Journal of Artificial Intelligence and Computational Technology

Journal homepage: https://ojs.omgfzc.com/index.php/JECT

Developing Parallel Requirements Prioritization Machine Learning Model Integrating with MoSCoW Method

Kawthar Ishag Ali Fadlallah [1*], *Mahir M. Sharif* [2], Moawia Elfaki Yahia Eldow [3]

[1] Department of Computer Science, Omdurman Islamic University, Omdurman, Sudan

[2]Department of Computer and Self Development, Common First Year Unit , Prince Sattam bin Abdulaziz University, AlKharj 11642, Saudi Arabia

[2] Computer Science Dept. _Faculty of Computer Science & Information Technology _ Omduram Islamic University _ Omduram _Sudan

[3] Faculty of Mathematical Science, University of Khartoum, Khartoum 11115, Sudan, University of North Texas, Denton, TX, USA

*Correspondence: E-mail: kawthar1140r@gmail.com

Article Info

Article History:
Submitted/Received 20 July 2024
Revised in revised format 25 August 2024
Accepted 01 October 2024
Available Online 15 October 2024
Publication Date 15 October 2024

Keyword:
Requirements Prioritization,
Parallel Ranking,
MoSCoW Method,
Random Forest

Cite this article: *Fadlallah, K. I. A., Sharif, M. M., & Eldow, M. E. Y. (2024). Developing parallel requirements prioritization machine learning model integrating with MoSCoW method. Journal of Artificial Intelligence and Computational Technology, 1(1).*

COPYRIGHT © 2024 First author, et al. This is an open access article distributed under the terms of the Creative Commons Attribution License (CC BY).

ABSTRACT

Requirements Prioritization (RP) is an attempt to rank the requirements based on the value added to the business. It is a preprocessingstep in software implementation as well as a prevalent need thing to get customer satisfaction, decrease the risk of requirements volatility, develop cost-effective software, and maintain the level of quality in the software system. Many research focusing on prioritizing the requirements using one or several criteria like time, dependency, and scalability. However, all of them concern with sequential prioritization only. To the best of our knowledge no work focused on parallel ranking in prioritization, which permit the simultaneous requirements implementation that reducing the implementation time. In this study we developed a new requirements prioritization for determine the requirements priority level in parallel format using Random Forest classifier based MoSCoW method (RF-MM). When we applied our prioritization model on to (Testcase MIS system with priority) industrial dataset. the total implementation time were equal to 76.0 seconds when ranking in sequential format; whereas the total time were equal to 33 seconds in parallel ranking. Hence, the parallel ranking capable of reducing implementation time to more than half.

1. Introduction

Requirements Prioritization (RP) is one of the pivotal activities in requirements engineering process. RP is defined as a decision-making process in which software engineers cooperate with stakeholders, to understand their demands, to define the application order of requirements by considering the cost, time, and technical constraints. RP aims is to evaluate requirements and concentrate on the most crucial ones based on the value that they can add to the business.

The concept RP appeared with the increased requests of complex software systems by stakeholders (Aurum, 2005). It is proven that it is very difficult to implement all stakeholders' requirements in a single release due to the increasing number of requirements from the availability of massive volume of data (big data) with inadequate resources such as time, lacking budget, and technical staff (Dabbagh & Lee., 20١٣). Moreover, the different opinions about requirements prioritization by different stakeholders make the RP task hard (Bukhsh et al., 20٢٠). Another challenging aspect in RP is handling requirements dependency (RD), which means the requirements are dependent or reliant on each other (Sher et al., 20٢٠). Therefore, the requirements prioritization must be considered, in order to get customer satisfaction, decrease the risk of requirements volatility, develop cost-effective software, and maintain the level of quality in the software system (Aurum, 2005; Sher et al., 20٢٠).

The use of machine learning (ML) has increased in different areas, both industrial and academic (Marco et al., 20٢٠). the most important advantages of ML, as the following: (i) ML techniques can assist faster decision-making and solutions to complex problems (ii) enhance performance in several regions (Achimugu et al., 2014). (iii) permit the automation of various tasks; and (iv) allow the prediction of favorite values of a particular sets and to make approximate ranks for requirements via modeling founded on training data. the modern ML techniques can improve features such as scalability and performance.

Several machine learning techniques for RP have been presented in the literature such as (Shao et al., 2017; Gupta & Gupta., 20١٨; Chua et al., 2022). Each technique uses one or numerous criteria in prioritizing requirements. However, all techniques have limitations, not only associated with scalability and requirements dependencies (Achimugu et al., 2014; Shao et al., 2017), but also due to unawareness on the impact of the requirements prioritization on reducing software implementation time.

Our model reduces the software implementation time, due to the awareness of parallel ranking process in RP. It conserns the determining and classifing the requirements priority levels into in parallel format which permit batch implementation using Random Forest classifier based MoSCoW method. We apply our model to the Testcase MIS system with priority dataset.

We develop machine learning model integrated with MoSCoW Method, that determine the requirements priority level in parallel format from an input Testcase MIS system with priority dataset after putting the values in an appropriate pattern, then removes null values and applies rules to determine the parallel priority level.

Our study aims to build a machine learning model relied on the software requirements document to determine the proper priority level by determining the MoSCoW categories (Must, Should, Could, and Wouldn’t) as priority levels (High, Medium, and low) in parallel format. Parallel format, which reduce the implementation time via simultaneous requirements implementation. The total implementation time we achieved were equal to 33 seconds in parallel ranking formant; whereas The total implementation time were equal to 76.0 seconds when ranking in sequential format.

2. Related Works

Machine learning approaches have been heavily used to develop requirements prioritization approaches (Qayyum, & Qureshi, 2018; Jan et al., 2020; Bukhsh et al., 20٢٠). These approaches were believed very effective due to the automation and tools supporting their implementation. Machine learning and data mining techniques has the capability to automation, resolve the conflicts between the stakeholders and developers (Hujainah et al., 2018; Sher et al., 20٢٠), etc., high level of handling the scalability problem (Hudaib et al., 2018; Hujainah et al., 2018; Sher et al., 20٢٠; Amelia& Mohamed 2022). We will show the previous research in this field. Most researchers had proposed algorithms to prioritize and classify the software requirements which handling the dependency between requirements. In this section, we will show the previous research in this field.

(Alrashoud & Abhari., 2015) presented a mathematical interpretation to model that tactile the uncertainty open issues in human estimation and their limited knowledge, using Fuzzy Inference System (FIS). They identify the Weighted importance, dependency constraints, and the risk criteria. The results demonstrated that the FIS model has achieved a higher degree of satisfaction when compared to a genetic algorithm-based model. However, the accuracy is not estimated.

(Allex et al., 2016) presented Interactive Next Release Problem (iNRP) model for requirements prioritization. The model utilizes Least Median Square (LMS) and Multilayer Perceptron (MLP) techniques. In the iNRP model two architectural settings is specified by decision maker(DM), the weight of the tacit preferences is compared to the explicit ones for the suitability calculation. Then, the learning process is executed using the set of samples captured in the preceding stage as a training dataset. However, the performance of this learning model has not been evaluated.

(Shao et al., 2017) developed a semi-automatic requirements prioritization approach, called Drank. Drank takes the dependencies among requirements and the stakeholders' preferences into consideration. Rank employs RankBoost algorithm to produce requirement prioritization formula in subjective manner. a controlled experiment made to validate the DRank efficiency, constructed on comparisons with Case Based Ranking, AHP, and EVOLVE. The results provided more effectiveness as compared with alternative approaches. However, this work seems to provide only the contribution and business dependencies. Further, the authors highlight that their approach is still motivated to the issue of subjectivity especially in the process of requirements evaluation.

(Gupta & Gupta., 20۱۸) proposed a dependency based collaborative requirement prioritization approach termed (CDBR), Three aspect measured in CDBR are dependencies between requirements, stakeholder and developers and scalability. They providing improvements in prioritization results, processing time, and producing acceptable and accurate priority list in comparison to AHP and IGA. However, it ignores the errors (false positive rate) implicated during the prioritization process.

(Gupta & Gupta., 20۱۸) presented collaborative requirement prioritization method to support developers in making right decision during prioritization via dependency classification and weight assignment method Good result achieved in term of low complexity when compared with existing method. However, it limited to three type of dependency, suffer from biases issues, lack of automation.

(Misaghian et al., 2019) presented a requirement prioritization approach, using fuzzy graph algebra weighted page rank algorithm tensor decomposition. In this approach, they mixed requirements order provided by tensor decomposition with the dependency order. The Results showed the improvements in accuracy, time consuming, and ease of use. However, it limited only t to the increase/decrease cost dependency type, and it ignores the stakeholders' conflicts.

(Abbas et al., 2019) presented a requirements prioritization method using a modified PageRank algorithm. The method restricted dependency type defined in Ecore meta-model. The Results showed the method outperform five existing requirements prioritization methods in terms of efficiency and accuracy. However, it limited to dependency type defined in Ecore meta-model and suffers from lack of automation.

(Gupta & Gupta., 2022) presented a scalable framework for prioritizing the requirements of obtaining the inputs from both stakeholders and developers. They used Intuitionistic Fuzzy Approach (IFS) to support stakeholder's opinion. Whereas the developer's offer dependency graph as their part of the input. the initial priority values calculated using Weighted Page rank. The result indicates that this framework is proficient of providing accurate and comparable results by handling technical constraints of dependency as compared to existing techniques. However, the requirement prioritization accuracy was not improved with less time.

(Devadas & Cholli., 2022) presented a novel method called the Interdependency-aware Qubit and BrownRoost Rank (IQ-BR) method to prioritize the large set of requirements. fairly good result had been achieved in in term of precisely and lessening the noise in a large set requirement prioritization. IQ-BR outperform CDBR, and IFS in term of accuracy which was detected to be 95%, 90%, and 91.66% respectively. However, it fails to address uncertainty and test suite execution issue among diverse stakeholders for large scale software requirements prioritization and the absent of supporting tool is also seen.

(Eldrandaly.,2023) presented a multi-criteria decision making (MCDM) framework for requirements prioritization, adopting the DEMATEL and TOPSIS methods in the neutrosophic environment. using the type-2 neutrosophic numbers (T2NNs) to compute and rank the criteria importance. The DEMATEL method used in the framework handles the interdependency between the requirements. Then the T2NN-based TOPSIS is used to rank the requirements. Lastly, the proposed frame work evaluated via case study. Good result achieved in addressing the fuzziness and vagueness in the stakeholders' decisions However, it needs to be tested on a large project for further validation.
As we noted in the previous works, all papers focus on handling the dependency between requirements when prioritizing the requirements as (Shao et al., 2017; Gupta & Gupta., 20١٨; Devadas & Cholli., 2022)] by using different techniques, concern with sequential prioritization only. There is no work focused on the impact on requirements prioritization on the software implementation time, building a model to determine the priority level (High, Medium, and low) and reduce the software implementation time. We develop a classification model concerning the parallel ranking in prioritization, which permit the simultaneous requirements implementation that reducing the implementation time. We aim to gain high rate of reducing the implementation time and other factors represented in determining priority level (high, Medium, and low), scalability, automation.

3. Research Model

In this research, we developed a machine learning model, which containing data elicitation and preprocessing and three main phases, namely: Phase 1: Initial weight for prioritization criteria assignment provided by both stakeholders and developers; Phase 2 : Introduce the concept of parallel ranking in requirements prioritization to reduce the software implementation time through simultaneous implementation; and finally, Phase 3: A classification technique using Random Forest-based MoSCoW Method (RF-MM). We applied our model to (Testcase MIS system with priority) industrial dataset. Figure 1 illustrates our proposed model

3.1 The Proposed Model

In the research, we will develop a machine learning model for determines the requirements prioritization; figure 1 illustrates our proposed model. The first step beginning with data elicitation and preprocessing; Then the stakeholders and developers provided the initial priory for each requirement. After that we apply the MoSCoW method and the concept of parallel prioritization; Finally, we will apply a classification technique called Random Forest classifier based MoSCoW method. Therefore, a research model designed for precisely determine the RP in parallel format, from requirements document. The following steps explain how our proposed model work:

1. Elicit the requirements in term of software requirements specification(SRS) document.
2. Prepare the software requirements specification document.

3.Stakeholders and developer prioritization

    3.1. Pick score for each requirement according to prioritization criteria.
    3.2. Identify the dependency between requirements criteria.
    3.2. Calculate score for each prioritization criteria.
    3.3. Rank the requirements.

3. Firstly, apply MoSCoW method

    3.1. Classify the requirements into (Must, Should, Could, and Wouldn't).

3. Secondly, apply MoSCoW Parallel Prioritization

    3.2. Classify each MoSCoW class to priority level (High, Medium, and Low).
    3.3. Ensure the requirements dependency.
    3.4. Rank the requirements in parallel format when they in the same MoSCoW class with the same priority level and no dependency between them.
    3.5. Drop Wouldn't category.
    3.5 Calculate implementation time.

4. lastly, apply Random Forest classifier algorithm.

    4.1. Specify the values in the requirements document as:
        4.1.1. prioritization criteria (Cost, Time, Complexity and Importance).
        4.1.2. MoSCoW method categories and levels.

5. Final prioritized list in parallel format with less implement time.

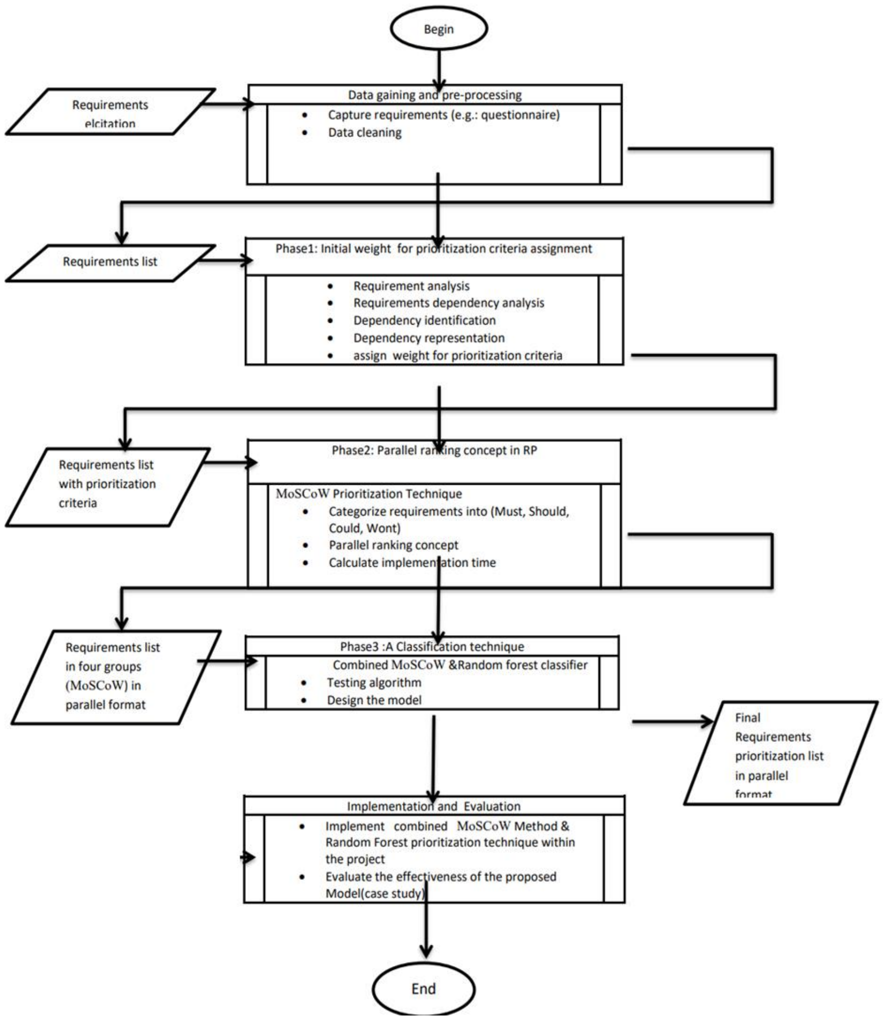

Begin
Requirements elcitation
Data gaining and pre-processing
Capture requirements (e.g.: questionnaire)
Data cleaning
Requirements list
Phase1: Initial weight for prioritization criteria assignment
Requirement analysis
Requirements dependency analysis
Dependency identification
Dependency representation
assign weight for prioritization criteria
Requirements list with prioritization criteria
Phase2: Parallel ranking concept in RP
MoSCoW Prioritization Technique
Categorize requirements into (Must, Should, Could, Wont)
Parallel ranking concept
Calculate implementation time
Requirements list in four groups (MoSCoW) in parallel format
Phase3 :A Classification technique
Combined MoSCoW &Random forest classifier
Testing algorithm
Design the model
Final Requirements prioritization list in parallel format
Implementation and Evaluation
Implement combined MoSCoW Method & Random Forest prioritization technique within the project
Evaluate the effectiveness of the proposed Model(case study)
End

Figure 1. Proposed Model.

3.2 Model Concepts

Stakeholders indicate ones who will describe the look of organization and will document the requirements (Gupta & Gupta., 20١٨). Stakeholders' emphasis customer satisfaction in term of urgency, needs, and business values (Keertipati et al., 2016; Gupta & Gupta., 20١٨; Gambo et al., 2021). Developer denotes engineers who develop or maintain related systems (Ahmad et al., 2022). Developers are concerned with project features such as effort and cost (Keertipati et al., 2016; (Devadas & Cholli., 2022). The MoSCoW abbreviation was created by (Clegg &. Baker), who in 1994 proposed the classification of requirements into Must Have, Should Have, Could Have and Won't Have. MoSCoW stands for "Must have," "Should have," "Could have," and "Won't have" prioritization categories. The four MoSCoW categories are defined as follows:

1. Must-Have (M): Identify the requirements that are absolutely critical for the success of project. These are non-negotiable and form the core of the project. Focus on defining and prioritizing the requirements that are essential for achieving the project's primary objectives.
2. Should-Have (S): List the requirements that are important but not as critical as the Must-Have items. These requirements significantly contribute to the project's success, but their omission would not be catastrophic. Prioritize them based on their importance, impact, and alignment with project goals.
3. Could-Have (C): Include requirements that are desirable but not essential for the project's immediate success. These are typically nice-to-have features or functionalities that can enhance the project but can be deferred if necessary. Prioritize these based on their potential to add value or improve the project.

4. Won't-Have (W): This category includes requirements that are explicitly excluded from the current project scope. These are items that are not essential or beneficial to the current project and can be considered for future iterations or separate projects.

MoSCoW considers the best prioritization technique when applying on huge set of requirements because it was the easiest, fastest and provide the highest user interference.

Definition 1. We define parallel ranking (PR), as a requirements Req in a particular class in MoSCoW method with the same priority level, and no dependency relation with other requirements, then possible set of requirements will be implemented simultaneously or in parallel manner. Given three requirements A, B and C, such that (A, B, C) exist in Must class in MoSCoW, and in high level then:

The three requirements A, B and C cloud be implemented at the same time. This could result in reducing the software implementation time. The whole process of prioritization is summarized in Figure 2.

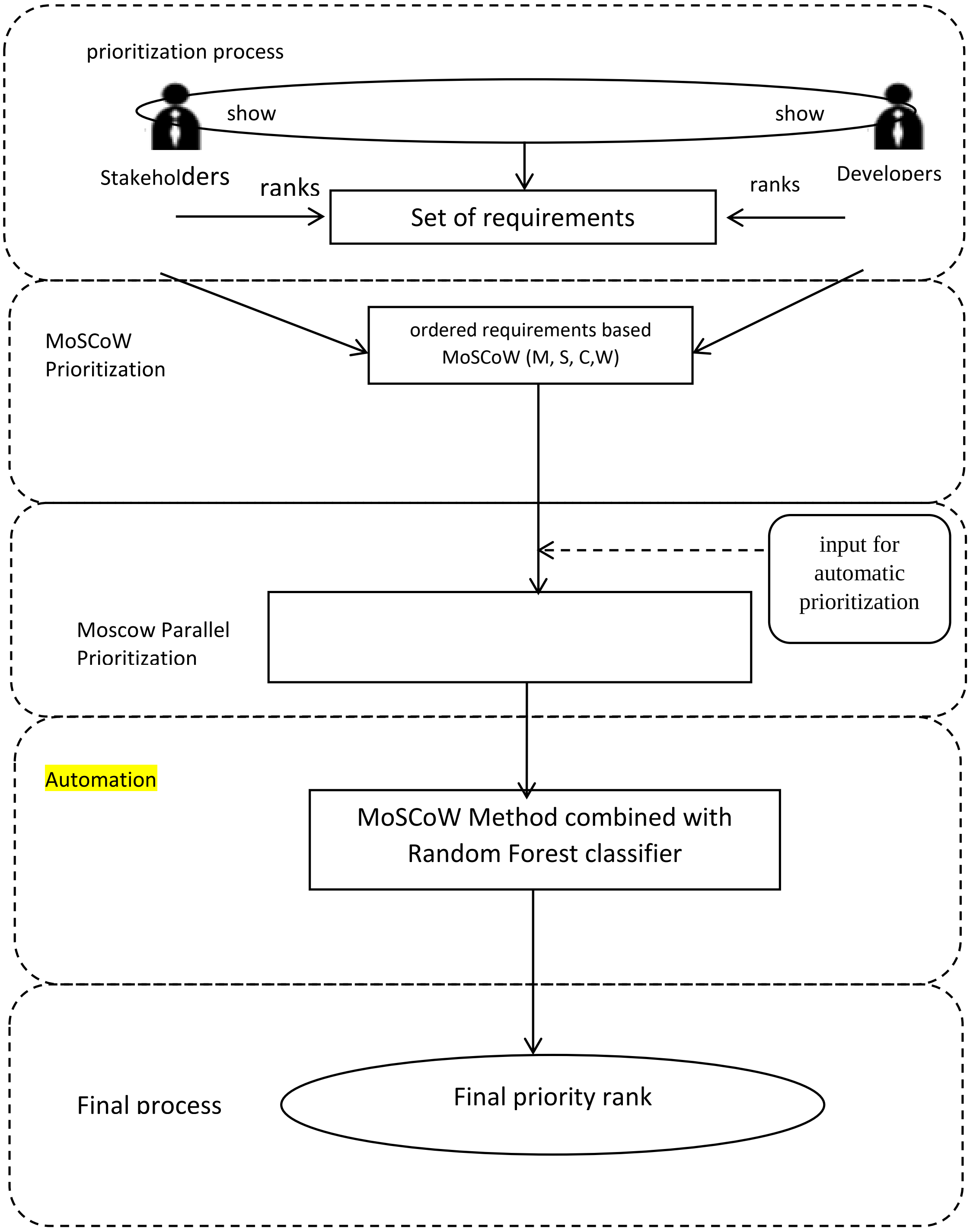

prioritization process
show
show
Stakeholders
ranks
Set of requirements
ranks
Developers
MoSCoW Prioritization
ordered requirements based MoSCoW (M, S, C,W)
input for automatic prioritization
Moscow Parallel Prioritization
Automation
MoSCoW Method combined with Random Forest classifier
Final process
Final priority rank

Figure 2. Prioritization process.

3.2.1 Stakeholder and developer's initial priority assignment

The stakeholders will pick a weight for each prioritization criteria. The prioritization criteria for the proposed model include MoSCoW, cost, time, complexity and importance. The proposed model prioritization criteria are represented in Table 1. The total weight will select by stakeholders for all requirements criteria will be 100 percent. Table 2 shows an example of criteria weight for the proposed requirements prioritization model.

Table 1: Prioritization criteria and their description

Criteria	Description
MoSCoW	Measure the impact of requirements using MoSCoW Method
Cost	Measure the cost required in developing the application
Time	Measure the time required to develop application
Complexity	Measure the requirements complexity
Importance	Measure the importance of requirements.

Table 2: Example of criteria weight for requirements prioritization

Criteria	Weight %
MoSCoW	10%
Cost	15%
Time	20%
Complexity	25%
Importance	30%
Total	100%

Developers adopt matrix based on dependence graph to represent dependency among requirements. The stakeholders and developer's initial priority are considered as ''perfect" preferences. These preferences will use as guide for estimation in MoSCoW ranking.

3.2.2 MoSCoW Prioritization

Within the stakeholder and developer's initial ranking in mind, MoSCoW method will progress towards identify and classify the requirements consequently. MoSCoW represents a guideline to confirm the basic importance of the requirements. The key stakeholders distribute the requirements into four categories (Must, Should, Could and Wouldn't) and three level (High, Medium, and Low) by applying MoSCoW method. The rating method for MoSCoW method is depending on the criteria (e.g: 1 - the maximum important priority, 4 - the minimum important priority) stated in Table3

Table 3: Rating criteria for MoSCoW Method

Criteria	Rate
Must	1
Should	2
Could	3
Won't	4

The stakeholders and developers will then use numerical assignment techniques to score the rating for further prioritization criteria as defined in (Chua et al., 2022), table 4 shows rating criteria. The rating method for these criteria is based on the criteria in table 4 with the scale of 1 to 10 (e.g: 1 - the fewer important priority, 10 - the greatly important priority). The rating method for importance criteria is based on the criteria (e.g: 1 - the very important priority, 3 - the less important priority) (Chua et al., 2022) in Table.5.

Table 4: Rating criteria for cost, time, and complexity

Criteria	**Rating**
Cost	1-10
Time	1-10
Complexity	1-10

Table 5: Rating criteria for importance

Criteria	**Rating**
High	3
Medium	2
Low	1

The prioritization part will begin with computing the rate based on the initial set of prioritized requirements by the stakeholders and developers. The weight criteria and rating method recognized previously will be used in computing the rate for requirements prioritization. The formula for computing each criteria of the requirement are defined as equations defined (Chua et al., 2022) (1), (2), (3), (4), and (5).

$$MoSCoW \rightarrow score/\,4 * moscow\_weight \qquad (1)$$

$$Cost \rightarrow score\ /\ 10 * cost\_weight \qquad (2)$$

$$Time \rightarrow score\ /\ 10 * time\_weight \qquad (3)$$

$$Complexity \rightarrow score\ /\ 10 * complexity\_weight \qquad (4)$$

$$Importance \rightarrow score\ /\ 3 * importance\_weight \qquad (5)$$

The total rates for requirements criteria will be computed using equation (6) when the criteria of each requirement are calculated.

$$Total\ Score \rightarrow MoSCoW + Cost + Time + Complexity + Importance \ (6)$$

3.2.3 MoSCoW Parallel Prioritization

Parallel prioritization concepts implemented when the requirements in the same MoSCoW category and the same priority level with no dependencies. For example, if Req2, Req3, and Req4 in are ranked according to MoSCoW method in (Must) category, and the same priority level(high) with no dependencies between them. Therefore, it could be implemented simultaneously to reduce the implementation time, which may contribute in accelerating the other higher level in software development. Then calculate implementation time: To confirm the hypothesis of simultaneous implementation of the requirements in parallel format could reduce the implementation time, the model must calculate implementation time using mathematical equations.

To calculate the implementation time, the time must be estimated for every requirement. The estimated time allocated for each requirement in each category (Must Have, Should Have, Could Have and

Wouldn't), based on required resources like complexity and cost. Then the (star_ time and finish_ time) must be estimated for each MoSCoW category as following:

In Must category(M) the star_ time==0, finsh_time ==finsh_time_m

In Should category(S) the star_ time== finsh_time_m, finsh_time== finsh_time_s

In Could category(C) the star_ time== finsh_time_s, finsh_time== finsh_time_C.

The Wouldn't category could be dropped. Calculate the total time for each Category through making summation of the estimated times for all requirements in each category (Must Have, Should Have, Could Have). The requirements within the same MoSCoW category and with the same priority levels when no dependencies can be considered for simultaneous implementation to reduce the implementation time.

3.2.4 A classification technique using Random Forest-based MoSCoW Method (RF-MM)

The classification process utilizes machine learning techniques in generating the priority levels and final priority ranks of requirements which contains two parts, prioritization and prediction. Specifically, Random Forest classifier -based MoSCoW Method (RF-MM) is used. The purpose of this classification model is to predict the priority levels. The second part of the proposed RF-MM model automatic prioritization is prediction of requirements priority level. The final priority rank will be calculated as a result represented as output of Random Forest-based MoSCoW Method (RF- MM). It uses these MoSCoW ranking for estimate of its final output concluded series of iterations. Classification done using and 70% split (i.e. 70% training, 30% testing). The training data consists of 70% of the original data (current requirements) and the other 30% of the data (new requirements) will be used as testing data.

3.3 Implementation: (The Proposed Model Random Forest classifier based MoSCoW method)

Our proposed model contains two techniques the first one concerned with dividing the requirements into four categories according to MoSCoW method namely Must have, should have, could have and Wouldn't have. The second techniques use the output of the first techniques to determine the final priority of the requirements in the document in parallel format.

Our proposed model uses the output of the MoSCoW parallel prioritization as input to determine the final priority of the requirements in the document in parallel format. There are several steps illustrated as following:

Preprocessing step: This step involves removing the noisy data of the requirements features, handling the missing values of the requirements features in order to attain optimal classification results. preprocessing concerns with the occurrence of the null values. null values is an inherent problematic in many real datasets, particularly when dealing with large datasets. Therefore, it vital to handle the null values (fill in the values or eliminated if not important), or solving the variations in the data before it is transformed into a comprehensible format. Once the requirements are pre-processed, the feature sets from the requirements features set are created. Finally, the extracted feature vectors are fed to classifier methods for testing.

Training: Train a Random Forest classifier using the Testcase MIS system with priority dataset, where the input features are the attributes of requirements, and the target variable is the priority level (High, Medium, and Low) according to MoSCoW method category and the implementation time estimation.

We built these classifiers using the Scikit-Learn library, a well-established machine learning package for Python. Scikit-Learn offers numerous utility functions for data preprocessing, model validation, and metric computations. The goal of our study is to achieve the best combination of the MoSCoW method and the Random Forest classifier as supervised learning techniques for prioritizing and classifying software requirements, with an added focus on estimating implementation time. The algorithm for our model is shown in Figure 3.

Testing and Prediction: Once the model is trained and evaluated, I can use it to predict the priority levels of new requirements based on their attributes.

Classification Model algorithm

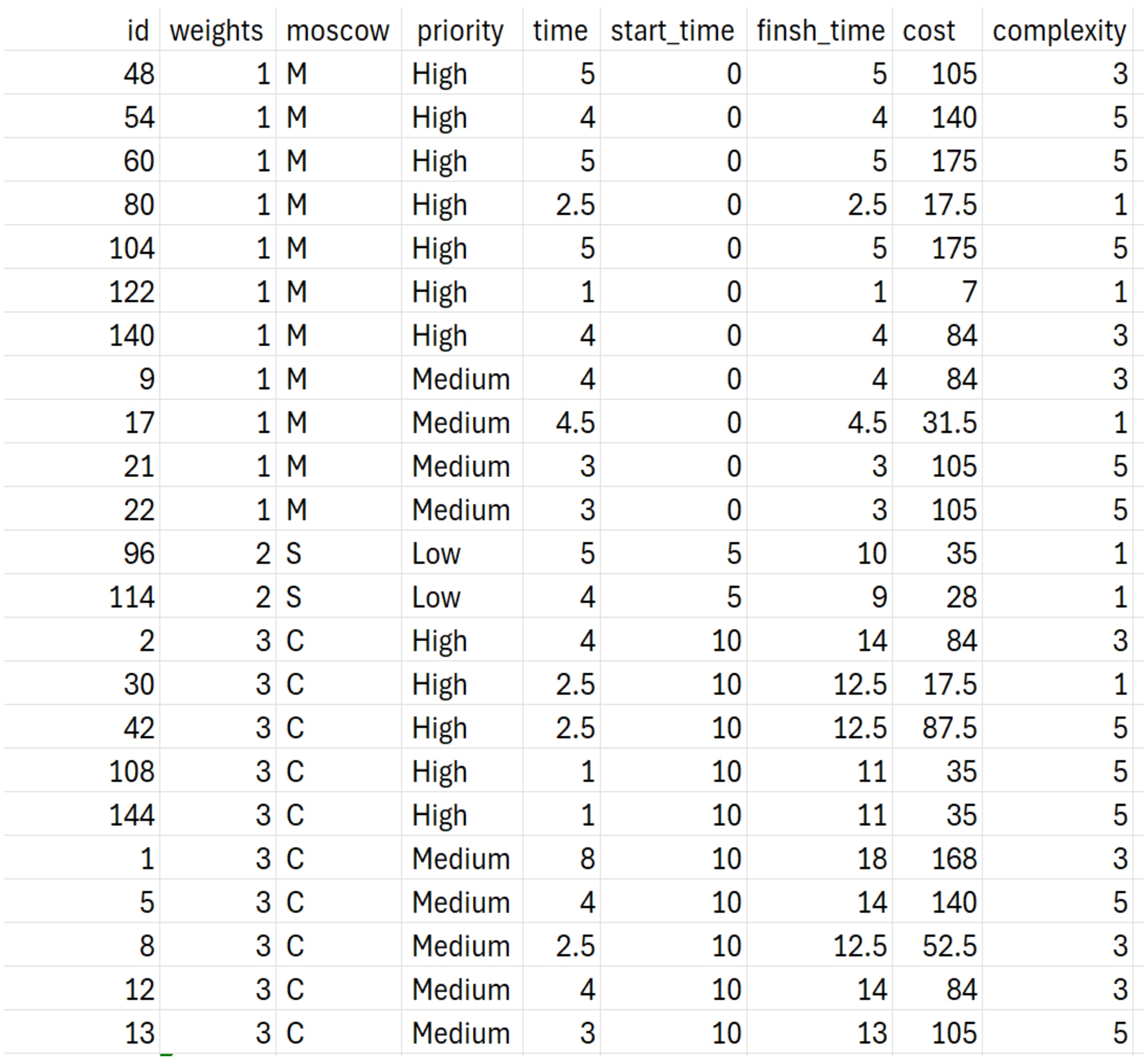
```
Input: the data set obtained from our MoSCoW parallel prioritization
Output: Pattern as values
Input prioritization criteria values (Cost, Time, Complexity and Importance).
Find pattern as values using Regular Expressions Array
If Importance is 1 Then
Req in Must MoSCoW_category == 'M'
if priority == 'High':
m_h.allocted(req) in array
elif priority == 'Medium':
m_m. allocted(req) in array
elif priority == 'Low':
m_l. allocted(req) in array
Else IF Importance is 2 Then
Req in Should MoSCoW_category == 'S'
if priority == 'High':
  s_h. allocted(req) in array
elif priority == 'Medium':
  s_m. allocted(req) in array
elif priority == 'Low':
s_l.allocted(req) in array
Else IF Importance is 3 Then
Req in Could MoSCoW_category == 'C'
if priority == 'High':
c_h. allocted(req) in array
elif priority == 'Medium':
c_m. allocted(req) in array
elif priority == 'Low':
c_l. allocted(req) in array
Else IF Importance is 4 Then
if priority == 'High':
w_h. allocted(req) in array
elif priority == 'Medium':
w_m. allocted(req) in array
elif priority == 'Low':
w_l. allocted(req) in array
calculate the total implementation time for requirements within each MoSCoW category
considering dependencies
Drop Wouldn't category
for each MoSCoW category  (M, S, C):
```

```
if No of Req in the same MoSCoW class with the same priority level & No dependency among them:
Rank them in parallel format.
END.
```

Figure 3. Classification model algorithm

3.3 Model Evaluation

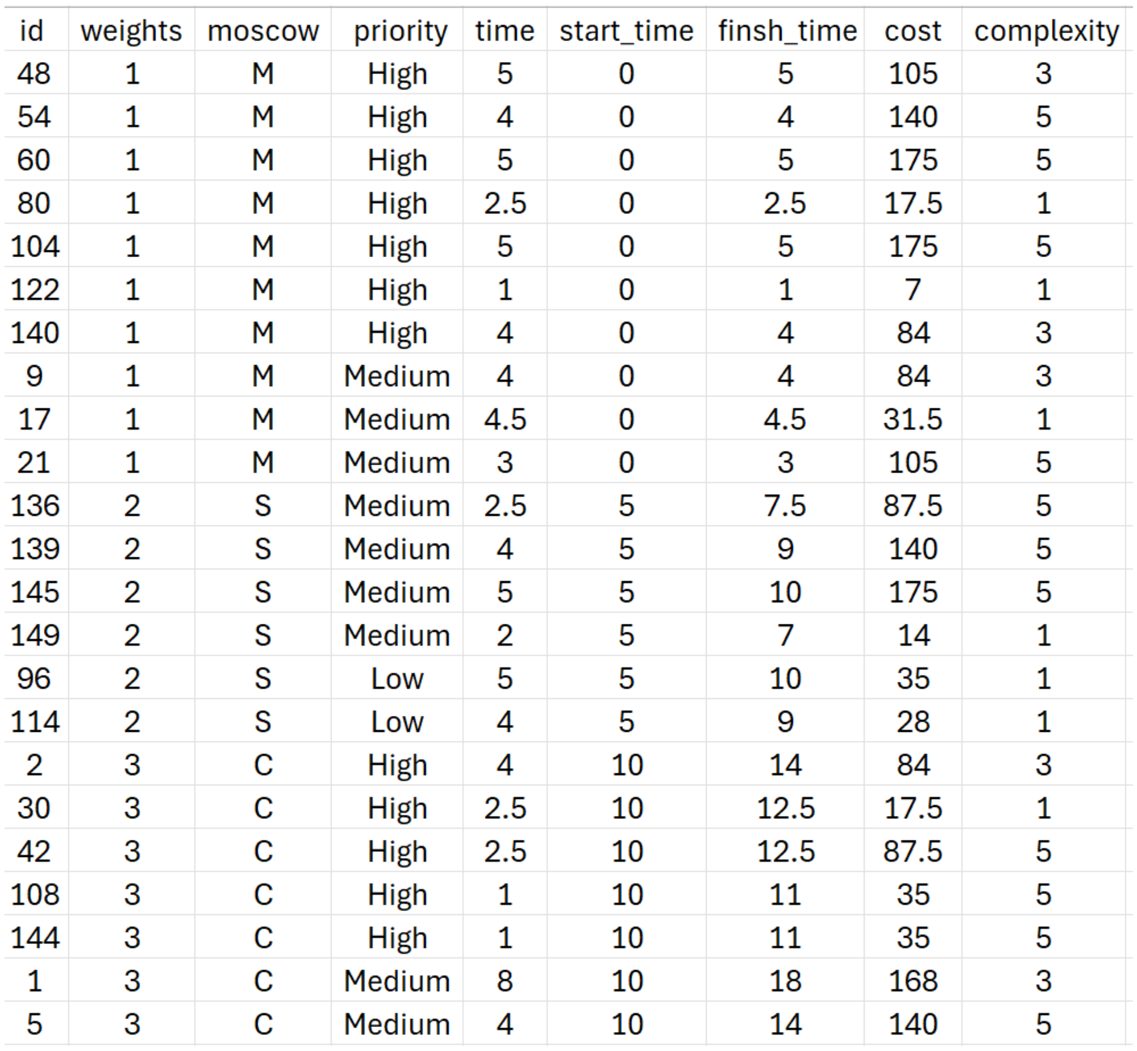
To confirm the hypothesis that implementing requirements in parallel format can reduce implementation time, the model calculates this time using mathematical equations. We computed the total implementation time for the "Must," "Should," and "Could" categories according to the MoSCoW method and compared the results. Figures 4 and 5 illustrates these findings.



id	weights	moscow	priority	time	start_time	finsh_time	cost	complexity
48	1	M	High	5	0	5	105	3
54	1	M	High	4	0	4	140	5
60	1	M	High	5	0	5	175	5
80	1	M	High	2.5	0	2.5	17.5	1
104	1	M	High	5	0	5	175	5
122	1	M	High	1	0	1	7	1
140	1	M	High	4	0	4	84	3
9	1	M	Medium	4	0	4	84	3
17	1	M	Medium	4.5	0	4.5	31.5	1
21	1	M	Medium	3	0	3	105	5
22	1	M	Medium	3	0	3	105	5
96	2	S	Low	5	5	10	35	1
114	2	S	Low	4	5	9	28	1
2	3	C	High	4	10	14	84	3
30	3	C	High	2.5	10	12.5	17.5	1
42	3	C	High	2.5	10	12.5	87.5	5
108	3	C	High	1	10	11	35	5
144	3	C	High	1	10	11	35	5
1	3	C	Medium	8	10	18	168	3
5	3	C	Medium	4	10	14	140	5
8	3	C	Medium	2.5	10	12.5	52.5	3
12	3	C	Medium	4	10	14	84	3
13	3	C	Medium	3	10	13	105	5

total Moscow: 218.75
total cost: 2392.95
total time: 115.15
total complexity: 102.5
total importance: 291.66
Time: 33.0

Figure 4. parallel ranking implementation



id	weights	moscow	priority	time	start_time	finsh_time	cost	complexity
48	1	M	High	5	0	5	105	3
54	1	M	High	4	0	4	140	5
60	1	M	High	5	0	5	175	5
80	1	M	High	2.5	0	2.5	17.5	1
104	1	M	High	5	0	5	175	5
122	1	M	High	1	0	1	7	1
140	1	M	High	4	0	4	84	3
9	1	M	Medium	4	0	4	84	3
17	1	M	Medium	4.5	0	4.5	31.5	1
21	1	M	Medium	3	0	3	105	5
136	2	S	Medium	2.5	5	7.5	87.5	5
139	2	S	Medium	4	5	9	140	5
145	2	S	Medium	5	5	10	175	5
149	2	S	Medium	2	5	7	14	1
96	2	S	Low	5	5	10	35	1
114	2	S	Low	4	5	9	28	1
2	3	C	High	4	10	14	84	3
30	3	C	High	2.5	10	12.5	17.5	1
42	3	C	High	2.5	10	12.5	87.5	5
108	3	C	High	1	10	11	35	5
144	3	C	High	1	10	11	35	5
1	3	C	Medium	8	10	18	168	3
5	3	C	Medium	4	10	14	140	5

```
total Moscow: 218.75
total cost: 2392.95
total time: 115.15
total complexity: 102.5
total importance: 291.66
Time:  76.0
```

Figure 5. Sequential ranking implementation

4. Results Discussion

Experiments on automated requirements priority detection were conducted using the Testcase MIS system with a priority dataset. We trained the requirements prioritization model on 1,314 requirements within this dataset and observed a promising reduction in implementation time. Specifically, parallel ranking reduced implementation time to 33 seconds, compared to 76 seconds in a sequential setup. This demonstrates that parallel ranking can cut implementation time by more than half, proving its effectiveness in optimizing requirements prioritization.

Table 7: Study Comparisons

	CDBR(2018)	**SARiP (2022)**	**IQ-BR(2022)**	**Proposed Model RF-MM (2024)**
Automation	semi-automated	semi-automated	Full-automated	Full-automated
Prioritization criteria	Stakeholders and developer's Communication, Dependency. Scalability, and Importance	Stakeholders prioritization, Cost, Time, Complexity, Risk, and Importance	Customer prioritization, Dependency Accuracy, Prioritization time.	Stakeholders and developer's criteria, Dependency, Cost, Time, Complexity, Importance, and Scalability
No of input Requirements	100	Not estimated	60	>149
Scalability	Considered	Not Considered	Considered	Considered
Parallel RP	-	-	-	Considered
Awareness with the effect of RP on software developments cycle	-	-	-	Considered
Implementation time	-	-	-	Considered

5.Conclusion

In this paper, we designed a machine learning model integrated with MoSCoW method, capable of determining requirement priority levels in a parallel format, significantly reducing implementation time. When measuring this reduction, we found that sequential ranking required 76 seconds, while parallel ranking reduced this to 33 seconds. This demonstrates that parallel ranking can cut implementation time by more than half. The model was applied to a Testcase MIS system with a priority dataset, providing valuable insights into the impact of priority ranking on software development cycles. Based on our findings, this research represents a step forward in computational intelligence. Future work will evaluate the model's performance in classification and priority level determination.

6. ACKNOWLEDGMENT
This work was supported by the Deanship of Scientific Research at Omdurman Islamic University, Omdurman, Sudan.
FUNDING:

"This research received no external funding"

7. REFERENCES

1. Aurum, A. (2005). Engineering and managing software requirements (Vol. 1). C. Wohlin (Ed.). Heidelberg: Springer.
2. Dabbagh, M., & Lee, S. P. (2013, July). A consistent approach for prioritizing system quality attributes. In 2013 14th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (pp. 317-322). IEEE.
3. Bukhsh, F. A., Bukhsh, Z. A., & Daneva, M. (2020). A systematic literature review on requirement prioritization techniques and their empirical evaluation. Computer Standards & Interfaces, 69, 103389.

4. Sher, F., Jawawi, D. N., Mohammad, R., Babar, M. I., Kazmi, R., & Shah, M. A. (2020). Multi-aspects Intelligent Requirements Prioritization Technique for Value Based Software Systems. In Intelligent Technologies and Applications: Second International Conference, INTAP 2019, Bahawalpur, Pakistan, November 6-8, 2019, Revised Selected Papers 2(pp. 357-371). Springer Singapore.
5. Barenkamp,M., Rebstadt, J., & Thomas, O.(2020). Applications of AI in classical software engineering. AI Perspectives, 2(1):1.
6. Achimugu, P., Selamat, A., Ibrahim, R., & Mahrin, M. N. R. (2014). A systematic literature review of software requirements prioritization research. Information and software technology, 56(6), 568-585.IEEE.
7. Shao, F., Peng, R., Lai, H., & Wang, B. (2017). DRank: A semi-automated requirements prioritization method based on preferences and dependencies. Journal of Systems and Software, 126,141-156.
8. Gupta, A., & Gupta, C. (2018). CDBR: A semi-automated collaborative execute-before-after dependency-based requirement prioritization approach. Journal of King Saud University-Computer and Information Sciences, 34(2), 421-432.
9. Chua, F. F., Lim, T. Y., Tajuddin, B., & Yanuarifiani, A. P. (2022). Incorporating semi-automated approach for effective software requirements prioritization: A framework design. Journal of Informatics and Web Engineering, 1(1), 1-15.
10. Alrashoud, M. and Abhari, A. (2015) Intelligent Automation & Soft Computing Perception-Based Software Release Planning, Intell. Autom. Soft Comput., vol. 21, no. 2, pp. 175–195, doi: 10.1080/10798587.2014.960229.
11. Allex, A., Matheus, A., I. Yeltsin, Dantas, A., and J. Souza, (2016). An Architecture based on interactive optimization and machine learning applied to the next release problem, Autom. Softw. Eng., vol. 24, no. 3, pp. 623–671, doi: 10.1007/s10515-016-0200-3.
12. Gupta, A., & Gupta, C. (2018, August). Towards dependency based collaborative method for requirement prioritization. In 2018 Eleventh International Conference on Contemporary Computing (IC3) (pp. 1-3). IEEE.
13. Misaghian, N., Motameni, H., & Rabbani, M. (2019). Prioritizing interdependent software requirements using tensor and fuzzy graphs. Turkish Journal of Electrical Engineering and Computer Sciences, 27(4), 2697-2717.
14. Abbas, M., Inayat, I., Jan, N., Saadatmand, M., Enoiu, E. P., & Sundmark, D. (2019, December). Mbrp: Model-based requirements prioritization using pagerank algorithm. In 2019 26th Asia-Pacific Software Engineering Conference (APSEC) (pp. 31-38). IEEE.
15. Eldrandaly, B. K. (2023). A Framework of Type-2 Neutrosophic for Requirements Prioritization. Neutrosophic Sets and Systems, 53, 421-432.
16. Qayyum, S., & Qureshi, A (2018, November). A survey on machine learning based requirement prioritization techniques. In Proceedings of the 2018 International Conference on Computational Intelligent Systems (pp. 51-55).
17. Jan, N., Inayat I., and Abbas, M. (2020). An Empirical Evaluation of Requirements Prioritization Techniques. Marketing and Branding Research,7 (1), 11.
18. Hujainah, F., Bakar, R.B.A., Al-haimi, B.& Abdulgabber,M.A.(2018). Stakeholder quantification and prioritisation research: A systematic literature review.Information and Software Technology ,102, 85–99, https://doi.org/10.1016/j.infsof.2018.05. 008.
19. Hudaib,A., Masadeh, R., Qasem, M.H., & Alzaqebah, A.(2018). Requirements Prioritization Techniques Comparison. Modern Applied Science, 12(2), 62.
20. Amelia, T., & Mohamed, R. (2022). A Review: Requirements Prioritization Criteria Within Collaboration Perspective. Journal homepage: http://www.pertanika.upm.edu.my/.
21. Ahmad, S., Rizawanti, R., Woodings, T., & Jalil, I. E. A. (2022). MCBRank Method to Improve Software Requirements Prioritization. International Journal of Advanced Computer Science and Applications, 13(7).

22. Keertipati, S., Savarimuthu, B. T. R., &. Licorish. S. A. (2016, June). Approaches for prioritizing feature improvements extracted from app reviews. In Proceedings of the 20 th international conference on evaluation and assessment in software engineering (pp. 1-6).
23. Gambo, I. P., Ikono, R., Iroju, O. G., Omodunbi, T. O., & Zohoun, O. K. (2021). Hybridized ranking model for prioritizing functional software requirements: Case study approach. International Journal of Software Innovation (IJSI), 9(4), 19-49.