Importance of Requirements Prioritization in Parallel Developing Software Projects

Muhammad Yaseen [1], Aida Mustapha [1], Atta Ur Rahman [2], Sadiq Khan [3], Wajid Kamal [4]

[1] Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia

[2] Department of Computer Science COMSATS University Islamabad, Islamabad

[3] Department of Computer Science, International Islamic University Islamabad

[4] Department of Computer Science and Software Technology, Abasyn University Peshawar

yaseen_cse11@yahoo.com

Abstract— Prioritizing software requirements is important and most difficult task during requirement management phase of requirement engineering especially when requirements are large in size. Large size requirements like Enterprise Resource Planning (ERP) system are difficult to manage and implemented by single developer. Requirements can be distributed in parallel team members. Requirements are dependent on each other, so for implementation of particular requirements, the pre-requisite requirements should be implemented first. Some requirements are needed for one team member, some for many and some for none. Giving importance and priority to some requirements over the others is necessary so that requirements can be available on time to developers. In this research work, requirements are prioritized on the basis of their implementation and importance with graph based approach. From directed graph, one can easily find set of all requirements which need particular requirement. Through experiment conducted on requirements of ODOO ERP, requirements are prioritized and time estimation is reduced. Time reduction and better management of requirements will cause successful delivery of software projects.

Keywords- Functional Requirements, Requirements Prioritization, Directed Acyclic Graph.

I. Introduction

Requirement Engineering is the systematic and discipline way of collecting user requirements for software system [1][2][3]. Requirements Prioritization (RP) is giving priority or ordering to requirements and is important activity during efficient requirements management [4][5][6]. There are five types of requirements i.e. Business Requirements (BRs) deals with benefits of implementing requirements [7][8][9], Process Requirements (PR) deals with time and cost issues of requirements [10][8], User Requirements (URs) are those requirements which comes from user [11][12], Functional Requirements (FRs) are those requirements which are necessary for software and which the system must do [13] [14] and Non Functional Requirements (NFRs) like usability, security, performance etc. [15][16]. Techniques like 'Cost value ranking', 'Attribute goal oriented', 'Value oriented' are suggested for prioritizing BRs [17]. Some of the techniques like 'AHP', 'Binary tree', 'value based', 'genetic algorithm', '' are suitable for prioritizing URs and FRs [18] and techniques like 'QFD', 'Contextual preference based technique' are suggested for NFRs [19]. Although most of the techniques like AHP work well for small size requirements but fails on large size requirements. Techniques like 'Machine Learning Based' 'SNIPR', 'ANN based', are suitable for prioritizing large size URs but are not applied to prioritize FRs from developer's perspectives [20]. FRs prioritization bears more significance especially when parallel team members are going to implement the requirements. Giving importance and priority to some requirements over the others is necessary so that requirements can be available on time.

II. Background Study

The Analytical Hierarchy Process (AHP) is the most famous, most used and simplest technique for RP. AHP prioritization is performed pairwise by comparing each and every requirement against each other. For n requirements, then n (n-1)/2 comparisons will be needed. For example, if the number of requirements is ten then AHP will perform forty five times comparisons of the requirements. If the requirements increase in size, so does the processing time. If the requirements is a thousand, there will be 1000*(1000-1)/2 = 499,500 comparisons, which is both very time consuming and difficult to execute. Because the technique is time consuming, it is not scalable for big requirements due to the pairwise comparisons for every requirement [21]. Cost Value Approach is also a famous technique whereby this technique assigns cost and value to each requirement and prioritize accordingly. Sometimes requirements are valued but cost is high and sometimes cost is low but requirements are not so valued and vice versa. This approach prioritizes based on dependency of cost and value on each other [7]. In Numerical Assignment [22] technique we divide and place requirements in groups according to the instructions of stakeholders. All the requirements in a group have same priority. The main problem with numerical assignment is that there is not standard criteria of defining groups, therefore leads to implementation of all requirements. In this approach, the requirements are categorized in three groups, which are critical, standard and optional. Cumulative voting is also known as 100 dollar method of RP. In this technique, 100 dollars are given to stakeholders and they assign dollar value out of 100 to every requirement. Nonetheless, this technique is suitable for a small group of stakeholders. Project having many stakeholders will be difficult to manage and prioritize using this technique [10] [23]. In Goal-based Technique, the goals of the customers are considered. Some requirements are implemented in first version, while some are implemented in subsequent releases. The goals can be low cost, quality or any selected features [14]. In [20], the goal-based technique was used during the requirement elicitation process. The proposed elicitation algorithm consist of 3 steps; (1) collect requirements using goal-oriented approach from the customers, (2) calculate cost and effort for each requirement, and finally (3) perform pairwise comparison based on AHP and prioritized accordingly [17]. Genetic algorithms (GA) are used as a technique for reducing the number of comparisons during RP. Using this technique, the knowledge is extracted from user and the requirements are prioritized accordingly [24].

III. GRAPH BASED APPRAOCH

3.1. Directed Acyclic Graph

This paper proposes a directed acyclic graph (DAG) approach for inter-relating and prioritization of FRs. Graph based approach is used in other study to inter-related FRs [25]. The inputs are the FRs collected from any sources using appropriate elicitation technique and must be specified in the form of Software Requirement Specification (SRS). In this research, FRs are represented as alphabets R1, R2. Rn enclosed in circles as nodes. Figure 1 shows DAG inter-relating FRs. In Figure 1, R1 is required for R2 and R3 while R3 is required for R4.

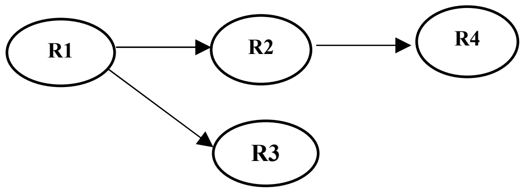

Figure 1: Representing FRs with DAG

The purpose is to apply efficient algorithms of graph that are helpful to give enough information's about requirements. Two types of information's are about requirements are required. First is the chain priority or implementation priority of requirement i.e. which requirement is necessary for the completion and implementation of requirement and second is set of all requirements are needed for which a particular requirement is necessary. E.g. A is required for B and B is required for C and D. Then A is required for B, C, and D. The following algorithm are used in our study:

3.2. Spanning Trees from Directed Acyclic Graph

Depth first search (DFS) is a method use to explore a graph using stack as the data structure [26]. It begins from the root of the graph, explore its first child, explore the child of next vertex until reach to the goal vertex or reach to final vertex having no further child. At that point, back following is utilized to give back the last vertex which is not yet totally explored. Modifying the post-visit and pre-visit, DFS is used to solve many important problems and it takes O (|V|+|E|) steps [12].

DFS algorithm traverses a graph in a depth ward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration [27][28][29][30].

This algorithm works according to the following rules.

Rule 1 − Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack.

Rule 2 − If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.)

Rule 3 − Repeat Rule 1 and Rule 2 until the stack is empty.

Spanning trees can be found in linear time by simply performing breadth-first search or depth-first search. These graph search algorithms are only dependent on the number of vertices in the graph, so they are quite fast [31][32]. Figure 2 shows different FRs inter-related with DAG.

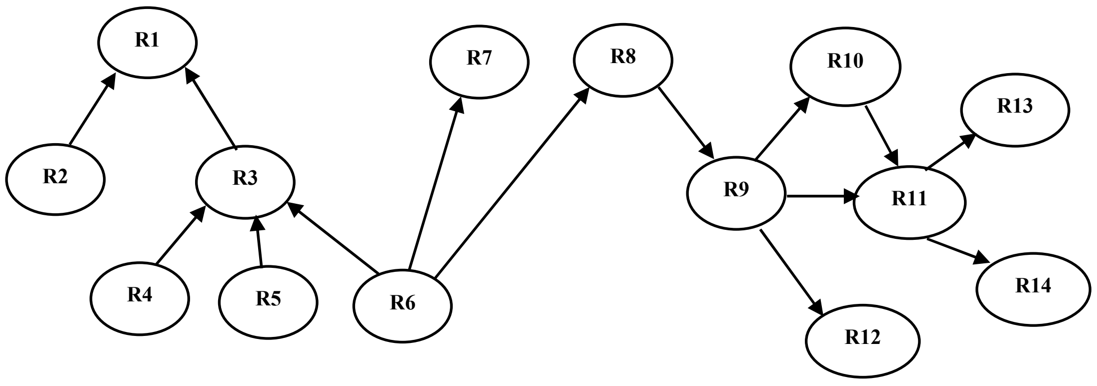

Figure 2: DAG used for making spanning tree

The resulted spanning trees of graph of Figure 2 are shown in Figure 3 respectively. We can see that four spanning trees are resulted. Height of tree shows the level of requirement in tree which can benefit in prioritization. Through DFS, one can easily calculate the height of tree. The following steps are included to calculate the height of spanning tree.

1. Start from root node, through DFS, move to the leave nodes and count the values of requirements in depth until the last node is reached. The value of count will determine the depth of particular chain.
2. Similarly repeat the same process for other chains through DFS.
3. The maximum count of any chain shows the depth or height of a tree.
4. If requirement is common in more than one chain, then maximum count will determine height of that requirement.

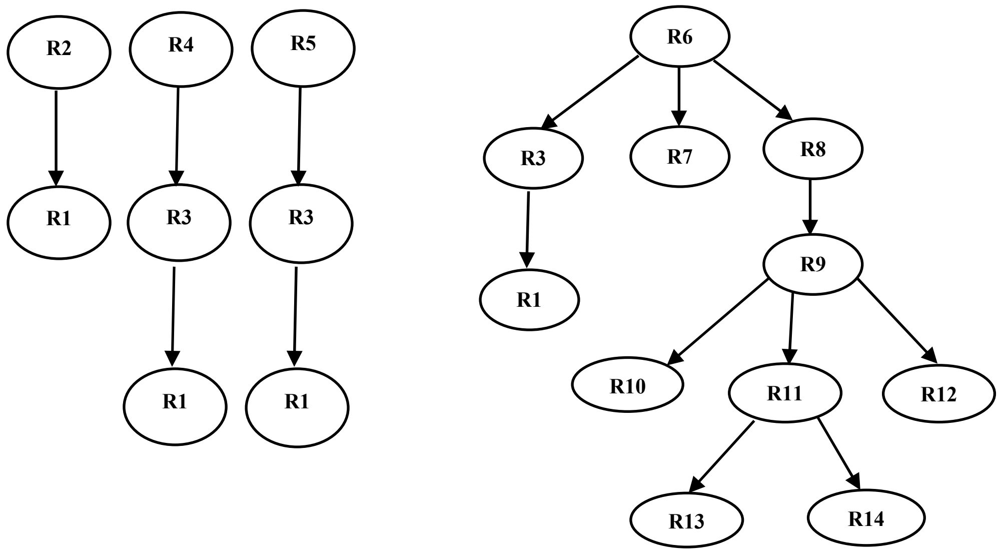

Figure 3: Resulted spanning trees from DAG of Figure 2

IV. EXPERIMENT

Requirements priority is based on how much they are needed or dependent for other requirements so that timely delivery of project can be assured. In most of the studies, delay in projects and cost issue are highlighted as reason of failure of projects. There are two kinds of delay that can occur in projects. Critical delay and non-critical delay. Critical delay is one which can delay the whole project. Critical path is that in which all requirements are tightly connected without any slack and delaying any requirement can delay a whole project. Another kind of delay is non-critical delay which doesn’t affect the overall delay of project. In such case, requirements are not on the critical path. The purpose of prioritization is not only to reduce critical delay but reducing non-critical delay is also necessary so that timely delivery of requirements can also be assured and this will not let developers to wait too much for requirements because requirements for implementation will be available in time.

4.1. Hypothesis:

In order to prioritize important requirements, the following hypothesis are finalized.

Hypothesis: H1 (Null): Requirements priority depends on how much they are needed for other requirements. Requirements priority increases when its need for other requirements increase.

Hypothesis: H1 (Alternative): Requirements priority not always depends on how much they are needed for other requirements. Requirements priority not always increases when its need for other requirements increases.

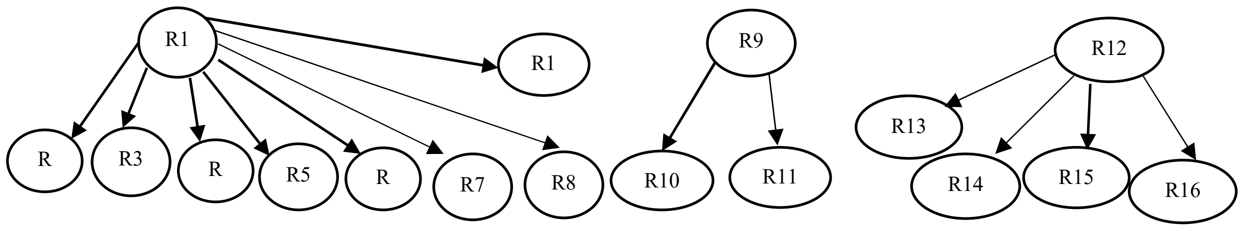

Figure 4: FRs represented with different spanning trees

We have to compare the priority of R1, R9 and R12 of Figure 4. Chain priority of all requirements are same i.e. 2 but all these are needed for different number of requirements. If single person is going to implement all these requirements than either a developer implement R1, R9 or R12, the overall efforts or total time estimation will remain same because single person will develop all the requirements. If requirements are distributed between two developers as shown in Figure 5 e.g. one member implements R1, R9 and R12 and may implement other requirements and some requirements are given to team member B. We make different scenarios for distributing these requirements.

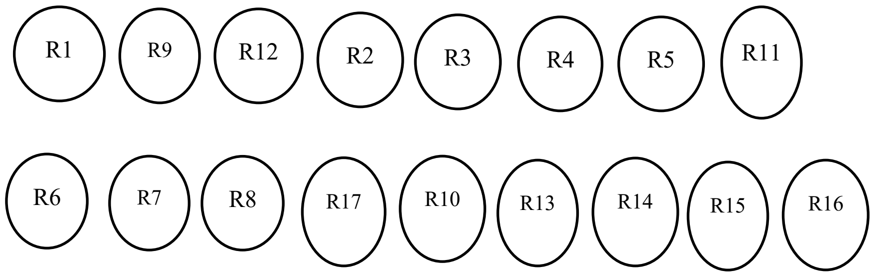

Figure 5: Distribution of requirements in two teams

Requirement R1 is required for R6, R7, R8 and R17. When R1 is implanted and R6 starts implementing then during implementation of R6, R9 will be implemented but R7, R8 and R17 will still wait for R6 completion because developer can implement only one requirement at time. After completion of R6 and R9, R7, R8, R17 and R10 are available for implementation. So we can say that either R6, R7, R8 or R17 wait for completion of R10 or either R10, R7, R8, R17 wait for the completion of R6, the delay is same in both cases. On the other hand if developer gives more priority to R9 instead of R1, then after completion of R9, both R10 and R1 can be implemented on same time, then R6, R7, R8 and R17 will wait for R10. Either R9 implement first or r1 implement first, waiting time or overall delay will be same.

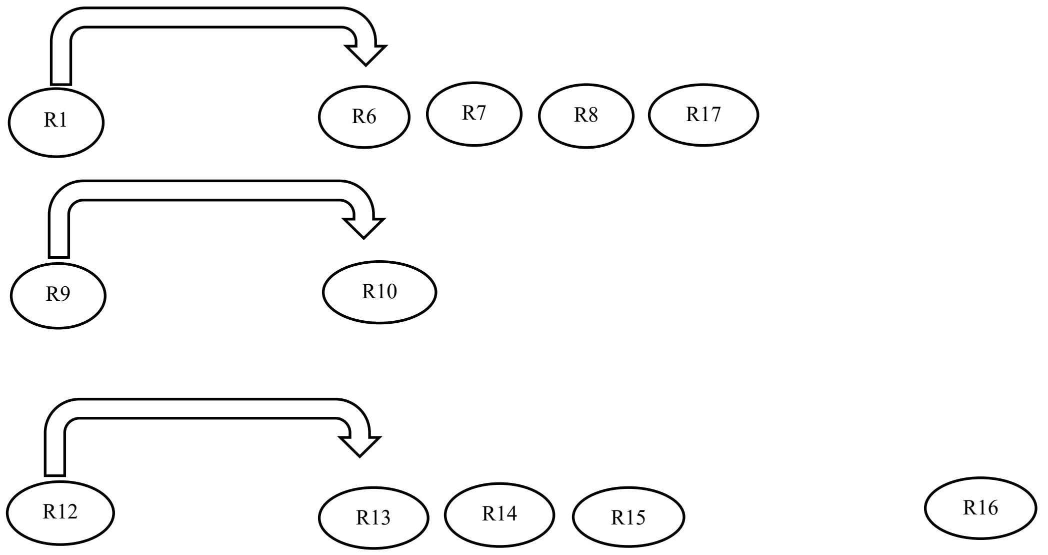

Figure 6: Relationship of requirements distribution and priority

Up to now we have found that the priority of R1 and R9 are same. Now compare priority of R1 with R12 as shown in Figure 6. As both are required for four other requirements but the difference is that one of the requirement R16 that needs R12 is to be implemented by team R3. Now two requirements i.e. R16 and R13 will be implemented in parallel. Requirements that need team member "A" requirements are implementing by one team member while requirements of R12 by two team members so delaying R12 can delay two teams' requirements.

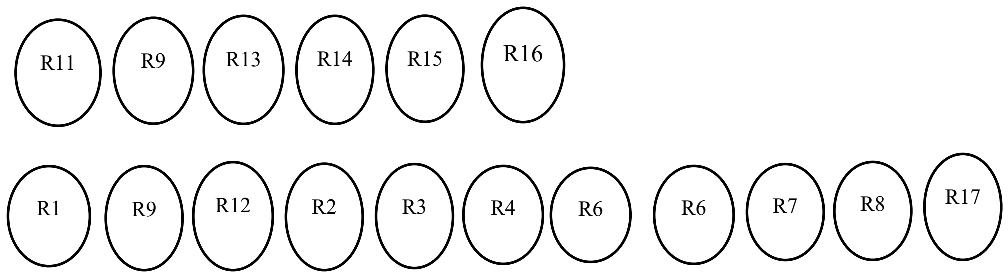

Figure 7: Distribution of requirements, with second team containing only two modules requirements

In case 2 as shown in Figure 7 , if we give more priority to requirement R1 than R11 will wait for R9 and this will delay the project because there is no requirement with team B that need R1 that can be implemented first to avoid the delay. Although R1 is required for eight other requirements but all the requirements are to be implemented by same team member A. we implement either R1 or R12 first, it will not affect the overall estimation time of A but it will delay team member B requirements because the depended requirements of R9 or R12 have to wait. The overall estimation time can also be affected if the total estimation time of B requirements is greater than total estimation time of A, then by giving priority to R1 will delay the whole project. As A has no depended requirements in team B, so it will be better to implement it after R12 or R9. If one of the requirement in R11 and R10 is given to team 'A' then still the priority of R9 will remain the same because team can work on one requirement at the one time. From this we can conclude if requirement is required for other requirements and that are to be implemented by same team member than this increase in importance of that requirement doesn't contribute to its increasing priority. Priority can be assigned on the basis of how much team members need a particular requirement.

From above hypothesis we can conclude that priority of requirement not depend on how much they are needed in number but how much they are needed by different teams. So null hypothesis is rejected and alternative hypothesis is supported.

Hypothesis: H2 (Null hypothesis): "If two requirements that are required for same number of other requirements but with different chain priorities or height in spanning tree have always same importance"

H2 (Alternative hypothesis): "If two requirements that are required for same number of requirements but with different chain priorities have unlike importance"

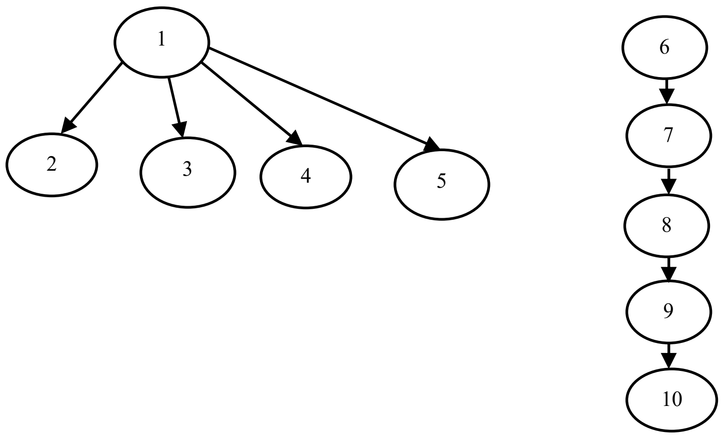

Figure 8: Same number of requirements but with different levels

We are comparing R1 and R6 of Figure 8, both requirements are required for four other requirements but depended requirements of R1 are not dependent on each other and all have same priority and height in tree while requirements of R6 are all dependent on each other with different priorities with tree height of 4.

If single team member implements all the requirements then there is no benefit of prioritizing requirements on the basis of how much important they are. They all will be consider equally important as concluded from the above hypothesis. Now increase the number of team members e.g. we take three team members A, B and C as shown in Figure 9 and Figure 10 respectively.

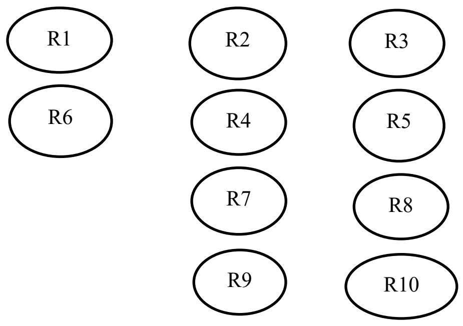

Figure 9: Equal distribution of two module requirements on teams

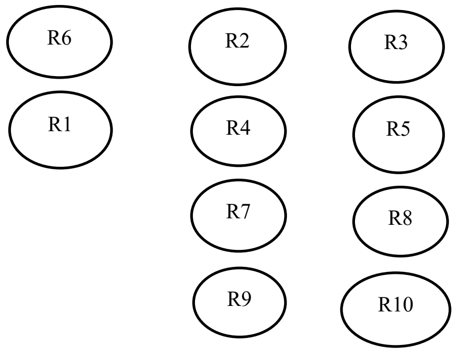

Figure 10: Equal distribution of two module requirements on teams with only swapping of R1 and R6

In such case where depended requirements of both R1 and R6 are equally distributed in same ratio on team member B and C and have same impact on overall delay. Because when we first implement R6, then R7 will be implemented and R8 in parallel shall wait for R7 and during this during R1 can start implementation. After completion of R8, R9 will start implementation but R10 will wait. After completion of R10, rest of the requirements that needed R1 can be implemented in parallel. If we implement requirements of R1 before the requirements that need R6, waiting time and total estimation time will remain same. Suppose all requirements take same hours' time of 10 hours to implement than the total estimation time of both cases will be same i.e. 70 hours. Now suppose all requirements of R6 are assigned to team B and requirements of R1 to team C as shown in Figure 11 and Figure 12 respectively.

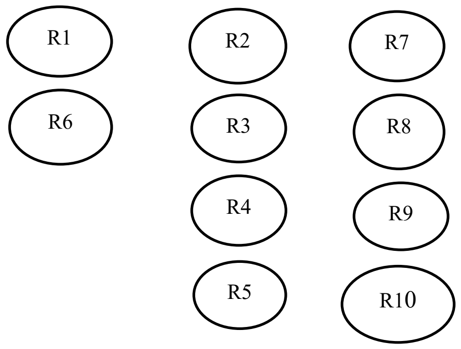

Figure 11: Second team get first module requirements and third team gets third get second team requirements

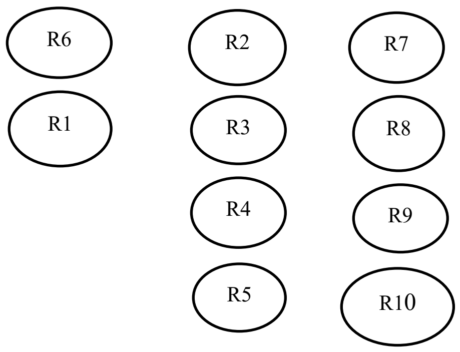

Figure 12: Modification in figure11 by swapping R1 and R6.

In such case still the total estimation time will be not effected. The reason is that one team member can implement only one requirement, in this case, B gets all requirements that need requirement R1 and C gets all requirements that need R6, but these team members can only implement single requirement at one time as stated before, so the effect reduces to single requirement at time only. If team A implement some of requirements of R6 e.g. R7 and R8 as shown in Figure 13 and Figure 14 then in this case the ratio of distribution is not same but both teams are implementing requirements of both R1 and R6. From this we can conclude that equal number of requirements are not necessary for teams, but equal distribution is necessary. If teams B and C are implementing requirements that need R1 and R6, priority will remain same although number of requirements are not same.

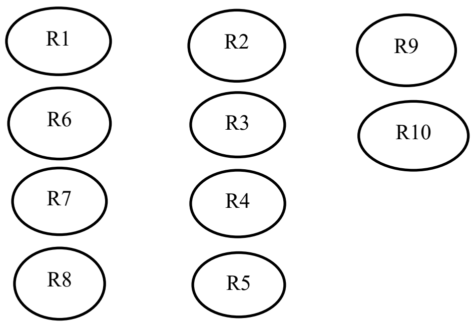

Figure 13: Two requirements of third team of figure 12 are to be developed by first team

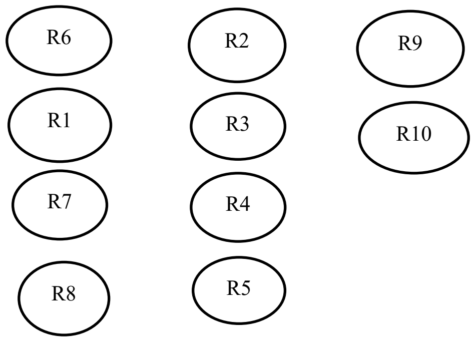

Figure 14: Modifications in figure13 by swapping R1 and R6

Suppose all requirements take same time of '10 hours' to complete their implementation. Total estimation time will be 60 hours, either we give priority to R1 or R6. Now after bringing changes in above cases, we have increased the team member's size as shown in Figure 15 and Figure 16.

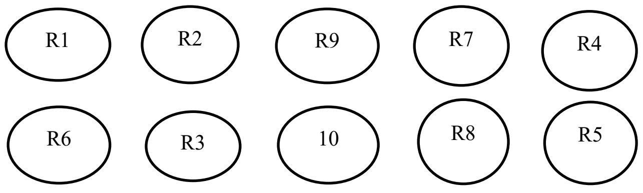

Figure 15: Distribution of requirements in five teams

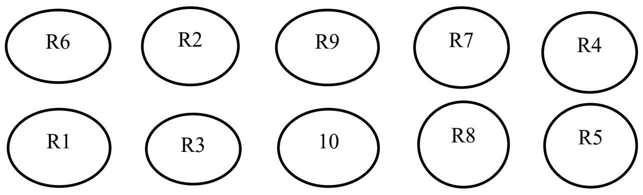

Figure 16: Modifications in figure 15 by swapping R1 and R6

Team B and C get requirements of R1 while D and E get requirements of R1. In this case though requirements are equally distributed but no team member implement requirements that need both R1 and R6. In first case, after implementing R1, it is available for R2 and R3 and both of them can be implemented in parallel. During this implementation R6 will start its implementation. After R6 completion, team member D can start implementing R7 but R8 cannot be implemented in parallel, team member E will wait for R7 and after completion of R7 then R8 can be implemented because R8 needs R7. Similarly after R8, R9 and then R10 will be implemented. Implementation of R7, R8, R9 and R10 takes a lot of time as they are implemented one after another. Suppose each requirement of Figure 15 and Figure 16 take 10 hours, then these four requirements will take 40 hours. If we give low priority to R6 which is needed for these four requirements, then total time will equal total time for these four requirements and R1 with R6. If each requirement takes 10 hours than total time will become 60 hours. But if we give more priority to R6 then time will reduce to 50. Delay of R2, R3 and R4, R5 will not delay the whole project as these requirements can be implemented in parallel, so if R1 is delayed than parallel development of these requirements will not delay whole project. So we can conclude that if single team is implementing requirements or multiple two team members than prioritization has no effect on total estimation time. In neither case we can say that R1 get more priority than R6, so it's better to implement R6 always before R1. Thus Null hypothesis is rejected and proved that requirements that are required for same number of other requirements but with different chain priorities have not always alike in importance. Let's suppose along with R1 and R6, team member A in above case is implementing R9 and 10, which means member D and E are only implementing R7 and R8. In this case the priority of R6 become decreased. Now at this level we will check the total estimation time of all teams. In this case total estimation is same either we implement R1 or R6 first. The team which takes more time to implement its requirement will be the total estimation time of project. In above case team member B and C takes maximum time where R9 and R10 are to be implemented by team member A as compare to D and E, so requirements of D and E will not delay the project by giving low priority to R6 as compare to R1. So besides from height of requirement or chain priority of requirement, sum of estimation times of individual requirements of particular team member is also necessary. While comparing two requirements, one with maximum height and other with maximum requirements, we will check estimation time of all other requirements which need these requirements and if estimation time is found same or maximum of requirements of maximum height than priority should be given to requirement with maximum height else priority will be consider same. Requirement that is needed for maximum requirements but possess low chain priority can have high priority if it is needed for maximum team members as compare to that requirement which high chain priority but needed for lesser team members because delaying this requirement will delay maximum team members requirements. For example in case shown below, R1 Priority will be high because it is needed for two members requirements while priority of R6 will be low. We can see the effect as delaying R1 can delay R2 and R3 of team member B but delaying any requirement R1 or R6 have no impact on delay of team member C requirements because team member C have available requirements that need both R1 and R6.

V. PRIORITZATION RULES

Based on the hypothesis presented in above section, we can formulate certain rules while prioritizing requirements. Summary of rules in the light of above hypothesis are given below:

1. If only single team member is going to implement all the requirements, then all requirements are consider to be of same importance. Priority will be given to requirements on the basis of chain priority in this case.
2. If multiple team members are going to implement the requirements, then priority will be given to those requirements that are needed for requirements of other team members. Importance of requirement will not increase with number of its need for other requirements, it will be decided on how much maximum teams require this particular requirement.
3. If the importance of two requirements found to be same in terms of its need for other teams then priority should be given on the basis of its height in tree or chain priority. Maximum height means more priority.

VI. CONCLUSION

In this research work, a prioritization rules are presented that can prioritize FRs from developer's perspective. Prioritizing FRs will not help only in easy management of requirements i.e. which requirements are needed for other requirements but will help in prioritizing important requirements that are required for requirements of other team members of parallel development projects. From experiments, we have concluded that importance of requirement not only increases with how much they are required for other requirements but how much it is required for different team members. Different team members can wait for particular requirement and this waiting time can delay the overall project so by prioritizing important requirements on the basis of how much they are required is essential. Through this framework, developers can easily implement prioritize requirements that can reduce total estimation time of the project.

Reference

[1] Z. Ali and M. Yaseen, 'Critical Challenges for Requirement Implementation in Global Software Development : A Systematic Literature Review Protocol with Preliminary Results', vol. 182, no. 48, pp. 17–23, 2019.

[2] M. Yaseen, S. Baseer, and S. Sherin, 'Critical Challenges for Requirement Implementation in Context of Global Software Development : A Systematic Literature Review', pp. 120–125, 2015.

[3] M. Yaseen and M. A. Awan, 'Practices for Effective Software Project Management in Global Software Development : A Systematic Literature Review', vol. 177, no. 36, pp. 1–6, 2020.

[4] M. Yaseen, A. Mustapha, and N. Ibrahim, 'MINIMIZING INTER-DEPENDENCY ISSUES OF REQUIREMENTS IN PARALLEL DEVELOPING SOFTWARE PROJECTS WITH AHP', vol. 8, no. Viii, 2019.

[5] M. Yaseen, A. Mustapha, and N. Ibrahim, 'An Approach for Managing Large-Sized Software Requirements During Prioritization', 2018 IEEE Conf. Open Syst., pp. 98–103, 2019.

[6] M. Yaseen, N. Ibrahim, and A. Mustapha, 'Requirements Prioritization and using Iteration Model for Successful Implementation of Requirements', Int. J. Adv. Comput. Sci. Appl., vol. 10, no. 1, pp. 121–127, 2019.

[7] Z. Ali, M. Yaseen, and S. Ahmed, 'Effective communication as critical success factor during requirement elicitation in global software development', vol. 8, no. 03, pp. 108–115, 2019.

[8] M. Yaseen and Z. Ali, 'Success Factors during Requirements Implementation in Global Software Development : A Systematic Literature Review', vol. 8, no. 3, pp. 56–68, 2019.

[9] M. Yaseen and Z. Ali, 'Practices for Effective Communication during Requirements Elicitation in Global Software Development', vol. 8, no. 06, pp. 240–245, 2019.

[10] M. Yaseen, 'Effective Negotiations Practices in Global Software Development : A Systematic Literature Review', vol. 9, no. 1, pp. 87–91, 2020.

[11] M. Yaseen, S. Ali, . A., and N. Ullah, 'An Improved Framework for Requirement Implementation in the context of Global Software Development: A Systematic Literature Review Protocol', Int. J. Database Theory Appl., vol. 9, no. 6, pp. 161–170, 2016.

[12] M. Yaseen, R. Naseem, Z. Ali, and G. Ullah, 'IDENTIFICATION OF CHALLENGES DURING REQUIREMENTS IMPLEMENTATION IN GLOBAL SOFTWARE DEVELOPMENT : A SYSTEMATIC', vol. 4, no. 1, pp. 23–40, 2019.

[13] M. Yaseen and U. Farooq, 'Requirement Elicitation Model (REM) in the Context of Global Software Development', Glob. J. Comput. Sci. Technol., vol. 1, no. 2, pp. 1–6, 2018.

[14] M. Yaseen, S. Baseer, S. Ali, S. U. Khan, and Abdullah, 'Requirement implementation model (RIM) in the context of global software development', 2015 Int. Conf. Inf. Commun. Technol. ICICT 2015, 2016.

[15] M. Yaseen, A. U. Rahman, I. U. Rahman, Z. Ullah, and M. Bacha, 'Inter-organizational Learning and Knowledge Sharing Management in Global Software Development', vol. 8, no. 1, pp. 52–57, 2020.

[16] M. Yaseen, N. Sarwar, M. Ali, and A. U. R. Rahman, 'Colloboration as Success Factor during Requirement Elicitation in Global Software Development', vol. 6, no. 3, pp. 39–46, 2020.

[17] N. Garg, M. Sadiq, and P. Agarwal, 'GOASREP : Goal Oriented Approach for Software Requirements Elicitation and Prioritization Using Analytic Hierarchy Process', pp. 281–287, 2017.

[18] M. Yaseen, A. Mustapha, N. Ibrahim, and U. Farooq, 'International Journal of Advanced Trends in Computer Science and Engineering Effective Requirement Elicitation Process using Developed Open Source Software Systems', vol. 9, no. 1, 2020.

[19] M. Yaseen, I. Journal, M. Yaseen, A. Mustapha, M. A. Salamat, and N. Ibrahim, 'International Journal of Advanced Trends in Computer Science and Engineering Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse09912020.pdf Prioritization of Software Functional Requirements : A Novel Approach using AHP and Spanning Tree', vol. 9, no. 1, 2020.

[20] N. Setiani and T. Dirgahayu, 'Clustering Technique for Information Requirement Prioritization in Specific CMSs', 2016.

[21] M. A. Iqbal, A. M. Zaidi, and S. Murtaza, 'A new requirement prioritization model for market driven products using analytical hierarchical process', DSDE 2010 - Int. Conf. Data Storage Data Eng., pp. 142–149, 2010.

[22] A. K. Massey, P. N. Otto, and A. I. Antón, 'Prioritizing Legal Requirements', vol. 1936, no. 111, 2010.

[23] P. Chatzipetrou, L. Angelis, P. Roveg??rd, and C. Wohlin, 'Prioritization of issues and requirements by cumulative voting: A compositional data analysis framework', Proc. - 36th EUROMICRO Conf. Softw. Eng. Adv. Appl. SEAA 2010, pp. 361–370, 2010.

[24] P. Tonella, A. Susi, and F. Palma, 'Interactive requirements prioritization using a genetic algorithm', Inf. Softw. Technol., vol. 55, no. 1, pp. 173–187, 2013.

[25] M. Yaseen, A. Mustapha, S. Qureshi, A. Khan, and A. U. Rahman, 'A Graph Based Approach to Prioritization of Software Functional Requirements', vol. 9, no. 3, pp. 64–73, 2020.

[26] L. G. Algorithms, 'Depth-first search and linear graph algorithms*', vol. 1, no. 2, pp. 146–160, 1972.

[27] M. M. Asadi, H. Mahboubi, J. Habibi, A. G. Aghdam, and S. Blouin, 'with Application to Underwater Sensor Networks', pp. 1–8, 2017.

[28] L. Arge and N. Zeh, 'I / O-Efficient Strong Connectivity and Depth-First Search for Directed Planar Graphs', 2003.

[29] K. Jun, 'Depth-First-Search based Region Merging for the Waterfall', pp. 540–545, 2015.

[30] M. Weigel, 'Connectivity algorithm with depth first search (DFS) on simple graphs Connectivity algorithm with depth first search (DFS) on simple graphs', pp. 4–8, 2018.

[31] M. Usman, D. Sakethi, R. Yuniarti, and A. Cucus, 'The Hybrid of Depth First Search Technique and Kruskal ' s Algorithm for Solving The Multiperiod Degree Constrained Minimum Spanning Tree', no. Icidm, pp. 0–3, 2015.

[32] S. Dhingra, 'Finding Strongly Connected Components in a Social Network Graph', vol. 136, no. 7, pp. 1–5, 2016.