International Journal of Technology 14(4) 713-723 (2023)
Received April 2022 / Revised August 2022 / Accepted January 2023

International Journal of Technology

http://ijtech.eng.ui.ac.id

ReqGo: A Semi-Automated Requirements Management Tool

Shi-Jing Koh[1], Fang-Fang Chua[1*]

[1]*Faculty of Computing and Informatics, Multimedia University,63100 Cyberjaya, Selangor, Malaysia*

Abstract. This study deals with issues of changes in requirements management by dealing with requirements ambiguity and prioritization. A hypothesis about the possibility of integrating machine learning techniques and requirements management processes has been proven. It highlights the efforts in automating requirements ambiguity identification, requirements classification, and prioritization considering multi-criteria in decision-making through the utilization of Natural Language Processing (NLP) techniques and Universal Sentence Encoder. Naïve Bayes (NB) classifier has been applied with its remarkable performance on binarily classifying requirements. Although existing methods proved to improve one or two of the process significantly, it rarely integrates the whole requirements management activity. The proposed tool helps the development team to manage the requirements systematically. The prioritization algorithm is proved to work as expected by considering multiple constraints before calculating the priority value. Meanwhile, it identifies the ambiguity that exists in the requirement automatically. The ambiguity classifier successfully identifies 87.5% of requirements accurately. Possible future work could be done in improving the prioritization module by allowing automated estimation of priority value upon requirements change. Future work may extend the automation coverage by providing test case generation.

Keywords: NLP; Requirements; Requirements ambiguity; Requirements prioritization

1. Introduction

Requirements Management has been the backbone of most software development to achieve the goal of every project. It handles rapidly changing requirements with proper planning, analysis, documenting, prioritizing, and integration of requirements to provide up-to-date requirements for a project.

It is crucial to get the "right" requirements from the clients and put the requirements in the "right" place (Hafeez, Rasheed, and Khan, 2017). Nevertheless, constantly changing requirements may end up in a large-scale system and require much effort in managing the details to ensure the information is always up to date. A key part of requirements management is managing the changes. The manual process of labeling requirements could be time-consuming and error-prone (Iqbal, Elahidoost, and Lucio, 2018). A failure to identify any issues in requirements in the early stage could result in project delay, which brings out the issues of loss of revenue, tarnished reputations, and loss of trust (Riazi and Nawi, 2018), other than adversely affect the expectation of the clients and the final product and finally, lead to project failure.

*Corresponding author's email: ffchua@mmu.edu.my, Tel.: +60383125406
doi: 10.14716/ijtech.v14i4.5631

This paper aims to propose a semi-automated requirements management tool, ReqGo and analyses how it benefits the existing requirements management process. More precisely, we pursue the following research questions: 1) How to identify ambiguity in requirements automatically? 2) How to improve requirements prioritization tasks through semi-automation? 3) How to integrate automated tasks into the requirements management process? The proposed tool makes use of Natural Language Processing (NLP) to classify requirements, detect requirement ambiguity, and automatically prioritize them using multi-criteria decision-making to facilitate effective resource utilization, besides providing the ability for users to manage their requirements and relevant artifacts. Among various NLP algorithms available to analyze and process the data, Naïve Bayes (NB) has been chosen due to its feature that supports binary classification, which is ideal for classifying the requirement into two categories, i.e., Functional and Non-Function Requirements.

According to previous studies, the correct requirements classification of requirements and clear definition of requirements have been the main focuses of researchers to allow filtering and prioritizing of requirements. There are numerous algorithms, including Term Frequency - Inverse Document Frequency (TF-IDF) (Wein and Briggs, 2021; Dias-Canedo and Cordeiro-Mendes, 2020) and machine learning techniques like Support Vector Machine (SVM) (Shariff, 2021; Kurtanović and Maalej, 2017), Naïve Bayes (NB) (Shariff, 2021), Logistic Regression (LR) (Dias-Canedo and Cordeiro-Mendes, 2020) and Natural Language Processing (NLP) (Wein and Briggs, 2021; Asadabadi *et al*., 2020; Aysolmaz *et al*., 2018; Kurtanović & Maalej, 2017; Emebo, Olawande, and Charles, 2016), have been implemented in various requirements management tasks to analyze and classify requirements by going through requirements normalization, feature extraction, feature selection, and finally classification.

Existing requirements management software such as IBM DOORS Next (IBM, n.d.) and MaramaAIC (Kamalrudin, Hosking, and Grun, 2017) provide the ability to manage requirements for complex software and systems requirements environment with NLP techniques supported to improve the abilities in detecting requirements quality issues and traceability through a certain degree of automation. However, it is notable that they are still lacking the integration of automatic requirements prioritization in the tool. Inspired by the existing tools, ReqGo emphasized capturing requirements issues in the early stage while proposing a semi-automated requirements prioritization module to reduce the human effort in ranking the requirements.

The remaining part of the paper is structured as the followings: Section 2 describes the fundamental theory and the working procedure of ReqGo, including its overall architecture, the workflow of requirements ambiguity identification and requirements prioritization with their corresponding algorithms, as well as print screens of the tool, followed by Section 3 which presents the results and summarize the major findings of our study. Finally, Section 4 concludes the paper.

2. Methods

2.1. ReqGo Architecture and Workflow

In general, ReqGo contains five modules which are User Account Management, Requirement Record Management, Requirement Prioritization, Requirement Verification & Validation, and Requirement Traceability. Figure 1 represents a high-level flow of how the user could utilize the tool. By using ReqGo, users need to insert the requirements collected in natural language. The extracted requirements will then be ready for documentation and analysis. The requirements are processed and tagged for different categories and possible ambiguity. The requirements will then be prioritized accordingly, and finally, verification and validation.

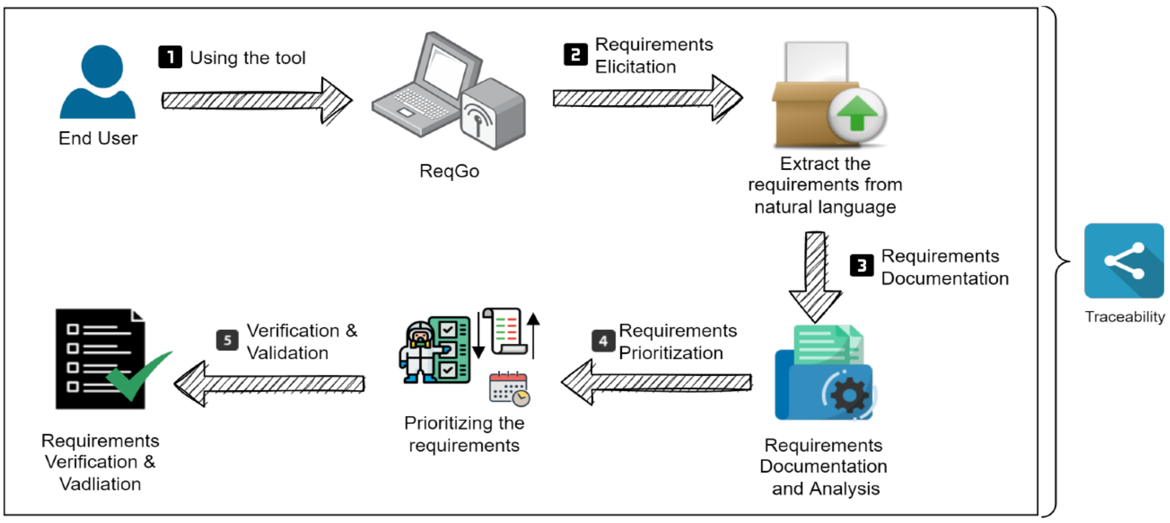

1 Using the tool
End User
ReqGo
2 Requirements Elicitation
Extract the requirements from natural language
3 Requirements Documentation
Traceability
Requirements Documentation and Analysis
4 Requirements Prioritization
Prioritizing the requirements
5 Verification & Validation
Requirements Verification & Vadliation

Figure 1 High-Level Flow of ReqGo

2.2. Data Collection

Before proceeding with any classification process, we need to collect the dataset for training and evaluation purposes. The dataset collected with being divided into two portions. The first part will be used to train the classifier, while the second part will serve as the validation set. To perform this requirements classification phase, the datasets are gathered from three sources (Lima *et al*., 2019; Ferrari *et al*., 2017; Cleland-Huang *et al*., 2007). Figure 2 presents a summary of requirements and their composition in the dataset. The dataset is divided into twelve categories, including Functional (F) and Non-Function requirements like Security (SE), Usability (US), Operational (O), Performance (PE), Look and Feel (LF), Availability (A), Maintainability (MN), Scalability (SC), Fault Tolerance (FT), Legal (L), and Portability (PO).

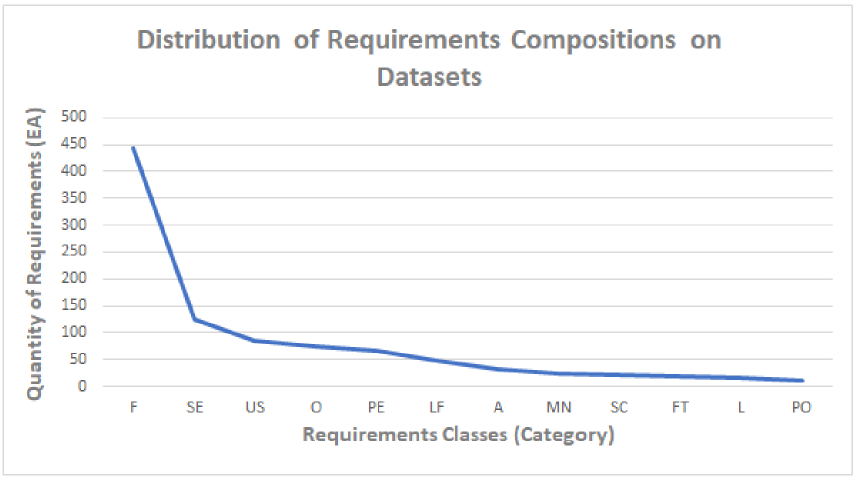


Figure 2 Distribution of Requirement Compositions on Datasets

2.3. Requirements Classification and Ambiguity

ReqGo is deploying Machine Learning techniques for classifying Functional Requirements and Non-Functional Requirements and identifying ambiguous terms in requirements to discuss the machine learning methods utilized for classifying the requirements. The implementation of Machine Learning can be divided into three processes, namely Text Normalization, Text Vectorization, and Text Classification. Before a requirement can be further processed according to the need, text normalization is essential to convert the requirement into a standard form. In our case, we make use of Tokenization, Part-Of-Speech (POS) tagging, and Lemmatization to break down the words in terms of sentence structure and vocabulary. Then, we utilize Google's Universal Sentence Encoder method to embed the text into meaningful vector representations to evaluate words in requirements (Figure 3). In ReqGo, we utilize its ability to measure the degree of two pieces of text that carry similar meanings. It is useful in sentence classification tasks by analyzing the semantics similarity through the vectors generated through cosine similarity calculation based on the given equation:

$$Similarity = cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \quad (1)$$

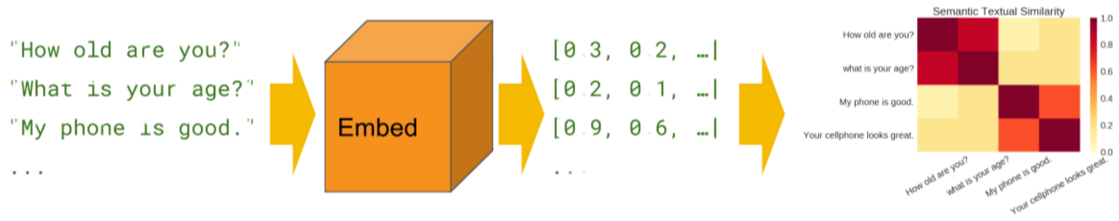


Figure 3 Illustration of the process of Universal Sentence Encoder

In ReqGo, we employ the NB classifier, a supervised classification algorithm, to group the requirements automatically. NB supports binary classification to divide the requirements into functional and non-functional requirement categories; the requirements are further identified as ambiguous or non-ambiguous to discover any possible issues that exist in the requirements. The classifier is trained using datasets collected before it can be used to provide a more accurate result. Figure 4 presents an algorithm for initializing requirements classification, including model training and model testing. The function requires labeled dataset input to produce the outcome of a well-trained requirement classifier. There are several values to be set at the beginning of the process, such as the total amount of data in the dataset used for this training, the amount of data for testing the classifier, and the amount of data for training the classifier. After the classifier has been trained and tested, it is stored in a JSON file for reuse purposes.

In order to analyze the ambiguity that exists in requirements, an ambiguity checker is implemented. The process is started with a step in line 1 of Algorithm 2 (as shown in *Figure 5*), where the threshold value is set. Threshold value places an important role as it determines whether a requirement is identified as ambiguous once it exceeds the value. Using TensorFlow.js, we embedded the requirements and ambiguous terms through Universal Sentence Encoded to support our process of examining the similarity of the words with high dimension vectors generated. Finally, the outcome is stored in the database. With ambiguous requirements detected, the users will be notified to take action to prevent any issues.

```
Algorithm 1 Requirement Classifier Training & Testing
    Input:   Labelled data set in comma-separated values (CSV) file type
    Output:  Trained Classifier
 1: Read stream from the CSV file into $data$
 2: Store labels and Features into $x$ and $y$
 3: Shuffle $x$ and $y$ arrays
 4: Set $M = totalnumberofdataset$
 5: Set $MTest = numberofdatathatwillbeusedfortesting$
 6: Set $MTraining = M - MTest$
 7: for $iteration = 1, 2, \ldots, iteration < datasize$ do
 8:     if $iteration < MTest$ then
 9:         Push data into $XTrain$ and $YTrain$
10:     else
11:         Push data into $XTest$ and $YTest$
12:     end if
13: end for
14: for $iteration = 1, 2, \ldots, iteration < MTrain$ do
15:     Tokenize and steam the data in $XTrain$ and $YTrain$
16:     Train classifier with $XTrain$ and $YTrain$
17:     Store trained classifier as $classifier$
18: end for
19: for $iteration = 1, 2, \ldots, iteration < MTest$ do
20:     Classify the testing set with $classifierS$
21: end for
22: Push $MTest$ of data into $XTrain$ and $YTrain$
23: Calculate the accuracy of the $classifier$
24: Store classifier into JSON file
```

Figure 4 Algorithm of Requirement Classifier Training and Testing

```
Algorithm 2 Requirement Ambiguity Checker
    Input:   Requirement
    Output:  Ambiguity matching result
 1: Set threshold value
 2: Set ambiguous terms
 3: Get embedded requirement and ambiguous term by using TensorFlow.js
 4: Set embedded terms, $terms$ equal to the embedded result
 5: Set embedded requirement, $sentence$ equal to embedded requirement outcome
 6: for $iteration = 1, 2, \ldots, iteration < termssize$ do
 7:     Calculate the similarity of the words in requirement that match the ambiguous terms
 8:     Set $ambiguous = result > threshold$
 9: end for
10: Return ambiguous in Boolean type
```

Figure 5 Algorithm of Requirement Ambiguity Checker

2.4. Requirements Prioritization Logic

ReqGo provides semi-automated requirements prioritizing methods based on a requirement priority value (RPV) formulation function (Hujainah *et al*., 2021), including the decision-making method, priority clustering, and insertion sort. To simplify the process, ReqGo involves the sorting algorithm after the calculation of RPV for each requirement without clustering algorithms. Figure 6 presents the flow of how the two categories of prioritization tasks interact to produce the desired outcome, i.e., reduce the manual effort in finalizing the priority value and the order of requirements collected when it comes to implementation.

Figure 7 illustrates the flow of the requirements prioritization method algorithm by gathering all the necessary data. This includes stakeholder weights, priority values assigned by each stakeholder, requirement dependencies and their occurrence as parent requirements, and the efforts required to turn the requirement into reality.

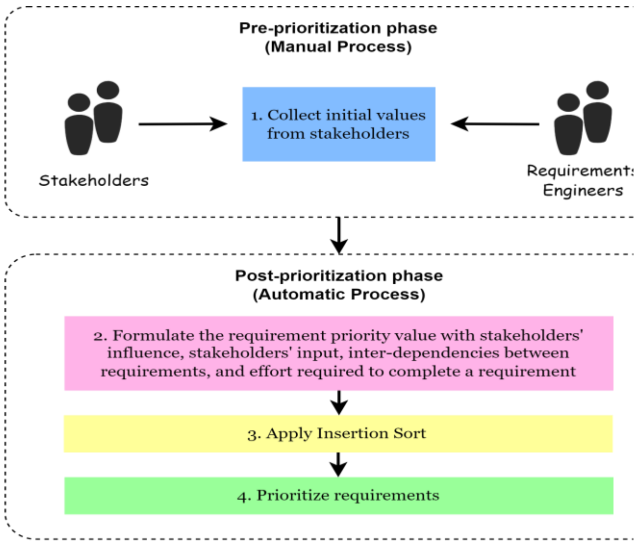

Pre-prioritization phase
(Manual Process)
Stakeholders
1. Collect initial values from stakeholders
Requirements Engineers
Post-prioritization phase
(Automatic Process)
2. Formulate the requirement priority value with stakeholders' influence, stakeholders' input, inter-dependencies between requirements, and effort required to complete a requirement
3. Apply Insertion Sort
4. Prioritize requirements

Figure 6 Illustration of Requirements Prioritization Flow in ReqGo

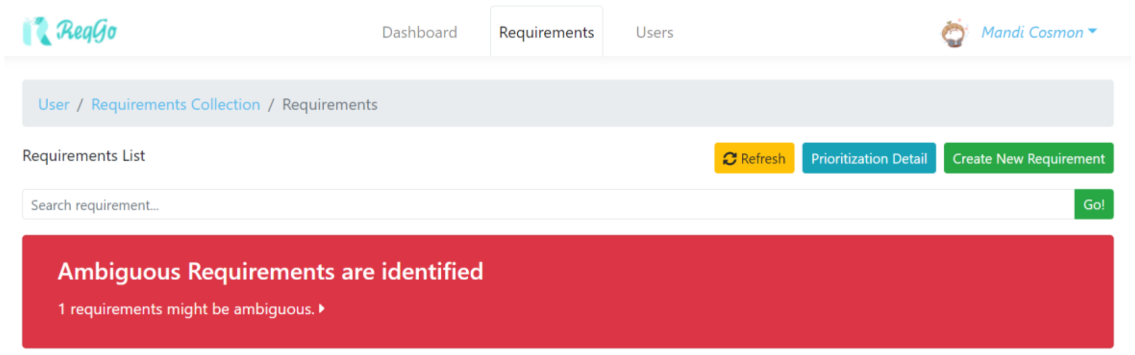
Algorithm 3 Requirement Prioritization

```
    Input: Stakeholder's weights, Stakeholder's priority input, Requ
ments Dependency, Requirements effort
    Output: Prioritized Requirements
for $iteration = 1, 2, \ldots, iteration < requirements.length$ do
    Set occurrence of a requirement as parent requirement that lin
    to one another.
    Set $rpv = \sum_{n=1}^{\infty} S_{weight} * S_{value}$
end for
Sort requirements according to the RPV, occurrence, and effort
for $iteration = 1, 2, \ldots, iteration < requirements.length$ do
    if Requirement has a parent then
        Get parent requirement index
        if $parentIndex > currentIndex$ then
            Insert current requirement after the parent requireme
        end if
    end if
end for
Store ranked priority value to each requirement
```

Figure 7 Algorithm of Requirement Prioritization

2.5. Printscreens of ReqGo

We have demonstrated several screens of ReqGo to achieve or complete tasks, specifically in triggering the ambiguity checker and automatic requirements prioritization. In such a case, the ambiguous requirement(s) is identified, and a warning message will be displayed on the requirements listing page (as shown in Figure 8).

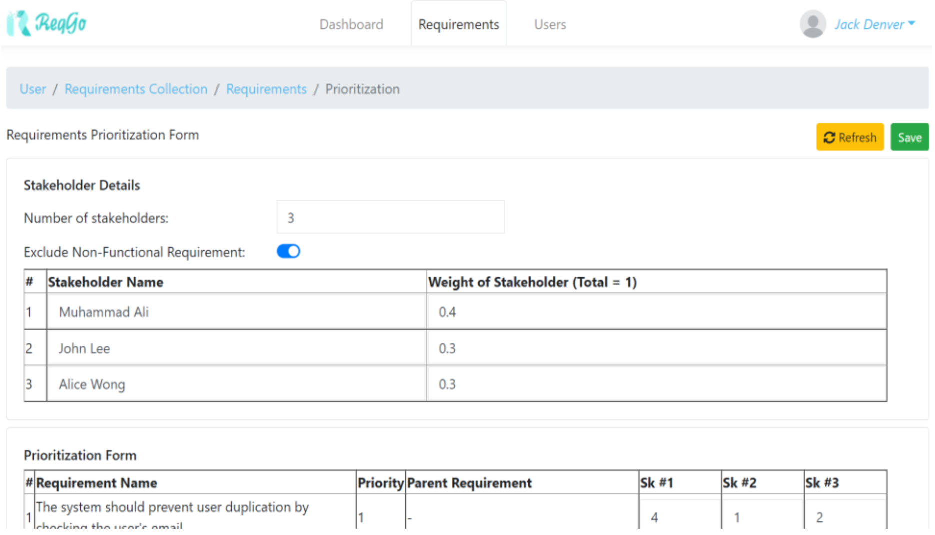


Ambiguous Requirements are identified

1 requirements might be ambiguous.

Figure 8 Ambiguous warning message on requirements listing page

In the view of prioritizing requirements, there are a few information needed before one can automate the priority ranking. With the values of stakeholders' weight and inputs collected, the tool will normalize the values, the dependencies among the requirements, and the efforts required to complete the requirement. The results present a ranking of the requirements to indicate the priority value of the requirements. Figure 9 demonstrates the information that has to be collected to allow requirements prioritization.



User / Requirements Collection / Requirements / Prioritization

Requirements Prioritization Form

Stakeholder Details

Number of stakeholders: 3

Exclude Non-Functional Requirement: ●

#	Stakeholder Name	Weight of Stakeholder (Total = 1)
1	Muhammad Ali	0.4
2	John Lee	0.3
3	Alice Wong	0.3

Prioritization Form

#	Requirement Name	Priority	Parent Requirement	Sk #1	Sk #2	Sk #3
1	The system should prevent user duplication by	1	-	4	1	2

Figure 9 Print Screen of Requirement Priority Form of ReqGo

3. Results and Discussion

In this study, we have proposed a semi-automated requirements management tool, ReqGo, to help in managing the requirements and their relevant artifacts while improving the process through the utilization of machine learning techniques. Testing has been performed to identify and remove defects while comparing the expected and actual outcomes to ensure the tool performs well and complies with the objectives. The accuracy of the requirements classifier is evaluated to verify the accuracy of the classifier in identifying the requirement categories. Then, a user evaluation was conducted by gathering feedback from the testers to verify the usability of the proposed tool to allow future improvements.

3.1. Classifier Accuracy Analysis

In this accuracy test, the requirement classifier is measured based on its ability to accurately classify the functional and non-functional requirements. One collection has been used for training the classifier, while the other is applied to test the classifier's accuracy. From the 1006 rows of requirements we have gathered, 200 data are used to test the correctness of the classifier in performing the classification task. As the requirements in the dataset will be shuffled and the data used for training and testing might vary for each training and testing phase, the process is repeated ten times. The mean value of the ten experiments is calculated to obtain a more reliable record. The result is then recorded and discussed. Equation 2 displays the formula for calculating the accuracy of the classifier, which computes the number of data classified correctly over the total number of datasets. The summary table of the accuracy obtained from ten times classifier execution is described in Table 1. It showed that the accuracy of the classifier is around 87.5% on average. The classifier remains stable at an accuracy of 86% to 87.5% most of the time. The classifier's result is satisfactory in classifying the functional and non-functional requirements. However, it can be further improved by expanding the datasets for training and testing, as Siswanto *et al.* (2022) noted that a more extensive and comprehensive dataset is required to achieve better accuracy. This will be useful to ensure the classifier is well-trained before utilizing it.

$$Accuracy = \frac{Number\ of\ true\ positive + Number\ of\ true\ negative}{Total\ data\ sets} \qquad (2)$$

Table 1 Classifier Accuracy Result

Execution No.	Accuracy (%)
1	89
2	90.5
3	84.5
4	90
5	86
6	87.5
7	87
8	87.5
9	87
10	86
Average	87.5

3.2. Comparative Analysis

Many researchers work with automating the requirements management process to reduce human effort, cost, and possible errors and mistakes. To observe the significance of ReqGo in enhancing the process, the comparison of ReqGo with current requirements management tools is discussed. As very little research has been conducted for a fully integrated requirements management tool, we have reviewed the current requirements management tool on the marketplace, with different degrees of automation, to discuss the features and functionalities provided.

Table 2 Comparison Among Requirements Management Tools with ReqGo

Feature & Functionality Automation	DOORS Next	HP ALM	MaramaAIC	ReqGo
Elicitation	✗	✗	✗	✗
Documentation	✓	✗	✓	✗
Tasks & Technique	Requirements quality analysis with IBM Watson	-	Inconsistencies checking with a pattern library	Requirements Ambiguity Identification & Classification with NLP
Prioritization	✗	✗	✗	✓
Tasks & Technique	-	-	-	Priority Value Formulation Function (Hujainah *et al.*, 2021)
Verification & Validation	✗	✓	✓	✗
Tasks & Technique	-	Test case automation with UFT	Prototype generation from pattern library matching	Manual test case management
Traceability	✗	✗	✓	✗
Tasks & Technique	-	-	Traceability automation with NLP	Manual traceability links.

Table 2 presents a few requirements management tools that exist in the marketplace, namely IBM Doors Next (IBM, n.d.), HP ALM (Micro Focus, n.d.), and MaramaAIC (Kamalrudin, Hosking, and Grun, 2017). The tools are mainly focusing on the requirements documentation with quality analysis and inconsistencies checking. Concerning vague requirements that might result in unexpected and unnecessary problems when acquiring software, ReqGo focuses on the identification of ambiguous terms in the requirements

through semantical analysis. Besides, ReqGo has made a move in integrating the semi-automated requirements prioritization considering multi-criteria make calculate the priority value.

3.3. User Evaluation

User evaluation has been conducted to study how well users can learn and utilize ReqGo to assist them in managing their requirements during the development process. In this study, a few qualitative post-task interviews have taken place to collect the opinions from the participants in measuring the tool usability pertaining to system performance, usability issues, or design suggestions. Several users have been invited to test out the tool, and users' satisfaction has been collected to analyze the usability of ReqGo. The evaluation was divided into two phases. The first phase was carried out with no documentation or guide provided before the testing; the second phase was performed after the users had been given clear guidance. Questions were asked in both the first and second phases of the evaluation to get insights into user experiences, existing design issues, and possible improvements throughout the utilization of the tool. On average, the testers might consider getting some assistance to fully utilize the tool even though the system gives an error message sufficiently clear to identify any problem to be fixed. Besides, a user claimed that more functionality and capabilities are expected to manage requirements. Then, testers declare that the ambiguity identification module is more likely to assist users in discovering any requirements defects in the early stage while enhancing productivity. However, it is also suggested to provide an ambiguous reason whenever vague requirements are found to allow requirements engineers to resolve the ambiguity.

Meanwhile, the requirements prioritization module provides a positive experience in effectively prioritizing the requirements. In the second evaluation, the users suggest a better dependency display within an entity detail page. Testers also reported that the response time of ReqGo is found to be slow, and it takes a few seconds to obtain the data from the server. Overall, most users are satisfied with the user interface, functionality, and workflow of ReqGo.

3.4. Discussion

Overall, the study has covered three research questions. The first research question, "How to identify ambiguity in requirements automatically?" has been addressed through the utilization of the Universal Sentence Encoder to match the semantic similarity of the requirements with our requirements ambiguity terms. The user evaluation shows that it has been helpful in identifying the issues in requirements in the early stage. On the other hand, the requirements prioritization task is automated with the initial value obtained from the user to calculate the priority value considering multiple criteria, namely stakeholders' influence, stakeholders' input, dependencies among requirements, and effort for executing the requirement. The testers show a positive response toward the implementation of requirements prioritization automation. Then, we integrated the automated tasks proposed in our requirement management tool, ReqGo, to allow easy requirements management while reducing the manual efforts in performing the tasks. Through the utilization of ReqGo, the requirements and their relevant artifacts could be traced along with the development phase. The user evaluation shows an encouraging user acceptance of ReqGo through usability verification with real-world testers.

4. Conclusions

In this paper, we have proposed a semi-automated requirements management tool to cover the requirements management process with automated tasks through the utilization of Machine Learning techniques. Naïve Bayes (NB) classifier, Natural Language Processing (NLP), and Universal Sentence Encoder has been used to support the requirements classification, requirements ambiguity identification, as well as requirements prioritization. Several limitations have been identified, including the number of datasets used to train the classifier is less than sufficient to increase the accuracy and reliability of the classification results. Besides, the response time of ReqGo is undesired. This tool may encourage researchers to automate the requirements management process through different algorithms based on the tasks by overcoming the limitations. The ambiguity of the requirements can be identified in the early stages of development to avoid severe issues in the later phase. The proposed requirements prioritization algorithm contributes to calculating the priority value based on multi-criteria. It can be improved by automatically estimating a requirements priority value on requirements change.

References

Asadabadi, M.R., Saberi, M., Zwikael, O., Chang, E., 2020. Ambiguous Requirements: A Semi-Automated Approach to Identify and Clarify Ambiguity in Large-Scale Projects. *Computers & Industrial Engineering*, Volume 149, p. 106828

Aysolmaz, B., Leopold, H., Reijers, H.A., Demirörs, O., 2018. A Semi-Automated Approach for Generating Natural Language Requirements Documents Based on Business Process Models. *Information and Software Technology*, Volume 93, pp. 14–29

Cleland-Huang, J., Mazrouee, S., Liguo, H., Port, D., 2007. NFR. *Zenodo*. Available online at https://doi.org/10.5281/zenodo.26854, Accessed on November 11, 2021

Dias Canedo, E., Cordeiro Mendes, B., 2020. Software Requirements Classification Using Machine Learning Algorithms. *Entropy*, Volume 22(9), p. 1057

Emebo, O., Olawande, D., Charles, A., 2016. An Automated Tool Support for Managing Implicit Requirements Using Analogy-Based Reasoning. *In*: 2016 IEEE tenth international conference on Research Challenges in Information Science (RCIS), Volume 2016, pp. 1–6

Ferrari, A., Spagnolo, G.O., Gnesi, S., 2017. Towards a Dataset for Natural Language Requirements Processing. In: *REFSQ workshops*

Hafeez, M.S., Rasheed, F., Khan, M.R., 2017. An Improved Model for Requirement Management System. *Journal of Information Technology & Software Engineering*, Volume 7(196), p. 2

Hujainah, F., Bakar, R.B.A., Nasser, A.B., Al-haimi, B., Zamli, K.Z., 2021. Srptackle: A Semi-Automated Requirements Prioritisation Technique for Scalable Requirements of Software System Projects. *Information and Software Technology*, Volume 131, p. 106501

IBM., n.d. IBM Engineering Requirements Management DOORS Next. Available online at https://www.ibm.com/my-en/products/requirements-management-doors-next, Accessed on November 11, 2021

Iqbal, T., Elahidoost, P., Lucio, L., 2018. A Bird's Eye View on Requirements Engineering and Machine Learning. *In:* 2018 25th Asia-Pacific Software Engineering Conference (APSEC), Volume 2018, pp. 11–20

Kamalrudin, M., Hosking, J., Grundy, J., 2017. Maramaaic: Tool Support for Consistency Management and Validation of Requirements. *Automated Software Engineering*, Volume 24(1), pp. 1–45

Kurtanović, Z., Maalej, W., 2017. Automatically Classifying Functional and Non-Functional Requirements Using Supervised Machine Learning. *In*: 2017 IEEE 25th International Requirements Engineering Conference (re), Volume 2017, pp. 490–495

Lima, M., Valle, V., Costa, E.A., Lira, F., Gadelha, B., 2019. Software Engineering Repositories: Expanding the Promise Database. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, Volume 2019, pp. 427–436

Micro Focus, n.d. HP ALM. Available online at: https://www.microfocus.com/en-us/products/alm-quality-center/overview, Accessed on November 11, 2021

Riazi, S.R.M., Nawi M.N.M., 2018. Project Delays in The Malaysian Public Sector: Causes, Pathogens and The Supply Chain Management Approach. *International Journal of Technology*, Volume 9(8), pp. 1668–1680

Shariff, H., 2021. Non-Functional Requirement Detection Using Machine Learning and Natural Language Processing. *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, Volume 12(3), pp. 2224–2229

Siswanto, J., Suakanto, S., Andriani, M., Hardiyanti, M., Kusumasari, T.F., 2022. Interview Bot Development with Natural Language Processing and Machine Learning. *International Journal of Technology,* Volume 13(2), pp. 274–285

Wein, S., Briggs, P., 2021. A Fully Automated Approach to Requirement Extraction from Design Documents. *In*: 2021 IEEE Aerospace Conference (50100), Volume 2021, pp. 1–7