected. Three (3) of the respondents have < 2 years of experience, four (4) respondents have 2 – 5 years of experience and three (3) respondents have more than 5 years of experience in requirements engineering. At first, the respondents were asked to use the prototype. Then, they were asked to answer several questions. A (six-level) Likert scale format was used in this survey: strongly agree, agree, moderately agree, moderately disagree, disagree, and strongly disagree. This user study mainly focuses to collect this information:

i. *The accuracy of the ambiguity detection result*. Seven (7) of the respondents moderately agree, two (2) of the respondents strongly agree while only one (1) agree on the accuracy of the result of the tool. We may summarize that the result of this tool is reasonably accepted.
ii. *The tool increases the productivity of developing SRS.* Three (3) of the respondents strongly agree that the tool increases the productivity of developing SRS. One (1) respondent agree while six (6) moderately agree that the tool increased the productivity in developing SRS. The result may indicate that the tool can facilitate software engineers in producing SRS.

6. Conclusions & Future Work

As part of our goal in providing a platform for requirement engineers to produce better requirement specifications, we introduce AmbiDetect, an ambiguous requirements specification detection tool. The ambiguous requirement specification detection engine is based on the combination of supervised machine learning and text mining to extract the classification features to formulate an ambiguous requirement specification model. AmbiDetect is now only available for requirement specifications that are written in the Malay language; however, the tool may easily be changed to other requirements that are written in other languages by changing the dataset. The initial validation of the tool showed that the classification result is reasonably acceptable and may improve the productivity of formulating SRS. We believe that tools have made a significant contribution to improving the quality of SRS and supporting requirements testing or review tasks.

For future work, we see that this work can be improved by (i) improving the ground truth of the training dataset, (ii) increase the volume of the dataset, (iii) formulating more feature-words, (iv) applying other text mining techniques, and (v) improving feature selection.

7. Acknowledgment

Thank you the respondents of the user study for their support and commitment.

References

1. IEEE (1998). IEEE recommended practice for software requirements specifications. IEEE Std 830-1998, pages 1–40.
2. ISO (2011). ISO/IEC/IEEE international standard - systems and software engineering – life cycle processes –requirements engineering. ISO/IEC/IEEE 29148:2011(E), pages 1–94.
3. Berry, D. M., Bucchiarone, A., Gnesi, S., Lami, G., and Trentanni, G. (2006). A new quality model for natural language requirements specifications. In Proceedings of the international workshop on requirements engineering: foundation of software quality (REFSQ).
4. Bourque, P. and Fairley, R. E., editors (2014). SWEBOK: Guide to the Software Engineering Body of Knowledge. IEEE Computer Society, Los Alamitos, CA, version 3.0 edition.
5. Breiman, L. (2001). Random forests. Machine learning, 45(1):5–32.
6. Dictionary.com (2018). www.dictionary.com.
7. Femmer, H., Fernández, D. M., Juergens, E., Klose, M., Zimmer, I., and Zimmer, J. (2014). Rapid requirements checks with requirements smells: Two case studies. In Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering, RCoSE 2014, pages 10–19, New York, NY, USA. ACM.
8. Fernández, D. M. and Wagner, S. (2015). Naming the pain in requirements engineering: A design for a global family of surveys and first results from Germany. Information and Software Technology, 57:616 – 643.
9. Haron, H., Abdul Ghani, A. A., and Haron, H. (2015). A conceptual model to manage lexical ambiguity in Malay textual requirements. ARPN Journal of Engineering and Applied Sciences, 10(3):1405–1412.
10. Hussain, A., Mkpojiogu, E.O.C., Nawi, M.N.M. (2017). Capturing customer satisfaction and dissatisfaction in software requirements elicitation for features in proposed software systems. Journal of Engineering and Applied Sciences, 12 (21), pp. 5590-5597.
11. Kamsties, E., Berry, D. M., and Paech, B. (2001). Detecting ambiguities in requirements documents using inspections. In Proceedings of the first workshop on inspection in software engineering (WISE'01), pages 68–80. Citeseer.
12. Mavin, A., Wilkinson, P., Harwood, A., and Novak, M. (2009). Easy approach to requirements syntax (ears). In Requirements Engineering Conference, 2009. RE'09. 17th IEEE International, pages 317–322. IEEE.
13. Mudavath, M., Kishore, K.H., Hussain, A., Boopathi, C.S. (2020). Design and analysis of CMOS RF receiver front-end of LNA for wireless applications. Microprocessors and Microsystems, 75, art. no. 102999.
14. Osman, H. (2018). AmbiDetect - an ambiguous software requirements specification detection tool. https://youtu.be/Vu-1wl7tJjs.
15. Osman, M. H. and Zaharin, M. F. (2018). Ambiguous software requirement specification detection: An automated approach. In Proceedings of the 5th International Workshop on Requirements Engineering and Testing, RET '18, pages 33–40, New York, NY, USA.ACM.
16. Pohl, K., and Rupp, C. (2011). Requirements Engineering Fundamentals: A Study Guide for the Certified Professional for Requirements Engineering Exam - Foundation Level - IREB Compliant. Rocky Nook, 1st edition.
17. Unterkalmsteiner, M., Feldt, R., and Gorschek, T. (2014). A taxonomy for requirements engineering and software test alignment. ACM Transactions on Software Engineering and Methodology (TOSEM), 23(2):16.