Volume 4, No. 1, January 2013
Journal of Global Research in Computer Science
ISSN-2229-371X
REVIEW ARTICLE
Available Online at www.jgrcs.info

COMPARISON OF SIX PRIORITIZATION TECHNIQUES FOR SOFTWARE REQUIREMENTS

Manju Khari[*1], Nikunj Kumar[2]
*[1] Department of Computer Science & Engineering, AIACT&R, Delhi, India
manjukhari@yahoo.co.in
[2] Department of Computer Science & Engineering, AIACT&R, Delhi, India
nikunjkumar14@gmail.com

***Abstract*:** There are many requirements prioritization techniques and selecting the most appropriate one is a decision problem in its own rights. This paper takes a closer look at the six requirement prioritization techniques and put them in a controlled experiment with the objective of understanding differences regarding ease of use, total time taken, scalability, accuracy, and total number of comparisons required to make decisions. These five criteria combined will indicate which technique is more suitable. The result from the experiment shows that Value oriented Prioritization (VOP) yields an accurate result, can scale up, and requires the least amount of time.

Keywords: Requirements, Requirements Prioritization, Prioritization Techniques, Comparison

INTRODUCTION

When requirements are elicited, it often yields more requirements than can be implemented at once. The requirements need to be prioritized so that the most significant ones are met by the earliest product releases [1]. During a project, decision makers in software development need to make many different decisions regarding the release plan. Issues such as available resources, milestones, conflicting stakeholder views, available market opportunity, risks, product strategies, and costs need to be taken into consideration when planning future releases. Unfortunately, there is a lack of simple and effective techniques for requirement's prioritization, which could be used for release planning [2].

Our goal in this paper is to compare six techniques for prioritizing software requirements. The chosen techniques are Analytic Hierarchy Process (AHP), Value Oriented Prioritization (VOP), Cumulative Voting (CV), Numerical Assignment Technique (NAT), Binary Search Tree (BST) and Planning Game (PG). To study these techniques, we systematically applied all techniques to prioritize a set of thirteen quality requirements. We then categorized the techniques from a user's perspective according to five criteria such as ease of use, total time taken, scalability, accuracy, and total number of comparisons required to make decisions.

MOTIVATION

In a review of the state of the practice in requirements' engineering, Lubars et al found that many organizations believe that it is important to assign priorities to requirements and to make decisions about them according to rational, quantitative data [3]. Still it appeared that no company really knew how to assign priorities or how to communicate these priorities effectively to project members. There is a growing acknowledgment in industrial software development that requirements are of varying importance. However, there has been little progress to date, either theoretical or practical, on the mechanisms for prioritizing software requirements.

A sound basis for prioritizing software requirements is the approach provided by the analytic hierarchy process, AHP [4]. In AHP, decision makers pair-wise compare the requirements to determine which of the two is more important, and to what extent. AHP has a fundamental drawback which impedes its industrial institutionalization. Since all unique pairs of requirements are to be compared, the required effort can be substantial. In small-scale development projects this growth rate may be acceptable, but in large-scale development projects the required effort is most likely to be overwhelming. Karlsson et al identified five complementary approaches to challenge AHP [5]. All of these methods involve pair wise comparisons, since previous studies indicate that making relative judgments tend to be faster and still yield more reliable results than making absolute judgments [6].

Such pair-wise comparisons are time-consuming and suffer from explosive growth as the number of requirements increases. Wiegers recommends a less rigorous approach that is based on weighted assessments of perceived value, relative penalty, anticipated cost, and technical risks [7]. The fundamental difficulty with Wiegers' approach is that the value assigned to a given requirement lacks the granularity necessary to determine whether or not the requirement meets key business core values. To overcome these limitations, there is a Value-Oriented Prioritization (VOP) process. VOP takes the form of an additive weighting method as described by Vetschera and expressed in the spreadsheet model of Wiegers [7, 8]. Paetsch et al claims that agile software development has become popular during the last few years and in this field, one of the most popular methods is the extreme programming, which has a prioritization technique called Planning Game (PG) [9]. Next section gives a brief description of each technique.

PRIORITIZATION TECHNIQUES

This section enlightens the prioritization techniques examined in this paper.

Numerical Assignment Technique (NAT):

The numeral assignment technique is based on the principle that each requirement is assigned a symbol representing the requirement's perceived importance. This approach is common in Quality Function Deployment (QFD) where prioritizing of candidate requirements is required [10]. Several variants based on the numeral assignment technique exist. A straightforward approach to the technique is presented by Brackett [11], who suggest that requirements should be classified as *mandatory*, *desirable*, or *inessential.* An approach using finer granularity is to assign each requirement a number on a scale ranging from 1 to 5, where the numbers indicate:

5. Mandatory (the customer cannot do without it).
4. Very important (the customer doesn't want to be without it).
3. Rather important (the customer would appreciate it).
2. Not important (the customer would accept its absence).
1. Does not matter.

Analytical Hierarchical Process (AHP):

The Analytic Hierarchy Process (AHP) was first developed and explained by Saaty [4] in 1980. Regnell et al [12] claim that even though this is a promising technique, the technique itself is not adapted to distributed prioritization with multiple stakeholders; hence it has to be modified in one way or another. However, at present time there have not been published any research how that kind of modification would function.

In AHP the candidate requirements are compared pair wise, and to which extent one of the requirements are more important than the other requirement. Saaty [4] states that the intensity of importance should be according to Table 1.

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*.

Since this technique prescribes pair-wise comparisons of all candidate requirements, the required number of comparisons grows polynomial. For a software system with *n* candidate requirements, *n. (n* - 1)/2 pair-wise comparisons are needed.

Value Oriented Prioritization (VOP):

VOP uses a framework that gives requirement engineers a foundation for prioritizing and making decision about requirements [13]. It provides visibility for all stakeholders during decision making, eliminating lengthy discussions and arguments over individual requirements by emphasizing the core business values. The first step in setting up a value oriented prioritization process is to establish a framework for identifying the business's core values and the relative relationships among those values. VOP uses the relationships that exist between core business values to assess and prioritize requirements and ensure their traceability. The VOP framework establishes a mechanism for quantifying and ordering requirements for an application increment, a prototype, or a software requirements specification. Company executives identify the core business values and use a simple ordinal scale to weight them according to their importance to the organization.

Table 2. Value Oriented Prioritization matrix

Requirements	***Business Values (V_1…..V_n)***					***Score***
	V_1=7	V_2=6	V_i=9	V_{I+1}=5	V_n=8	
R_1						
R_2		W_{ij}				
….						
R_N						

Table 2 shows an example of a matrix incorporating five business values and. $V_{0,i}$ is the weight of business value i. $W_{i,j}$ is the weight assigned to requirement r_i with respect to business value V_j. Formally, we can express the score ($\mathbf{S_r}$) for each requirement **r**, in the set, **R** of all possible requirements, as:

$$\mathbf{S_r} = \sum_{i=1}^{n} (\boldsymbol{Vi} \times \boldsymbol{Wr,i}) \quad (1)$$

Cumulative Voting (CV):

The Cumulative Voting (CV) or 100-Point Method or Hundred-Dollar ($100) test, described by Leffingwell and Widrig, is a simple, straightforward and intuitively appealing voting scheme where each stakeholder is given a constant amount (e.g. 100, 1000 or 10000) of imaginary units (for example monetary) that he or she can use for voting in favor of the most important issues [14]. In this way, the amount of money assigned to an issue represents the respondent's relative preference (and therefore prioritization) in relation to the other issues. The points can be distributed in any way that the stakeholder desires. Each stakeholder is free to put the whole amount given to him or her on only one issue of dominating importance. It is also possible for a stakeholder to distribute equally the amount to many of, or even to all of the issues.

CV is sometimes known as "proportional voting" since the amount of units assigned to an issue represents the relative priority of the specific issue in relation to the other issues. The term "proportional" in this case also reflects the fact that if the amount of units assigned to an issue is divided by the constant number of units available to each stakeholder, the result becomes a proportion between zero and one. The stakeholder's ratings for a set of issues can be therefore considered as the "composition" or "mixture" of a person's opinion towards the issues, in the abstract sense that each issue occupies a certain proportion (or percentage) of preference inside the person's belief or judgment.

The procedure may result to issues that are assigned zero units showing that the specific stakeholder considers these issues completely unimportant. The zeros are generally a problem in this kind of data, because they make the notion of relative preference or importance completely meaningless and the computation of ratios impossible. Of course, a questionnaire where zeros are not allowed could be designed, but in general, the principle of CV is to allow stakeholders to spread freely their total amount without further restrictions.

Binary Search Tree (BST):

BST is a computer algorithm with the purpose to store information, which then could be retrieved or sought after. The BST *T* usually is either empty, or has one or two child nodes. The child nodes to the right (T_r) have greater value/importance than the root node *R*, and the child nodes to the left (T_l) have less value/importance then the root node *R*. Each child node is in itself a root node to its child node. If a node does not have any child nodes, it is called a leaf. This makes it possible to search in the BST recursively. The benefit for using BST, when prioritizing requirements, is that with *n* requirements, it takes only *n* log *n* [15] comparisons until all the requirements have been inserted in order. That makes BST a fast candidate, which could be good if there is a lot of requirement to prioritize among, i.e. BST could easily scale up to thousands of requirements, and still be a very fast candidate. There is one important thing to know about the BST algorithm, which is that a tree needs to be balanced to have the shortest insertion time.

A balanced BST is a BST where no leaf is more than a certain amount farther from the root than any other leaf. After a node has been inserted or deleted the tree might have to be rebalanced if but only if the BST would reach an unbalanced stated. The reason for this is that the insertion of a node should be optimal, i.e. log *n*.

The scale between each requirement is on the ordinal scale. That means that I only could find out if one requirement is more important than another, but not to what extent. Another negative problem with BST is that there is no consistency ratio that we could calculate, hence we do not know if we have done a precise prioritizing or not.

Planning Game (PG):

In extreme programming the requirements are written down by the customer on a story card. Then the customer divides the requirements into three different piles. According to Beck, the piles should have the names; "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" [16]. At the same time as that the customer sorts the story cards, the programmer estimates how long time each requirement would take to implement and then begin to sort the requirements into three different piles, i.e. sort by risk, with the names; "those that can be estimated precisely", "those that can be estimated reasonably well" and "those that cannot be estimated at all".

The customer or one or several representatives for the customer could either decide on a fixed release date, or decide which requirements that should be included in the next release. The end result of this sorting is a sorted list of requirements on an ordinal scale. Since PG takes one requirement and then decides which pile the requirement belongs to and each requirement is not being compared to any other requirement, the time to prioritize *n* requirements is *n* comparisons. This means that PG is very flexible and can scale up to rather high numbers of requirements, without taking too long time to prioritize them all.

EXPERIMENT FRAMEWORK

This section describes the experiment design and how the experiment will be conducted.

Introduction:

The aim of the experiment is to compare the six prioritizing techniques to evaluate which one of them seems to be the better, i.e. which technique is the easiest to use, takes shortest amount of time, scalable when adding more requirements, accurate and takes fewer number of comparisons. This is tested by letting the participants' answer how they experience and believe that each technique would be able to fulfill each criterion. This experiment is highly influenced by the experimental approach outlined in [17].

Design:

With the motivation of gaining a better understanding of requirements prioritization techniques, we performed a single project study with the aim of characterizing and evaluating the six prioritizing techniques from the perspective of users [17]. The experiment was populated with seven graduate and post graduate students. They were asked to prioritize thirteen quality requirements using the prioritization techniques under consideration [18]. The requirements were prioritized by the participants independently, and to the best of their knowledge. The quality requirements were prioritized without taking the cost of achieving the requirements into account. That is, only the importance for the customers was considered. Moreover, the requirements were considered orthogonally, i.e. the importance of one requirement is not interdependent on another.

In order to minimize the risk that the participants remember how they did the last prioritization, we spread the test over a period of time with fixed intervals. Only one technique was studied in a day. Every day, 20 minutes were allocated for presenting the technique which was under observation on that day and after getting the confirmation from each participant whom the technique was understood clearly, 60 minutes were allocated for completion of the experiment of that day. Each participant was supplied with necessary papers and time taken by each participant to complete the experiment was recorded separately.

Threats to Validity:

When reading a result from an experiment, one of the most important questions is: How valid is the result? That makes validity of the result an important question to consider when an experiment is designed. The aim of the experiment was the evaluation of six requirements prioritization techniques by making comparisons among them. We do not argue that the results obtained in this experiment can be generalized and used by any user in any environment for any

application. Rather, we tried to illustrate the requirements prioritization techniques to gain a better understanding of them. The following threats have been identified:

***Too few requirements*:** In the analysis of the data, it became obvious that the experiment had too few requirements. However, before the experiment, it was discussed whether it would be possible to consider more than thirteen requirements, but since there was a time limit, i.e. how much time the participants could participate; the number of requirements had to be limited. To really reflect a real project, the number of requirements should be a couple of hundred; this would be more or less impractical to handle within the limited timeframe of this experiment. Therefore, the decision was taken that the number of requirements should only be thirteen.

Few persons involved in the experiment: The significance of the results is limited due to involvement of few persons (seven persons) with the experiment. That's why the outcomes were more inconclusive, and hence can be regarded as a partial threat to the evaluation. However, if requests to attend to the experiment are going to a large population, there is a greater chance that the risk would be minimized.

Offline Evaluation: The evaluation was carried out independently from a real software project which may be considered as a potential problem for this experiment. However, it is not regarded as being a major threat as the main objective of this evaluation was to gain understanding and illustrate a number of possible methods for prioritizing software requirements.

Only non functional requirements considered: This experiment was only concerned with non functional requirements. This limitation is, however, not believed to be a major threat to the results from the experiment.

Requirements are interdependent: In practice, the interdependence between the requirements must be considered. None of the prioritizing techniques described in this paper provides means for handling interdependence; hence this limitation of the experiment is not believed to influence the actual evaluation of the different methods.

It is always important to identify threats in an experiment in order to allow for determining both the internal and external validity of the results attained. Thus, the above potential threats should be kept in mind when analyzing the results.

Analysis of collected data:

The testing begins with the first question of every technique; followed by the second and third and so on. For each question, participants ranked each method and finally mean value was taken. Those questions that the participants were asked after each technique were the following:

a. The first question that the participants were asked was how easy the prioritization technique was to apply. The answer of the question is shown in fig. 1.

Fig. 1 clearly indicates that participants thought that Planning Game (PG) followed by VOP was the easiest method to apply. NAT followed by AHP was most difficult to handle. CV and BST were in the middle of these two groups.

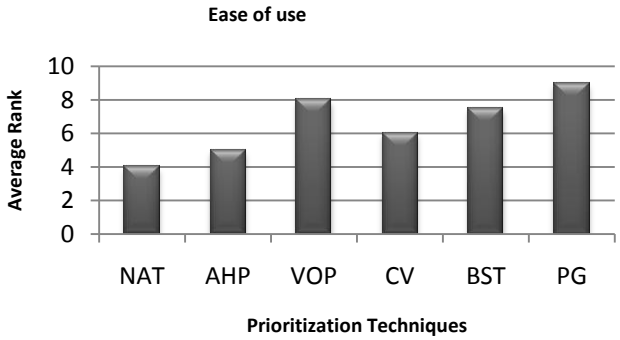

Figure 1. Comparison among the techniques for the criteria "Ease of use"

b. The second question that the participants were asked was how long time it took for the participants to perform the prioritization with the techniques under consideration. The result of the question is shown in fig. 2.

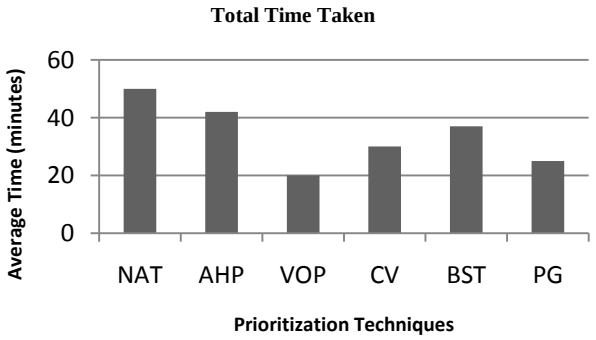

Figure 2. Comparison among the techniques for the criteria "Total time taken to prioritize"

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 was to arrange the methods according to how the participants believed that the methods would work with many more requirements than the 13 that were in the experiment. The result is presented in fig. 3.

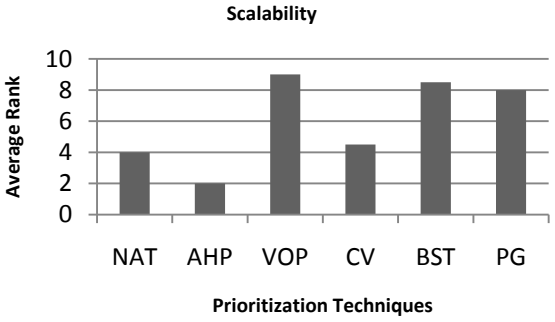

Figure 3. Comparison among the techniques for the criteria "Scalability"

The result in fig. 3 indicates most of the participants thought VOP, and BST were the prioritization techniques that were more suited as candidates to handle much more

requirements. The participants found that AHP followed by NAT would be the worst candidate to scale up for more requirements. In the middle was PG.

d. The fourth question was that the participants were asked to arrange the techniques according to their opinion about accuracy of the result produced by each method. The result is shown in fig. 4

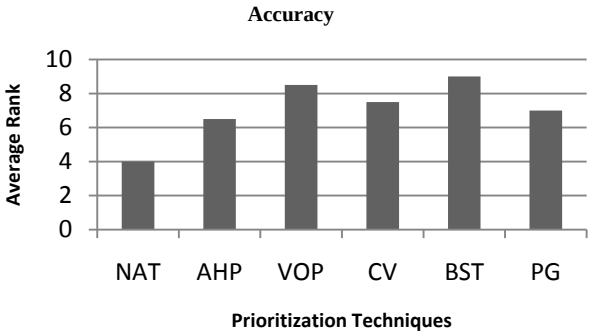

Figure 4. Comparison among the techniques for the criteria "Accuracy"

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. 5.

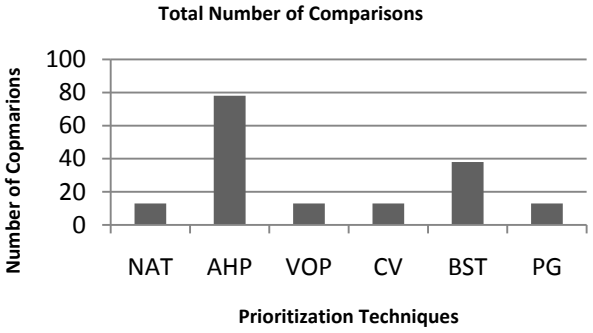

Figure 5. Comparison among the techniques for the criteria "Total Number of Comparisons"

The result in fig. 5 clearly indicates that AHP was required the highest number of comparisons because the number of comparisons in AHP is n(n-1)/2. NAT, VOP, CV, and PG were required the lowest number of comparisons because they only require n comparisons. BST was in the middle of these two groups, because it require n(logn) comparions.

FINDING THE OVERALL BEST PRIORITIZATION TECHNIQUE

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.

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

$$OS(T_j) = \frac{\sum_{i=1}^{NC} Ci,j}{NC} \quad (3)$$

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.

This order of the requirement prioritization techniques obtained from this experiment, however, is not a global one as rankings can be reordered if criterion weights are assigned differently. Nevertheless, the technique and formulae used here to compare among different prioritization techniques can be used in any scenario with appropriate criterion weights suitable for that scenario.

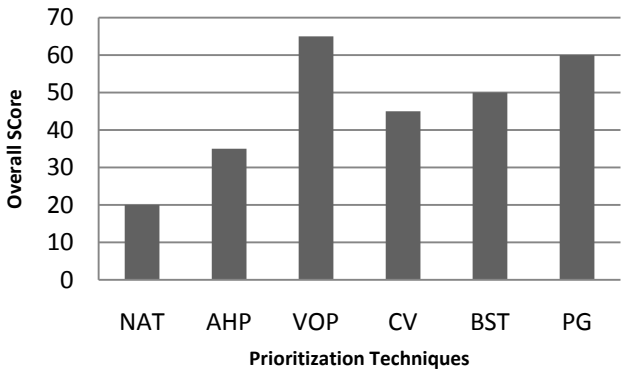

Figure 6. Comparison among the techniques on the basis of weighted value of criteria's

CONCLUSION

Outcome of the experiment says that VOP is supposed to be the best method for prioritizing software requirements. It is an easy method, it gives one of the most accurate results, and it is rather comfortable to handle even if there are many

more requirements. In most questions' PG and BST were located in the middle, neither the best nor the worst techniques. However, the test subjects thought that PG was the next-best method of these six techniques to be used when prioritizing. The worst candidate according to result is NAT. The reasons for worst performance of NAT are determining the absolute information is difficult than relative information, participants' subjective opinions regarding a number differ widely, it is not effective when numbers of requirements are low, less accurate and informative, it takes maximum time to prioritize. However, this order of the requirement prioritization techniques obtained from this experiment, however, is not a global one as rankings can be reordered if criterion weights are assigned differently. Nevertheless, the technique and formulae used here to compare among different prioritization techniques can be used in any scenario with appropriate criterion weights suitable for that scenario.

The generalisability of the paper is limited due to the small sample and the specific context. A real project has requirement's interdependencies, and time and budget pressure to consider, which cause the decision-making to be far more difficult. However, we believe that VOP is valid as prioritization technique. The main disadvantage of the experiment being the difficulty to generalize to industrial projects, it would be valuable to try the experiment out in a case study. The participating organization would then get knowledge about prioritization and perhaps find a technique that suits their needs.

REFERENCES

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

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

[3]. M. Lubars, C. Potts, and C. Richter, "A review of the state of the practice in requirements modeling," in: Proceeding of the IEEE International Symposium on Requirements Engineering, pp. 2–14, 1993.

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

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

[6]. J. Karlsson, "Software requirements prioritizing," in: Proceeding of 2nd IEEE International Conference on Requirements Engineering, pp. 110–116, 1996.

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

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

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

[10]. L. Sullivan, Quality function deployment: A system to assure that customer needs derive the product design and production process, Quality Progress, pp. 39-50, 1986.

[11]. J.W. Brackett, Software Requirements Technical Report SEI-CM-19-1.2, Software Engineering Institute, Camegie Mellon University, USA, 1990.

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

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

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

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

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

[17]. V.R. Basili, R.W. Selby, and D.H. Hutchens, "Experimentation in software engineering," IEEE Trans. Software Engineering 12 (7), pp. 733- 743, 1986.

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

Short Bio Data for the Author

Ms. Manju Khari has received her M.Tech in Computer Science from Guru Gobind Singh Indraprastha University, Delhi. Currently she is pursuing P.HD from Delhi Technological University, Delhi. Her research interest encompasses network security, and different sector of software engineering mainly software testing. She has coauthored in various research papers published in various International journals and conferences proceedings. She is working as Assistant Professor in Department of Computer Science and Engineering of AIACT&R, geeta colony, Delhi.

Nikunj Kumar has received his B.Tech in Information Technology form USIT, Guru Gobind Singh Indraprastha University, Delhi in 2011. He is pursuing M.Tech in Information Security from AIACT&R, Guru Gobind Singh University, Delhi. His area of interest encompasses internet security, and various areas of software engineering include requirement engineering, software testing.