*International Journal on Applications in Information and Communication Engineering*
*Volume 5 : Issue 1 : June 2019 , pp 61 –67 www.aetsjournal.com* ISSN (Online) : 2394 - 6237

# PRIORITIZATION TECHNIQUES FOR SOFTWARE REQUIREMENTS AND THEIR COMPARITIVE ANALYSIS

**Selvaraj K, Aarthi L, Karpagavalli P, Sangeetha M, Selva Mani R**
Department of Information Technology
Rathinam Technical Campus, Coimbatore, Tamilnadu, India

***Abstract—*** The technique of needs prioritisation that is most acceptable for each software system is contingent on the manner in which the product will be utilised as well as the degree to which it will be scalable. In this study, several different approaches to needs prioritisation are investigated in depth and compared using a set of standard metrics. These metrics include ease of use, total time required, accuracy, scalability, cost, and the total number of comparisons that are necessary to select the requirements for software development. The paper presents an approach to effectively prioritising requirements on the basis of the aforementioned criteria and illustrates how to do so.

***Index Terms***— Prioritization techniques for necessities, Prioritization techniques for necessities, and Comparison.

## I. INTRODUCTION

There are frequently more requirements raised than can be implemented at once. Prioritizing the needs will ensure that the most important ones are satisfied by the earliest product releases [1]. Software development decision-makers must make a variety of choices on the release plan throughout the course of a project. When preparing for upcoming releases, consideration must be given to factors including available resources, milestones, divergent stakeholder perspectives, potential market opportunity, risks, product strategies, and expenses. Unfortunately, there aren't many quick and efficient methods for prioritising requirements that could be applied to release planning [2]. This study compares six methods for ranking software needs in order of importance. The methods used are Value-Oriented Prioritization (VOP), Cumulative Voting (CV), Numerical Assignment Technique (NAT), Binary Search Tree (BST), and Planning Game (PG). We utilised each methodology in a methodical manner to rank a group of thirteen quality needs in order to study these techniques. Then, from the viewpoint of the user, we categorised the methods based on five factors: ease of use, total time necessary, scalability, accuracy, and total number of comparisons needed to reach a judgement.

2.Motivation

According to Lubars et alstudy .'s of the state of requirements engineering practise, many businesses think it's critical to prioritise requirements and make choices about them based on logical, quantitative evidence [3]. Still, it seemed that no organisation really understood how to set priorities or how to effectively convey these priorities to project participants. In the field of industrial software development, the idea that requirements vary in relevance is becoming more widely accepted. However, there hasn't been much advancement in the methods for software demand prioritisation, either theoretically or practically, up to this point. The method offered by the analytical hierarchy process, or AHP, offers a reliable foundation for ranking software needs [4]. In AHP, decision-makers analyse the requirements pair-wise to determine which is more crucial and to what degree. A major flaw in AHP prevents the industrial institutionalisation of the programme. Since all distinct pairings of requirements must be compared, a significant amount of work may be necessary. This growth rate may be acceptable for small-scale development initiatives, but it is likely to be too much work for large-scale projects. Five complimentary strategies to address AHP were discovered by Karlsson et al [5]. Since recent research suggests that making relative judgments likely to be faster and still produce more trustworthy findings than making absolute judgments, all of these strategies involve pair-wise comparisons [6]. These pair-wise comparisons take a lot of time and see rapid development as the number of requirements rises. Using weighted assessments of perceived value, relative penalty, predicted cost, and technical concerns, Wiegers advises a less exacting approach [7]. Wiegers' technique has a basic flaw in that the value assigned to a given need lacks the level of specificity required to assess whether or not it satisfies important company core values. A Value-Oriented Prioritization (VOP) process was created to get around these restrictions.

According to Vetschera and Wiegers' spreadsheet model, VOP is expressed as an additive weighting approach [7, 8]. According to Paetsch et al, agile software development has gained popularity in recent years, and one of the most well-liked approaches in this area is extreme programming, which uses a prioritising method called Planning Game (PG) [9]. Each technique is briefly described in the next section.

3. Prioritization Strategies

The prioritisation methods covered in this study are clarified in this section.

The number assignment technique (NAT) is founded on the idea that each demand is given a symbol that represents how important the requirement is thought to be. This method is frequently used in Quality Function Deployment (QFD), when it is necessary to prioritise candidate requirements [10]. There are numerous variations based on the numerical assignment method. Brackett [11] offers a simple method for using the concept, suggesting that criteria be categorised as essential, desirable, or inessential. A method that uses finer granularity is to give each criterion a number between 1 and 5, with the numbers indicating:

Obligatory (the customer cannot do without it).

Extremely crucial (the client doesn't want to be without it).

Very significant (the customer would appreciate it).

Not crucial (the customer would accept its absence).

Is irrelevant.

Analytical Hierarchical Process (AHP): Saaty [4] created and described the AHP for the first time in 1980. Although this is a promising technique, Regnell et al. [12] contend that it needs to be adjusted in some way because it is not designed to handle distributed prioritisation with numerous stakeholders. However, no studies explaining how such kind of alteration would work have yet been published. In AHP, the candidate requirements are assessed pair-by-pair to determine whether one need is more crucial than the other. According to Saaty [4], Table 1 should be used to determine the importance's level of importance.

Table 1. Basic scale according to Satty for pairwise comparison in AHP

| ***Sr.No.*** | ***How Important*** | ***Description*** |
|---|---|---|
| 1 | 1 | Equal Importance |
| 2 | 3 | Moderate difference in importance |
| 3 | 5 | Essential difference in importance |
| 4 | 7 | Major difference in importance |
| 5 | 9 | Extreme difference in importance |
| 6 | Reciprocals | If requirement *i* has one of the above numbers assigned to it when compared with requirement *j*, then j has the reciprocal value when compared with *i*. |

The required number of comparisons increases in a polynomial fashion due to the fact that this method mandates comparisons of all candidate requirements on a pair-wise basis. When designing a software system with n candidate requirements, it is necessary to perform n. (n - 1)/2 pair-wise comparisons.

Value-Oriented Prioritization (VOP): [13] VOP makes use of a framework that gives requirement engineers a basis for prioritising and selecting the needs of the organisation. By putting emphasis on the core values of the company, visibility is increased for all of the stakeholders involved in the decision-making process. This helps reduce the need for drawn-out debates and arguments regarding specific requirements. In order to construct a value-oriented prioritisation system, the first thing that needs to be done is to set up the framework for defining the core values of the company as well as the relationships that exist between those values. VOP conducts an analysis of customer requirements and establishes a hierarchy for meeting those requirements based on the connections between fundamental business principles. A technique for quantifying and ranking requirements for a software requirements specification, a prototype, or an application increment can be generated by the VOP framework. Executives of the company rank the most important business values using a straightforward ordinal scale, placing them in order of significance to the organisation.

Table 2. Value Oriented Prioritization matrix

| ***Requirements*** | ***Business Values ($V_1$…..$V_n$)*** | | | | | ***Score*** |
|---|---|---|---|---|---|---|
| | $V_1$=7 | $V_2$=6 | $V_i$=9 | VI+1=5 | $V_n$=8 | |
| $R_1$ | | | | | | |
| $R_2$ | | Wij | | | | |
| …. | | | | | | |
| | | | | | | |
| $R_N$ | | | | | | |

Voting cumulatively (CV)

Cumulative Voting, also known as the 100-Point Method or the Hundred-Dollar ($100) test, was developed by Leffingwell and Widrig. It is an easy-to-understand and aesthetically pleasing voting procedure in which each stakeholder is given a fixed number of imaginary units (for example, money) such as 100, 1000, or 10,000 to use as a voting token for the most important issues [14]. Other names for this voting procedure include the 100-Point Method and the Hundred The amount of money that is given to a certain issue therefore reflects the respondent's preference (and, consequently, their ranking) of that issue in comparison to the other issues. The shareholder is free to divide the points in any manner that they deem appropriate. Every stakeholder has the option of allocating the entire sum that has been allotted to them to a single issue that is of the utmost importance. Another option for a stakeholder is to distribute the funds equitably among a number of the issues, or even all of them.

CV is sometimes referred to as "proportional voting" because the number of units assigned to an issue reflects its relative priority in relation to the other issues. In this discussion, the word "proportional" can also refer to the fact that a proportion between zero and one is obtained by dividing the total number of units that are assigned to an issue by the total number of units that are made available to each stakeholder. This results in a proportion that is somewhere in the range of zero to one. Because each problem occupies a specific proportion (or percentage) of preference in the person's belief or judgement, the stakeholder ratings for a collection of issues can therefore be thought of as the "composition" or "mixture" of a person's view toward the concerns. This is true in the sense that each problem takes up a specific proportion (or percentage) of preference in the person's belief or judgement. It is possible for the process to generate problems with zero units, which indicates that the particular stakeholder views these problems as being of the utmost insignificance. When dealing with this kind of information, the presence of zeros is typically problematic because they render the concept of relative preference or priority completely meaningless and prevent ratio computation from taking place.

Naturally, it is possible to construct a questionnaire that does not allow zeros; however, the purpose of CV, in general, is to give stakeholders the opportunity to distribute their total amount without any additional restrictions.

The Binary Search Tree, abbreviated as BST: BST is a computer method that was developed to store information that can either be maintained or retrieved at a later time. In most cases, the BST T has one or two child nodes, but it can also be empty. When compared to the root node R, the child nodes to the right (r) are considered to have a greater value and importance, whereas the child nodes to the left (l) are considered to have a lesser value and importance. Every child node also acts as the root node for its own child node. When a node does not have any child nodes, we refer to it as a leaf node. Recursive searching within the BST is made possible as a result of this. Using BST has the advantage that when prioritising requirements, it only requires n log n [15] comparisons to insert all of the requirements in the correct order. This is the case even if there are n requirements to be prioritised. As a consequence of this, BST is a candidate that is quick, which may be advantageous in situations in which there are many requirements that need to be prioritised. In other words, the BST could be expanded to accommodate thousands of requirements while maintaining its status as a rapid candidate. The BST algorithm has one essential component that you should be aware of, and that is the requirement that a tree's balance be maintained in order to achieve the fastest possible insertion time. The BST is said to be in a state of balance when all of the leaves are located within a predetermined distance of the root. If the BST gets into an unbalanced state, the tree might need to be rebalanced after a node has been added or removed, but this will only be necessary if the state of the tree becomes unbalanced. Because of this, the insertion of a node ought to be as efficient as possible, or log n. On an ordinal scale, each prerequisite is separated from the others. This means that the only thing I will be able to determine is whether or not one requirement is more important than another, but not to what extent. Another shortcoming of BST is that it does not provide any kind of consistency ratio for us to use in determining whether or not the prioritisation we did was correct.

Extreme programming requires the customer to document their needs on a story card. In this version of Game of Planning (PG), the customer must record their needs. The requirements are then reorganised by the customer into three distinct piles. According to Beck [16], the piles ought to be labelled as follows: "those without which the system will not function," "those that are less essential but provide significant business value," and "those that would be nice to have." At the same time that the customer is sorting the story cards, the programmer will begin to sort the requirements into three distinct piles, or sort by risk. These piles will be referred to as "those that can be estimated precisely," "those that can be estimated reasonably well," and "those that cannot be estimated at all." This will occur simultaneously with the customer's sorting process. The programmer makes an estimate regarding the amount of time necessary to fulfil each requirement.

The customer or one or more of their representatives can decide on a fixed release date as well as the specifications that should be included in the upcoming release. These are both decisions that can be made by the customer. The final product of this sorting is an ordered list of requirements in the appropriate order. Because PG takes one requirement at a time and decides which pile the requirement belongs to without comparing it to any other requirement, the amount of time it takes to prioritise n requirements is equal to the number of comparisons. This demonstrates that PG is very flexible and able to handle a large number of requirements at the same time without requiring a significant amount of time to prioritise each of them.

4. Experimentation

The experiment's design and methodology are described in this section.

4.1 Initiation

In the experiment, the six different prioritisation methods are compared to one another in order to determine which one appears to be the most effective. Specifically, the goal is to determine which method is the quickest, most accurate, scalable when additional requirements are added, and requires the fewest comparisons. This is put to the test by asking the participants to describe how they feel and think each technique would be able to satisfy each of the criteria. This experiment was significantly impacted by the experimental design that is discussed in [17], which can be found here.

4.2 Form:

In order to gain a deeper comprehension of requirements prioritisation techniques, we carried out a single project study with the intention of characterising and assessing each of the six techniques for prioritisation from the point of view of the users [17]. Seven students who had already completed their graduate or postgraduate degrees took part in the experiment. They were tasked with ranking thirteen different quality requirements using the prioritisation strategies that were taken into consideration [18]. The participants, to the best of their knowledge, independently ranked the requirements in order of importance. During the process of determining the order of importance for the quality requirements, the cost of satisfying the requirements was not taken into account. In other words, the only factor that was taken into consideration was how important it was to the customers. In addition, it was believed that the requirements were orthogonal, which indicates that the significance of one requirement did not depend on the significance of another requirement.

We gave the examination in stages, separated by predetermined intervals, over the course of some time in order to lessen the likelihood that the participants would remember how they responded to the previous prioritisation. One specific strategy was broken down and discussed each day. Twenty minutes of time were allotted on a daily basis to facilitate the presentation of the method that was the focus of the experiment on that particular day. After receiving confirmation from each participant that they understood the procedure, sixty minutes were set aside in order to bring the experiment to a successful conclusion for that particular day. Every participant received the necessary papers, and the amount of time it took them to finish the experiment was meticulously recorded for each individual participant.

4.3 Dangers to the Validity When reviewing the findings of an experiment, one of the most essential questions to ask is, "How valid are the results?" Because of this, the question of the result's validity needs to be taken into consideration whenever an experiment is being designed. The purpose of the experiment was to assess and rank the significance of six different approaches to requirements prioritisation by making direct comparisons between them. We make no claim that the findings obtained from this experiment can be generalised and utilised by any user in any setting for any kind of application, as this is not our position. Instead, we focused on illustrative approaches to requirements prioritisation in an effort to acquire a deeper comprehension of these methods. It has been determined that the following dangers exist:

4.4 Too minimal requirements:

During the process of analysing the data, it became clear that there were insufficient requirements for the experiment. However, prior to the experiment, it was discussed whether it would be possible to take into consideration more than thirteen requirements. However, due to the fact that there was a time limit, i.e. how much time the participants could participate, the number of requirements needed to be limited. In order for this experiment to faithfully replicate the conditions of a real-world undertaking, the number of requirements ought to be in the neighbourhood of two hundred; however, it is unlikely that this will be possible within the constraints of the available time. Because of this, it was decided that there should only be thirteen requirements to fulfil.

4.5 The involvement of a small number of people in the experiment The significance of the findings is restricted due to the involvement of a small number of people (seven people) in the experiment. Because of this, the results were less conclusive than expected, and as a result, they can be considered to pose some difficulty for the evaluation. On the other hand, if invitations to participate in the experiment are sent out to a large number of people, there is a better chance that the potential for harm will be reduced.

4.6 Offline Evaluation: The evaluation was conducted separately from a genuine software development project, which could be interpreted as a potential obstacle for the purpose of this experiment. However, due to the fact that the primary purpose of this evaluation was to gain an understanding of and illustrate a number of possible methods for prioritising software requirements, it is not considered to be a major threat at this time.

4.7 Non-functional requirements are the only ones taken into consideration:

In this particular experiment, only non-functional requirements were taken into consideration. Despite this restriction, it is not anticipated that it will pose a significant challenge to the findings of the experiment.

4.8 Requirements are interdependent:

In actual practise, one must take into account the interdependence that exists between the requirements. This limitation of the experiment is not believed to have an effect on the actual evaluation of the various methods because none of the techniques for prioritising that are described in this paper offer any means for managing interdependence.

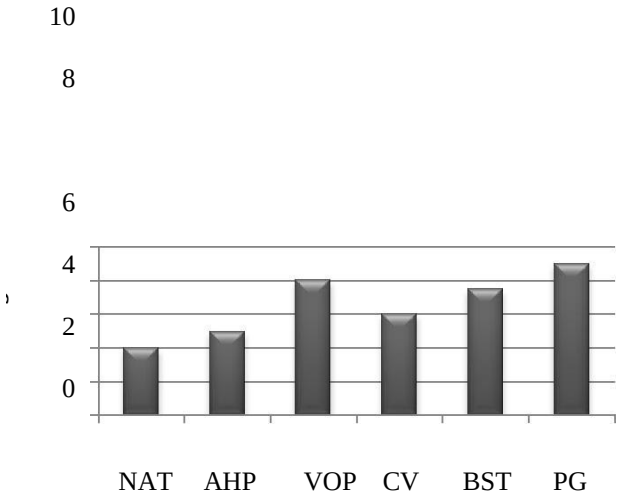
It is always important to identify potential risks associated with an experiment in order to allow for the determination of the internal as well as the external validity of the results that were obtained. Therefore, when analysing the results, the potential dangers that were discussed earlier should be kept in mind.

4.9 An analysis of the data that was collected:

The evaluation starts off with the initial inquiry into each method, which is then followed by the second, third, and so on questions. Participants ranked each approach to answering the question, and the final step was to determine the mean value. The following questions were posed to the participants after each technique was performed:

a. The first question that was asked of the participants was regarding the ease with which the prioritisation technique could be applied. The solution to the problem is presented in figure 1, as is appropriate.

According to what can be seen in Fig., the participants believed that the PlanningGame (PG) followed by VOP was the method that was the simplest to implement. The NAT, followed by the AHP, was the most challenging to manage. Both the CV and the BST scores were located in the middle of these two groups.**Ease of use**




**Prioritization Techniques**

b.The second question that was asked of the participants was how much time it took for the participants to perform the prioritisation with the techniques that were being considered. The answer to the question is presented in figure 2, as can be seen.

From the result in fig. 2, clearly NAT took the longest time to execute, followed by AHP. The fastest technique was VOP and PG. Between fastest group of techniques and slowest group of techniques was CV.

c.The third question asked to arrange the methods in accordance with how the participants believed that the methods would work with many more requirements than the 13 that were in the

The result in fig. 4 clearly indicates that most of the participants thought that BST and VOP were the best techniques. NAT followed by AHP yields less accurate result. CV and PG were located between these two groups. It was expected that AHP would produce the most accurate result as in this method requirements were prioritized according to mathematical rules. An explanation to why AHP more or less did so poorly here can be that the participants did not understand how to read out, the matrix that presented the prioritization results.

e. Finally the participants were asked to keep records of how many comparisons were required for each technique. The result is shown in fig.

After collecting data based on above motioned criteria, we assigned weight for each criterion and then applied formula (2) and (3) to find out the overall best requirements prioritization technique. Each of the above criteria was assigned weight according to Table III.

Table 3. Weight table for each criterion

| Criteria | Weight |
|---|---|
| Ease of Use | 9 |
| Total time taken | 7 |
| Scalability | 8 |
| Accuracy | 8.5 |
| Total number of comparisons | 8 |

Then following formulae were used to calculate overall score by each of the prioritization techniques under consideration.

$$S_{ij} = W(C_i)*((N+1) - R_i(T_j)) \quad (2)$$

$$OS(T_j) = \frac{\sum_{i}^{NC}}{NC} Sij \quad .....(3)$$

Where,

N = Number of techniques used

$S_{i,j}$ = Score of technique j in criteria i

$W(C_i)$ = Weight of criteria i

NC = Number of criteria's

$R_i(T_j)$ = Ranking of technique j in criteria i

$OS(T_j)$ = Overall score of technique j

The result after calculation is shown in fig. 6

Fig. 6 clearly indicates that among all the requirement prioritization techniques under consideration, VOP is supposed to be the best one based on the mentioned evaluation criteria.

However, the order of the requirement prioritisation techniques that was obtained from this experiment is not a global one. Rankings can be reordered if criterion weights are assigned in a different manner, so this particular order is not definitive. Despite this, the method and formulae that were used in this article to compare various prioritisation techniques can be used in any scenario by simply adjusting the criterion weights to be appropriate for that particular setting.

The results of the experiment indicate that using VOP as a method for ranking software requirements is the most effective approach. In spite of the fact that it requires a much larger number of prerequisites, the process itself is straightforward and results in some of the highest levels of accuracy. PG and BST were not the best nor the worst solutions to the majority of the issues; rather, they were approaches that fell somewhere in the middle. However, the participants in the test ranked PG as the second-best option out of these six different ways to prioritise the items. According to the findings, NAT is the least desirable candidate. It is more difficult to determine absolute information than it is to determine relative information, participant opinions regarding a particular number have a tendency to differ greatly from one another, it is ineffective when there are few requirements, it is less accurate and informative, and it takes the longest to prioritise. These are the reasons for NAT's poor performance. Because the ranks can be rearranged if different criterion weights are provided, the order of the requirement prioritising strategies presented in this experiment is not necessarily intended to be taken as a universal one. Despite this, the approach and formulas that were used in this article to evaluate the various methods of prioritisation are adaptable to any circumstance provided that the appropriate criterion weights are used.

## CONCLUSION

The generalizability of the paper is limited because of the limited size of the sample and the specific environment that was used. Real-world projects typically have a number of requirements that are dependent on one another, in addition to time and budgetary limitations, which makes decision-making a significantly more difficult process. VOP, on the other hand, is something that we acknowledge as a valid priority method. Running the experiment as part of a case study would be beneficial because the primary limitation of the experiment is that it is difficult to generalise the results to projects in the industrial sector. Following the completion of the prioritisation training, the participating company may find a method that is suitable for their needs.

## REFERENCES

[1]. "Requirements engineering: the emerging knowledge," IEEE Software 13 (2), pp. 15-19, 1996. J. Siddiqi and M.C. Shekaran.

[2]. "Prioritizing requirements using a cost-value approach," IEEE Software 14 (5), pp. 67-74, 1997; J. Karlsson and K. Ryan.

[3]. M. Lubars, C. Potts, and C. Richter, "A review of the state of the practise in requirements modelling," Proceedings of the IEEE International Symposium on Requirements Engineering, vol. 1, no. 2, 1993, pp. 2-14.

[4]. The Analytic Hierarchy Process: Planning, Priority Setting, Resources, Allocation, T.L. Saaty, McGraw-Hill, Inc. 1980.

[5]. "An evaluation of approaches for prioritising software requirements," Information and Software Technology, pp. 939-947, 1998, by J. Karlsson, C. Wohlin, and B. Regnell.

[6]. J. Karlsson, "Software requirements priority," Proceedings of the Second IEEE International Conference on Requirements Engineering, vol. 2, no. 2, 1996, pp. 110–116.

[7]. Software Requirements, 2nd ed. K. Wiegers, Microsoft Press, 2003.

[8]. Preference-Based Decision Support in Software Engineering, by R. Vetschera, in Value-Based Software Engineering, edited by S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, and P. Grünbacher, Springer, 2006, pp. 67-89.

[9]. Requirements engineering and agile software development: Proceedings of the 12th IEEE International Workshop, F. Paetsch, A. Eberlein, and F. Maurer, IEEE Computer Society, 2003, pp. 1-6.

[10]. L. Sullivan, "Quality function deployment: A system to ensure that customer needs derive the product design and production process," Quality Progress, vol. 1, no. 1, 1986, pp. 39–50.

[11]. Software Requirements Technical Report SEI-CM-19-1.2 by J.W. Brackett was published in 1990 by the Software Engineering Institute at Carnegie Mellon University in the United States .

[12]. An Industrial Case Study on Distributed Prioritization in Market-Driven Requirements Engineering for Packaged Software, by B. Regnell, M.J. Höst, P. Beremark, and T. Hjelm, was published in Requirements Engineering, vol. 6, 2001, pp. 51–62.

[13]. "Value Oriented Requirements Prioritization in a Small Development Organization," IEEE Software, pp. 32–73, 2007, J. Azar, R. K. Smith, and D. Cordes.

[14]. Managing Software Requirements: A Use Case Approach, 2nd ed., Addison-Wesley, Boston, USA, 2003 [14]. D. Leffingwell and D. Widrig.

[15]. T. Standish, Data Structures in Java, Boston, USA: Addison-Wesley, 1997.

[16]. Extreme programming: explained, 7th edition, Addison-Wesley, Boston, USA, 2001 [16]. K. Beck

[17]. "Experimentation in software engineering," IEEE Trans. Software Engineering, vol. 12, no. 7, 1986, pp. 733–743.

[18]. S.E. Keller, L.G. Kahn, and R.B. Panara, "Specifying software quality needs with metrics," in System and Software Requirements Engineering, R.H. Thayer and M. Dorfman (Eds.), 145–163, 1990.