Using Interdependencies for the Prioritization and Reprioritization of Requirements in Incremental Development

Aryaf Al-Adwan[1]
Department of Computer and Networks Engineering
Faculty of Engineering Technology
Al-Balqa Applied University
Amman, Jordan

An'aam Aladwan[2]
Department of Management Information Systems
Al-Ahliyya Amman University
Amman, Jordan

***Abstract*—There is a growing trend to develop and deliver the software in an incremental manner; to achieve greater consistency in the developed software and better customer satisfaction during the requirement engineering process. Some of the developed increments in the incremental model will be delivered to consumers and run in their environments, so a set of these requirements are evaluated, introduced, and delivered as the first increment. Other requirements are delivered as the next step and so on for the next increment. The priority of requirements plays an important role in each increment, but it is precluded by the interdependences between the requirements and resources constraints. Therefore, this paper introduces a model for requirements prioritization and a reprioritization based on these important factors. The first one is the requirement interdependencies which are described as a hybrid approach of tractability list and directed acyclic graph, and the second factor is the constraints of the requirements resources that are used based on the queuing theory for requirements reprioritization. In order to achieve this, two algorithms namely; Priority Dependency Graph (PDG) and Resources Constraints Reprioritization (RCR), were proposed with a linear time complexity and implemented via a case study.**

Keywords*—Requirement engineering; incremental model; requirement prioritization; requirement interdependencies; dependency graph; queuing theory

I. Introduction

In the incremental software model, small releases of the software are implemented in a series mode instead of providing the whole system after a long period of development. This model efficiently impacts the prioritization of requirements in such a way that the most relevant requirements can be introduced in the system's first releases. On the other hand, later phases are left with less important requirements. When requirements are elicited, a large number of them are often created, which is very difficult to implement them at the same time [1]. This is due to the market impact, the user persisting to have the software finished, and the limitations on cost and staff. Therefore, the requirements need to be prioritized in such a way that the earliest product releases meet the most critical ones, particularly when an incremental model is used, where the product is designed, implemented, and tested incrementally until the product is completed.

The requirements affect each other and are related to each other during software development in a way that prevents treating them separately. This is referred to as the interdependence between requirements. Consequently, requirement interdependency concerns about the relationships between requirements which during software development will influence decisions and activities. This play an important role in the prioritization of requirements, especially with incremental development which requires a careful selection of requirements that meet the growth of the various increments. Choosing one requirement may therefore activate the selection of several other requirements that rely on it.

On the other hand, the literature that discusses the interdependence requirements, limited work has been carried out. Interdependence of requirements is a special form of traceability of requirements that defines the relationships between different requirements. The traceability list, which is a table of relationships describing the dependencies between requirements [1], is one of the techniques for representing requirement interdependencies.

Carlshamre [2] used the directed graph (digraph) to represent the interdependencies between requirements, as well as classifying the interdependencies into five relationships; and, REQUIRES, TEMPORAL, CVALUE, ICOST, and OR, which visualized later by the directed graph. The concern in this approach wasn't to visualize types of interdependencies rather than representing them as dependency graph and apply an algorithm to prioritize the requirements. Another way to represent the interdependencies is to use ontology-based representation and a formal graphical representation to visualize the requirements interdependencies in a proper way [3].

It is possible to prioritize requirements, taking into account several different aspects, such as importance, cost, penalty, time, risk and dependencies [4]. The literature is full of several prioritization strategies for requirements. These include the process of analytical hierarchy (AHP) which is the most common priority-based technique that is designed to permit decision-makers to set priorities and decide the correct decision. Initially, AHP specifies the parameters and substitutes for each requirement and uses them to construct a

hierarchy to activate pair-wise comparisons; then the users can determine their favorites for each pair of attributes by assigning a decision scale. However, this technique requires a quadratic time to prioritize the requirements and suffers from scalability issues particularly when the number of requirements increases [5]. The creation of binary search tree, in which each requirement is shown in a node. The tree needs to be prioritized; the low priority requirements are set on the left side of the tree and high priority needs are positioned on the right. Although this method is fast but it the comparison of BST is typical, only showing which requirement is more desirable [6]. Another technique is the bubble sort in which the principle is similar to AHP, where they both make use of the comparison operation pair-wise and they require a time complexity of n^2. The distinction between them is that it is only possible for the decision-maker to consider which requirement is more significant between the compared requirements in bubble sort. [7]. Cumulative voting or the 100-dollar test is a straight-forward process that gives the system's stakeholders 100 units to be divided between requirements. The higher unit requirement has a higher priority and the lower unit requirement has a lower priority. The stakeholder controls distribution process of these units based on the priority of the requirements. However, if there are quite several requirements, this approach has a downside, because this method will not work well and will calculate the prioritization in wrong way. Also, it can be difficult to be aware the quantity of units that must be allocated and those that must be left [8-9]. Spanning trees technique is similar to AHP, where they both make use of the comparison operation pair-wise, but uses the minimum spanning technique. This can be done by the use of spanning tree architecture in order to eliminate the redundant comparisons, consequently, reducing the total number of comparison. On the other hand, it is not efficient when the number of requirements is large [10]. Numerical assignment (grouping) which provides a scale to all requirements based on separating them into different groups. Each requirement will then be assigned to a 5-point scale to assess its significance, however this technique provides low rate of reliability as well as fault tolerance [11-12]. Wieger Method determines the priority of the requirement by dividing the value of the requirement by the amount of costs and the technological risks associated with its implementation, and by assessing its customer significance, by applying 1-9 scale, as well as its implications, if this requirement were not enforced. It has drawback in which the stakeholders can easily influence it to achieve their objective goals [13]. MoSCoW technique is focused on cooperation between analysts and stakeholders to group the requirements into four categories. The efficiency here is good, but human attempts are required with disagreements between analysts and the views of stakeholders, so this approach would therefore be rated as low scalability and other hybrid techniques [14]. Most of the algorithms mentioned previously require quadratic complexity, so for a large number of requirements, the efficacy of the method becomes poor. Several papers were proposed in the literature in order to compare these method [15-17, 22, 23]. The purpose of this paper is not to compare the different approaches, but to suggest a new algorithm for the prioritization of requirements. However, the proposed approach for prioritizing requirements in this paper differs from the previous methods in achieving linear time complexity as well as the ability to reprioritize the requirement based on the resources availability.

The need for reprioritization has emerged from the fact that despite the effort expended in order to prioritize the requirements, this would be influenced by the constraints of precedent and resources constraints [18]. Therefore, this paper aims to introduce a hybrid approach of Traceability list and Directed Acyclic Graph to represent the requirements interdependencies for the prioritization process. As well as introducing a new algorithm for reprioritizing the requirements based on the queuing theory.

As discussed earlier, there are many types of requirement interdependencies mentioned in the literature. So it is worth to mention that this paper, proposed the prioritization and reprioritization requirement algorithms irrespective of the types of interdependencies and the methods used to identify them between the requirements, which are beyond the scope of this paper. Instead of focusing on the types of interdependencies and how they are described in any software project, this paper focuses on prioritizing the requirements based on the proposed algorithms using the proposed dependency parameters.

The paper is structured as follows: section 1 is the introduction and related work, section 2 gives a description of requirements interdependencies, section 3 presents the proposed approach in requirement prioritization, section 4 illustrates the approach as a case study and section 5 is the paper conclusion.

II. Requirements Interdependencies

The requirements influence each other and are linked to each other in a way that prevents handling them separately. This can be referred to as the dependencies between requirements. Basically, these requirements can also affect the decisions and activities during the development of the software. Requirements can, for example, affect each other through implementation constraints, the cost of implementing other requirements, or the customer satisfaction [19]. This means that in order to make accurate decisions during the development process, it is important to study the interdependencies. Simply stated, requirement interdependencies mean that a dependent relationship exists between the requirements. For instance, it is safer to start developing Ri before Rj if the Rj requirement requires Ri to work.

III. Methodology

A model for the prioritization and reprioritization of requirements based on a hybrid approach of representations of requirements is introduced. Fig. 1. demonstrates this model, which consists of two phases: the phase of prioritization and the phase of reprioritization. The first step can be achieved by prioritizing the requirements using dependency graph, while the next step is used to reprioritize the requirements using the queuing theory. The requirements are presented as a traceability list and then as a dependency graph, as shown in Fig. 1. The dependency graph is subsequently regarded as an input to the Priority Dependency Graph (PDG) algorithm in

order to create a priority list of requirements. This list is then processed on the basis of the resources constraints in the system by the Resources Constraints Reprioritization (RCR) algorithm.

A. Dependency Constraints

Any software can be defined as a set of R requirements where R={R1, R2, ... ,Rn}. When an incremental model is followed, then the first increment is analyzed, implemented and delivered as a set of these requirements. Other requirements are delivered as the next step and so on for the next level. In each increment the priority of requirements is playing an important role, but it is often precluded by requirements interdependencies. The approach to prioritizing the requirements in this paper is therefore focused on the representation of dependencies between requirements using the dependency graph. First of all, a simple description of the dependency factors used is discussed below:

1) Dependency scope: The Dependency Scope determines the scope of the requirements according to their dependencies, two types are available:

A Requirement R2 is an External Dependent on requirement R1 if and only if:

a) Execution of R1 precedes execution of R2.

b) Execution of R1 implies execution of R2 in the future increment.

A Requirement R2 is an Internal Dependent on requirement R1 if and only if:

a) Execution of R1 precedes execution of R2.

b) Execution of R1 implies execution of R2 in the same increment.

2) Dependency volume: The dependency volume determines the number of requirements that are internal dependent on the current requirement.

3) Dependency intensity: The dependency intensity determines the degree of dependency for each requirement, two types are available:

a) "Loose Dependencies is defined as: it would be ok to continue task without awareness of dependencies but would be better with awareness" [20].

b) "Tight Dependencies is defined as: the successor task has to wait until all its precursor tasks finish, the failure of the precursor will block the successor" [20].

The first parameter indicates the two main types of interdependencies in our method. External dependency determines whether the requirement in the future increment is dependent on another requirement, and internal dependency determines whether the requirement in the same increment is dependent on another requirement, as shown in Fig. 2.

The precedence constraint defined in the first parameter illustrates the relationship between the requirements in terms of precedence where in any software; the requirements must be implemented before other requirements. Therefore, for all iterations of increments there must be an order in which the requirements are executed. Loose dependencies can be used when there is no strict use of dependency between requirements, such as some requirements in mobile applications that doesn't require awareness of the context of the mobile user, but if provided the software will behave in an efficient manner. On the other hand, tight dependency is for those requirements that must be executed before other requirements as is the case in most software applications.

As depicted in Fig. 2. , the requirements are represented as a dependency graph where a directed acyclic graph (DAG) is used to define the R requirements as vertices V and the precedence constraints as edges E. In order to calculate the priority for each requirement in each increment based on the dependency types mentioned earlier, a topological sorting [21] with slight modifications is then performed.

An example of a dependency graph is represented in Fig. 3. R1 and R9 do not have dependencies in this DAG, while vertices R2 to R10 are dependent on other vertices; R4, for instance, depends on R1. Note that R4 has a volume of dependency greater than R3 that influences its prioritization process.

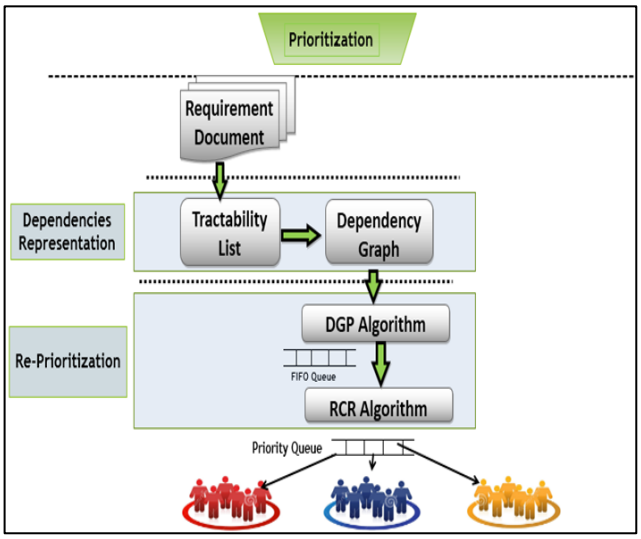

Fig. 1. The proposed Model.

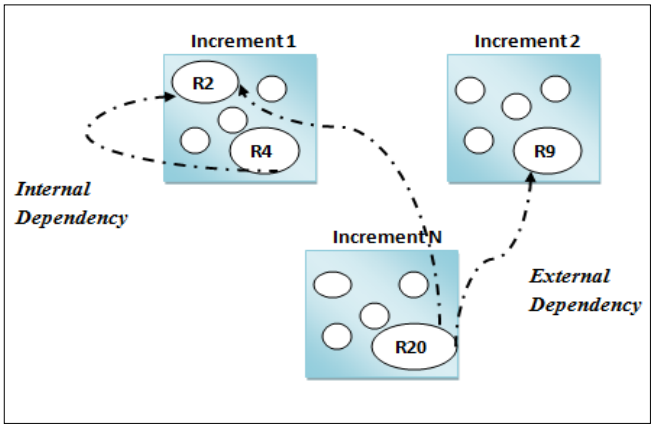

Fig. 2. External and Internal Dependencies.

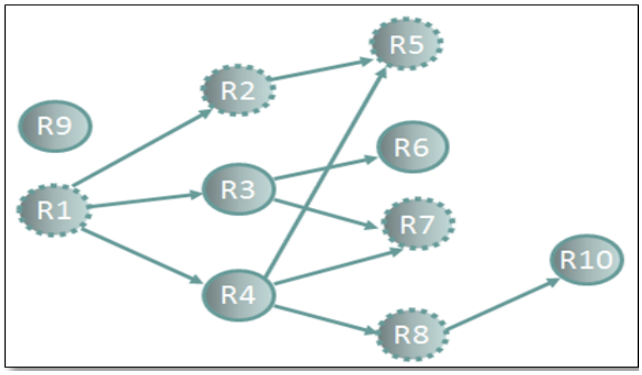

Fig. 3. Dependency Graph for Requirements.

Algorithm: Priority Dependency Graph (PDG) algorithm

Input: Digraph G = (V,E) ,Set of Requirements R as V, Set of Precedence Constraints as E, In_Degree array In_D, Out_Degree array Out_D, Dependency Scope array DS, Dependency Intensity array DI, Queue Q.

Output: list of Requirements R each associated with its priority

1: In_D ← { }
2: Store each vertex's InDegree in In_D array
3: Initialize Q with all in-degree zero vertices
4: While Q is not empty do
5 If DS for vertex v =1
6: Dequeue and output a vertex v
7: Set higher priority for v
8: Reduce In-Degree of all vertices adjacent to v by 1
9: Enqueue v which the In-Degree for it became zero
10: else
11: Dequeue and output a vertex v
12: Set lower priority for v
13: Reduce In-Degree of all vertices adjacent to v by 1
14: Enqueue v which the In-Degree for it became zero
15: repeat
16: end

Four arrays are used to calculate the priority in the proposed algorithm:

- InDegree array that contains the number of InDegree edges for each vertex.
- OutDegree array that contains the number of OutDegree edges for each vertex and represent the dependency volume.
- Dependency Scope array that determine the dependency scope for each vertex in the graph whether it is external or internal based on equation 1.
- Dependency Intensity that determine the intensity of the dependency for each vertex in the graph whether it is tight or loose based on equation 2.

The algorithm starts by initializing queue with all vertices that has zero InDegree, then while this queue has vertices in it.

$$\text{Dependencyscope} = \begin{cases} 1\ Internal\ Dependency \\ 0\ External\ Dependency \end{cases} \quad (1)$$

$$\text{DependencyIntensity} = \begin{cases} 1\ Tight\ Dependency \\ 0\ Loose\ Dependency \end{cases} \quad (2)$$

Higher priority will be granted to the vertices with internal dependency than those with external one; if two requirements are equal in the scope of dependency, then the decision can be made based on the volume of dependency. Such that, the requirements with tight dependency intensity will have higher priority for internal dependency than the requirements with loose dependency intensity; the algorithm will calculate the priority for those vertices until it is empty.

B. Complexity Analysis

Consider the complexity analysis of the proposed algorithm. Hence, a queue is used to store the vertices of zero InDegree. So, each time a node's InDegree is modified, we check if the value of it is 0 we add it to the queue, this will take O (|V|). Now to find a node of zero InDegree it takes O (1). The Dequeue operation will take O (|V|) while reducing the InDegree of all adjacent vertices to a vertex will take O (|E|). Therefore, the algorithm can be implemented to run in O (|V| + |E|) which is a linear running time. Note that most of the prioritization methods in the literatures have a quadratic complexity where our algorithm requires a linear one.

C. Resources Constraints Reprioritization Algorithm

As illustrated earlier, requirement prioritization is precluded by the available resources that are needed to develop the requirement or task, therefore the concept of the queuing theory is used to reprioritize and schedule the requirements to the available teams in the system. The method here is based on the outcome from the previous stage where the requirements are prioritized and added along with their priorities to a list or queue. The service facility may have of one or more teams. So, a requirement at the head of the queue can go to any team that is free. If there is more than one team, then a concurrent development will take place.

Each requirement in the queue must have the following characteristics:

1) Arrival Time λ to the queue for each requirement.
2) Waiting Time wt for each requirement, which indicates the waiting time for each requirement.
3) Status S: either FREEZE or INPROCESS. This parameter used to freeze the requirement and their dependent requirements.
4) Old Priority P_{old}, this parameter is used to indicate the old priority for the requirenment.
5) New Priority P_{new} , this parameter is used to indicate the new priority for the requirenment.
6) Available Resources Flag (ARF), this parameter will be used to indicate whether the resources are available to perform the requirement.

The RCR Algorithm involves the following steps:

1) Select the requirement with the minimum λ.
2) Check ARF whether it is set or not.
3) If ARF is equal to zero, then change the state S of this requirement to FREEZE and increment wt by 1.

4) If ARF is equal to 1 then change the state to IN_PROCESS and change Pold to Pnew and schedule it to the available team.

5) Each time check the ARF for the freeze requirement before reprioritize the new requirement since it has higher priority than it due to the dependency factor.

Fig. 4. demonstrates the prioritizer and scheduler that carries out the previous steps for reprioritizing the requirements based on the resources available and schedules them to the available team.

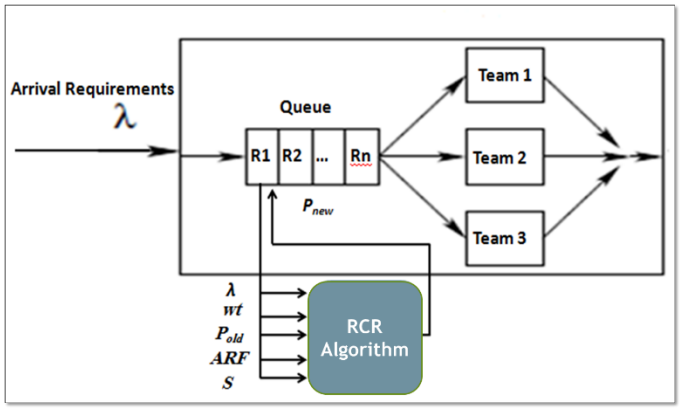

Fig. 4. RCP Algorithm.

IV. Case Study

To demonstrate our approach in a practical way, a sample software project for a library management system is considered, with three increments and twenty requirements, R = {R1,…, R20}. The first increment contains 9 requirements with the following dependencies:

InternalDependency = (R1), (R2, R1), (R3, R1), (R4, R1)

,(R5,R2,R3),(R6,R3),(R7,R3,R4),(R8,R5,R6),(R9)},the tuple (R3, R1) means that R1 depends on R2.

The second increment contains six requirements with the following dependencies:

InternalDependency={(R10),(R11,R10),(R12,R10),(R13,R11, R12),(R14,R11), (R15, R12)}.

The third increment contains five requirements with the following dependencies:

InternalDependency={(R16),(R17,R16),(R18),(R19,R17),(R2 0,R17)}.

Table I represents the traceability list for the twenty requirements. This list is converted into a directed acyclic graph which represents the requirements and interdependencies between them; Table II describes the number of InDegree and OutDegree edges for each vertex in the graph, Table III is the findings of the proposed algorithm which represents all the requirements and its associated priorities, notice that the requirement R1 has the highest priority with 2.0. While the requirement R19 has the lowest priority with 0.1, the scale is based on the total number of requirements in the software. Table IV showed the reprioritization process based on the resources constraints. Owing to the unavailability of the tools needed to develop these requirements, those requirements and their dependent requirements are frozen.

TABLE I. Traceability List

Increments	Requirement	Depends-On
Increment1 (9 Req)	R1	-
	R2	R1
	R3	R1
	R4	R1
	R5	R2,R3
	R6	R3
	R7	R3,R4
	R8	R5,R6
	R9	-
Increment2 (6 Req)	R10	-
	R11	R10
	R12	R10
	R13	R11,R12
	R14	R11
	R15	R12
Increment3 (5 Req)	R16	-
	R17	R16
	R18	-
	R19	R17
	R20	R17

TABLE II. InDegree and OutDegree for Requirements Independency Graph

Increment	Requirement	In Degree	Out Degree
Increment1 (9 Req)	R1	0	3
	R2	1	1
	R3	1	3
	R4	1	1
	R5	2	1
	R6	1	1
	R7	1	0
	R8	2	0
	R9	0	0
Increment2 (6 Req)	R10	0	2
	R11	1	2
	R12	1	2
	R13	2	0
	R14	1	0
	R15	1	0
Increment3 (5 Req)	R16	0	1
	R17	1	2
	R18	0	0
	R19	1	0
	R20	1	0

TABLE III. REQUIREMENT PRIORITIZATION

Requirement	Dependency Volume	Dependency Scope	Dependency Intensity	Priority
R1	3	1	1	2.0
R2	1	0	0	1.6
R3	3	1	1	1.8
R4	1	1	1	1.7
R5	1	1	1	1.5
R6	1	1	0	1.4
R7	0	0	1	1.3
R8	0	0	1	1.2
R9	0	0	1	1.9
R10	2	1	1	1.1
R11	2	1	1	1.0
R12	2	1	0	0.9
R13	0	0	1	0.7
R14	0	1	1	0.8
R15	0	0	1	0.6
R16	1	1	1	0.5
R17	2	1	1	0.3
R18	0	1	1	0.4
R19	0	1	0	0.1
R20	0	1	1	0.2

TABLE IV. REQUIREMENT REPRIORITIZATION

Requirement	Arrival Time λ	Waiting Time *wt*	*ARF*	P_{old}	P_{new}
R1	1	0	1	2.0	2.0
R9	2	0	1	1.6	1.9
R3	3	0	1	1.8	1.8
R4	4	3	0	1.7	1.4
R2	5	0	1	1.5	1.7
R5	6	0	1	1.4	1.6
R6	7	1	1	1.3	1.5
R7	8	1	0	1.2	1.2
R8	9	0	1	1.9	1.3
R10	10	0	1	1.1	1.1
R11	11	0	1	1.0	1.0
R12	12	1	0	0.9	0.8
R14	13	0	1	0.7	0.9
R13	14	0	0	0.8	0.7
R15	15	0	0	0.6	0.6
R16	16	0	1	0.5	0.5
R18	17	3	0	0.3	0.1
R17	18	0	1	0.4	0.4
R20	19	0	1	0.1	0.3
R19	20	0	1	0.2	0.2

V. CONCLUSION

In incremental software model small releases of the software are implemented in a sequence fashion instead of delivering the whole system after a long time of development. Therefore, this model can influence the prioritization of requirements in efficient manner so that the most important requirements can be implemented in the first releases of the system. A model were proposed to achieve requirement prioritization and reprioritization based on requirement interdependencies which represented as a hybrid approach of tractability list and directed acyclic graph, and on the resources constraints of the requirements. The proposed algorithms were introduced, analyzed and implemented using a case study. PDG and RCR algorithms require time complexity of O (|V| + |E|) which is a linear running time compared to the quadratic time complexity provided by the available algorithms that handle requirement prioritization. Also, the proposed approach has the ability to reprioritize the requirement based on the resources availability. Future work may add an improvement to the proposed algorithms or may combine them with other priority algorithms, in order to provide a hybrid solution that enhances the overall process.

REFERENCES

[1] K. Pohl, K.: Process-centered Requirements Engineering. Wiley, New York (1996).

[2] P. Carlshamre, K. Sandahl, M. Lindvall, B. Regnell and J.N och Dag, " An industrial survey of requirements interdependencies in software product release planning" . In Proceedings Fifth IEEE International Symposium on Requirements Engineering ,pp. 84-91, 2001.

[3] S. Soomro , A. Hafeez , A. Shaikh , SH. Musavi, "Ontology based requirement interdependency representation and visualization", InInternational Multi Topic Conference, pp. 259-270, Springer, 2014

[4] I. Sommerville, Software Engineering, Ninth edition, Pearson, 2011.

[5] vestola, " A comparison of nine basic techniques for requirements prioritization". Helsinki University of Technology, 2010.

[6] L. Karlsson, H. Martin, and R. Björn. "Evaluating the practical use of different measurement scales in requirements prioritisation." In Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering, pp. 326-335, 2006.

[7] M. Pergher and B. Rossi. "Requirements prioritization in software engineering: a systematic mapping study". In Empirical Requirements Engineering (EmpiRE), 2013 IEEE Third International Workshop on , pp. 40-44, 2013.

[8] A. Asghar, A. Tabassum, Sh. Bhatti and A. Shah. "The impact of analytical assessment of requirements prioritization models: an empirical study." International Journal of Advanced Computer Science and Applications (IJACSA) , vol. 2, pp. 303-313, 2017.

[9] V. Ahl, V. "An experimental comparison of five prioritization methods - investigating ease of use, accuracy and scalability". Master's thesis, Blekinge Institute of Technology, Ronneby, Sweden, 2005.

[10] M. Yaseen, A. Mustapha, N. Ibrahim ,"Prioritization of Software Functional Requirements: Spanning Tree based Approach", International Journal of Advanced Computer Science and Applications (IJACSA) vol, 10, pp.489-497, 2019.

[11] JA. Khan, IU. Rehman , YH .Khan, IJ .Khan, S. Rashid, "Comparison of Requirement Prioritization Techniques to Find Best Prioritization Technique". International Journal of Modern Education & Computer Science. Vol. 11, pp. 53-59, 2015.

[12] C. Duan, P. Laurent, J. Cleland-Huang, and C. Kwiatkowski. "Towards automated requirements prioritization and triage" . Requirements Engineering, vol. 2, pp. 73-89, 2009.

[13] F. Moisiadis, "The fundamentals of prioritising requirements", InProceedings of the systems engineering, test and evaluation conference (SETE'2002) ,2002,

[14] S. Hatton, "Early prioritisation of goals. In Advances in conceptual modeling – Foundations and applications", ER 2007 Workshops CMLSA, FP-UML, ONISW, QoIS, RIGiM, SeCoGIS, Auckland, New Zealand, (pp. 235-244), 2007.

[15] M. Khari, N. Kumar, "Comparison of six prioritization techniques for software requirements", Journal of Global Research in Computer Science. vol. 4, pp. 38-43. 2013.

[16] M. Yousuf, MU. Bokhari, M. Zeyauddin," An analysis of software requirements prioritization techniques: A detailed survey", In2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), pp. 3966-3970, 2016.

[17] M. Yaseen, N. Ibrahim , A. Mustapha," Requirements Prioritization and using Iteration Model for Successful Implementation of Requirements", International Journal of Advanced Computer Science and Applications (IJACSA) vol, 10, pp.121-127, 2019.

[18] Z. Racheva, and M. Daneva, "Reprioritizing the Requirements in Agile Software Development: Towards a Conceptual Model from Clients' Perspective". In SEKE ,pp. 73-80, 2009.

[19] I. Bassey,"Towards Release Planning Generic Model: Market-driven software development perspective, IJERT, vol. 2 , 2013.

[20] L. Qi, "value based dependency aware inspection and test prioritization", PHD Dissertation , University of Southern California ,2012.

[21] TH. Cormen, C. Leiserson, E. Rivest, R. L., and C. Stein," Introduction to algorithms". MIT press, 2009.

[22] N. Saher, F. Baharom, and R. Romli, "Guideline for the Selection of Requirement Prioritization Techniques in Agile Software Development: An Empirical Research", International Journal of Recent Technology and Engineering (IJRTE), vol, 8, pp.3381-3388, 2020.

[23] N. Borhan, H. Zulzalil, S. Hassan, N. Mohd Ali, "Requirements Prioritization Techniques Focusing on Agile Software Development: A Systematic Literature Review", INTERNATIONAL Journal of Scientific and Technology Research, vol. 8, pp. 2118- 2125, 2019.