An Evaluation of Software Requirement Prioritization Techniques

Mohammad Shabbir Hasan[1, a], Abdullah Al Mahmood[2, a], Md. Jahangir Alam[3, b], Sk. Md. Nahid Hasan[4, a], Farin Rahman[5, a]

E-mail: [1]*shabbir_cse03@yahoo.com*, [2]*sajibcseofkuet@gmail.com*,[3]*palash_14@yahoo.com*, [4]*auvi_2k5@hotmail.com*,[5]*farinrahman.cse@gmail.com*,

a. Panacea Research Lab, Dhaka, Bangladesh

b. Dhaka International University, Dhaka, Bangladesh

***Abstract*— Requirements prioritization plays an important role in the requirement engineering process, particularly, with respect to critical tasks like requirements negotiation and software release planning. Selecting the right set of requirements for a product release largely depends on how successfully the requirement candidates are prioritized. There are different requirement prioritization techniques available which are some more elaborated than others. This paper takes a closer look at nine different techniques of requirement prioritization namely Analytical Hierarchy Process (AHP), Hierarchy AHP, Minimal Spanning Tree, Bubble Sort, Binary Search Tree (BST), Priority Group, Planning Game (PG), 100 points method and Planning Game combined with AHP (PGcAHP) and then put them into a controlled experiment, in order to find out the best one. The evaluation was done on the basis of some criteria like: ease of use, certainty, accuracy of result, method's ability to scale up to many more requirements, required number of comparisons, and required time to make decision. Analysis of the data from the experiment indicates that the analytic hierarchy process to be a promising candidate, although it may be problematic to scale-up. However, the result clearly indicates that the Planning Game (PG) yields accurate result, is able to scale up, requires least amount of time, the easiest method to use and so on. For these reasons, finding of the experiment is, the Planning Game (PG) method is supposed to be the best method for prioritizing requirements.**

Keywords- Requirement Engineering, Requirement Prioritization, Requirement Negotiation, Software Product Management, Software Release Planning.

I. INTRODUCTION

In market-driven software development, products which are intended for an open market are developed in several consecutive releases. Market-driven development does not have easily identifiable customers and the requirements often need to be invented based on the needs of several potential users [1]. When only one stakeholder is involved in the project, it is relatively easy to make decisions since only one stakeholder's opinion needs to be considered. But when more than one stakeholder is involved in the project, decisions can be harder to make, since different stakeholders have different perspectives and in this case, more requirements are yield than can be implemented at once. Again, not all the requirements contain equal user satisfaction. For example, project developers look for the requirements which can be implemented fast, financial managers look for the requirements with low cost, market managers look for the requirements with high market value, and end users look for the requirements which are easy to use. One requirement may be of low cost, with short implementation time, but also have low market value and be hard to use. Conversely, another requirement may have a high cost, but short time to be implemented, high market value and be easy to use. It can be a challenge for the software development team to decide which requirements need to be implemented first. Requirements prioritization is a technique that can uncover the most important requirements to maximize the stakeholders' satisfaction.

In commercial software development, decision makers need to make many different decisions regarding the release plan according to some issues like available resources, milestones, conflicting stakeholder views. Available market opportunity, risks, product strategies, and costs need to be taken into consideration when planning future releases. Nowadays, unfortunately, projects are suffering low success rates. According to an annual report named 'CHAOS Summary 2009' prepared by Standish Group [27], only 32% of all projects were considered as successful which are delivered on time, on budget, with required features and functions. Among the rest, 44% were challenged which are late, over budget, and/or with less than the required features and functions and 24% failed which are cancelled prior to completion or delivered and never used. Ten main factors causing challenged or failed projects are unveiled. Four of them are lack of user involvement, lack of resources, unrealistic expectations, and changing requirements and specifications. Requirements prioritization increases user involvement by letting the stakeholders decide which requirements the project should contain. It helps stakeholders to be realistic by letting them understand the current constraints on resources and accepting the trade-off

http://sites.google.com/site/ijcsis/
ISSN 1947-5500

decisions on conflicting perspectives. Karlsson et al think it helps stakeholders to allocate resources based on the priorities of the requirements [2], detect requirements defects, such as misunderstanding or ambiguous requirements [3] and reduce the number of changes to requirements and specifications in the later stage of projects. Hatton [4] says requirements prioritization has become an essential step in the software development process in order to reduce software failure. Ngo-The and Ruhe [5] note requirements prioritization has been recognized as one of the most important decision making processes in the software development process.

Several approaches have been proposed [6-10] which adopts a common model for the requirements prioritization process. This paper provides an investigation of nine candidate methods for prioritizing requirements: Analytic Hierarchy Process (AHP), Hierarchy AHP, Minimal Spanning Tree, Bubble Sort, Binary Search Tree (BST), Planning Game (PG), Priority Group, 100 points method and Planning Game Combined with AHP (PGcAHP). To study these methods, we systematically applied all methods to prioritize 14 well-defined quality requirements of a mobile set. We then categorized the methods from a user's perspective according to a number of criteria such as accuracy, certainty, method's ability to scale up to many more requirements, required time to make decision, total number of decisions and ease of use.

This paper is organized as follows. Section 2 motivates this work, and the paper continues in Section 3 by outlining the nine different prioritizing methods. Section 4 describes the evaluation framework and Section 5 presents the way to find out the best one among the techniques under consideration followed by a discussion of the result in Section 6. We finish by drawing some broad and necessarily speculative and personal conclusions in Section 7.

II. MOTIVATION

Industrial software development has a growing acknowledgement that requirements are of varying importance. Yet there has been little progress to date, either theoretical or practical, on the mechanisms for prioritizing software requirements [11]. In a review of the state of the practice in requirements engineering, Lubars et al. [12] 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. Still it appeared that no company really knew how to assign priorities or how to communicate these priorities effectively to project members [3].

A sound basis for prioritizing software requirements is the approach provided by the analytic hierarchy process, AHP [13] where decision makers compare the requirements pair-wise to determine which of the two is more important, and to what extent. In industrial projects, this approach has been experienced as being effective, accurate and also to yield informative and trustworthy results [7]. Probably even more important, after using the approach in several commercial projects, practitioners are found to be very attracted by the approach, and continue to use it in other projects [3]. AHP has only been used in few applications in the software industry. Finnie et al. [14], for example, used AHP to prioritize software development factors. Other applications of AHP include a telecommunications quality study performed by Douligeris and Pereira [15], and software requirements prioritizing in a commercial development project by Karlsson [16]. Despite some positive experience, 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 [3].

Since AHP may be problematic for large-scale projects, Karlsson et al [3] identified five complementary approaches to challenge AHP. All of these methods involve pair-wise comparisons, since previous studies indicate that making relative judgments tends to be faster and still yield more reliable results than making absolute judgments [7]. They focused on methods which may reduce the required effort, but still able to produce high-quality results, considered trustworthy by its users. Again, Paetsch et al [17] 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). In this paper, we investigated PG with all the requirement prioritization techniques used in the experiment carried out by Karlsson et al [3]. We also investigated a rather easy and quick method (at least according to the theory), and that is the 100 points method. Next section gives a brief description of each method, both in theory and then how it works practically.

III. PRIORITIZATION METHODS

Prioritizing methods guide decision makers to analyze requirements to assign numbers or symbols that reflect their importance. According to Karlsson et al [3], a prioritizing session may consist of three consecutive stages:

(1) The *preparation* stage where a person structures the requirements according to the principle of the prioritizing methods to be used. A team and a team leader for the session is selected and provided all necessary information.
(2) The *execution* stage where the decision makers do the actual prioritizing of the requirements using the information they were provided with in the previous stage. The evaluation criteria must be agreed upon by the team before the execution stage is initiated.

http://sites.google.com/site/ijcsis/
ISSN 1947-5500

(3) The *presentation* stage where the results of the execution are presented for those involved. Some prioritizing methods involve different kinds of calculations that must be carried out before the results can be presented.

This section describes the prioritization techniques investigated in this paper:

A. Analytic Hierarchy Process (AHP)

The Analytic Hierarchy Process (AHP) was first developed and explained by Saaty [13] in 1980. Regnell et al [18] 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 is more important than the other requirement. Saaty [13] states that the intensity of importance should be according to Table 1.

TABLE I. BASIC SCALE ACCORDING TO SAATY [13] FOR PAIR-WISE COMPARISONS IN AHP

How Important	Description
1	Equal importance
3	Moderate difference in importance
5	Essential difference in importance
7	Major difference in importance
9	Extreme difference in importance
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*.

To fully understand AHP it is easiest to divide AHP into three different phases.

1. Comparing every pair of requirements, this is the "engine" of AHP, according to Saaty [19].
2. Derives a priority vector of relative weights for these requirements, i.e. the principal eigenvector.
3. Calculate the by-product from 2, i.e. the inconsistency measure.

First we take the requirements that should be prioritized (the total amount of requirement is *n*), and put them into a matrix, where the rows have the index of *i* and columns have the index of *j*. The matrix is called *W* and the elements in the matrix are called *w*. The requirement that is placed in row *i* and column *j* gets the index *ij*. Therefore the element w_{ij} has the row index = *i* and column index = *j*.

TABLE II. MATRIX OF PAIR WISE COMPARISONS

	Requirement 1	Requirement 2	…….	Requirement n
Requirement 1	1	w_{12}	w_{1j}	w_{1n}
Requirement 2	w_{21}	1	w_{2j}	w_{2n}
……	w_{j1}	w_{j2}	1	w_{jn}
Requirement n	w_{n1}	w_{n2}	w_{nj}	1

Each matrix element consists of the comparison between two requirements (*i* and *j*), which gives us the following relationship:

$$w_{ij} = \frac{w_i}{w_j} \quad (1)$$

An important notice is that the person that does the prioritization does not put any value on w_i and w_j, instead he or she decides the value for w_{ij} which is the ratio between w_i and w_j. That leads us to another important relationship, which is that for every index of *i*, *j*, *k* has the following relationship:

$$w_{ij} = w_{ij}^{-1}, \; w_{ij} = w_{ik}\, w_{kj} \quad (2)$$

With the information from formulae (1) and (2) and the matrix Table 2 we can see that some pair-wise comparisons are doing twice. The problem with human perception and judgments are subject to change if the human becomes tired or something changes the human psychological state (i.e. the level of blood sugar is dropping, and thereby the concentration). To solve this problem, Saaty [13] proposed that we should only compare a_{ij}, $j > i$. With this solution we do not need to do n^2 comparison. Instead we only need to do half the comparison, since the formulae (2) say that $w_{ji} = \frac{1}{w_{ij}}$. So it is really easy to apply this formula (2) to the comparisons that are not necessary. This leaves us to the diagonal, with the comparison with requirement w_i and w_i they will always be equal (i.e. the reciprocal value 1). Hence, we do not need to do this comparison either. This led us to the formulae (3):

$$\text{Total number of comparisons} = \frac{n(n-1)}{2} \quad (3)$$

The next step according to Saaty [13] is to calculate the eigenvector *v*. The elements of the eigenvector correspond to the priorities of the requirements. Gass and Rapcsák [20] describe it in the following way: If W is a consistent matrix, i.e. formulae (2) holds for all the indices of i, j, k, then *w* is of rank one and $\lambda_{max} = n$. Iff the relationship $\lambda_{max} = n$ is true, W is a positive reciprocal matrix.

$$Wv = \lambda v \quad (4)$$

http://sites.google.com/site/ijcsis/
ISSN 1947-5500

The formula (4) is the mathematical definition on the relationship between the eigenvalue and the eigenvector. This is nothing that is specific for AHP, but is valid for all matrices.

This means that *v* must be the eigenvector of W that correspond to the maximum eigenvalue λ. What this mean in reality is that every requirement is in the matrix and then the sum of j columns is calculated.

$$w_{11}+w_{21}+ w_{i1}+\ldots+ w_{(n-1)1} +w_{n1} = z \quad (5)$$

Then each element in the column is divided by the sum, *z*, calculated from formulae (5). In the next step, elements in row *i* are added and then each row sum is divided by the total number of requirements *n*. This results a matrix representing the normalized eigenvector of the comparison matrix. Based on the elements in the normalized eigenvector, also known as the priority vector, conclusion about the importance of the requirements can be drawn. The degree of importance could be what kinds of business value that requirement yields or what it costs to develop or any other kind of importance. All depending on what aspect that the person that prioritized had in his/her mind during the prioritization.

The final step is to calculate how consistent the prioritization has been done. The reason of calculating this consistency is: If a person prioritizes that A is more important than B, B is more important than C and finally, C is more important than A, this will mean that C is more important than C, which cannot be true, i.e. the person has done a judgment error, hence it is important to find out if the person is consistent in his/her judgment. The consistency index (CI) is calculated by the formulae (6)

$$CI = \frac{\lambda_{max} - n}{(n-1)} \quad (6)$$

λ_{max} is the maximum eigenvalue of the matrix. If the λ_{max} value is close to *n*, the number of requirements, then there have been little judgment errors and the result is more consistent.

To check whether CI is acceptable, Consistency Ratio (CR) is calculated. The CR is a ratio from CI and RI, where RI is one of the Random Indices [13]. The value of RI can be obtained from Table 3:

TABLE III. RANDOM INDICES FOR AHP

1	2	3	4	5	6	7	8	9	10	11	12	13
0.00	0.00	0.58	0.90	1.12	1.24	1.32	1.41	1.45	1.48	1.49	1.51	1.56

$$CR = \frac{CI}{RI} \quad (7)$$

According to Saaty [13] a result of CR within the range of 0.10 or less is to be considered acceptable.

All this calculation is the strong part of AHP, which give us a good way to evaluate whether the decision take is good or not. AHP has been used to predict a variety of decision, from stock market portfolio, economic forecasts, oil prices, political candidate, etc [3].

However AHP has some weakness. Since it takes $\frac{n(n-1)}{2}$ comparisons, it does not work well for a large number of requirements. For example if the amount of requirement is 10, then it takes 45 comparisons. If the number of requirements is 20, then it takes 190 comparisons, and if the number of requirements is 100, it takes 4950 comparisons. A software project could have up to several thousand requirements, which mean that the number of comparisons exceeds over 500 000, and it will take too long time to compare all the requirements, and the question would then be; should AHP be used or not?

A-1. How to implement AHP in Requirement Prioritization:

(1) As preparation, all unique pairs of requirements are outlined.

(2) As execution, all outlined pairs of requirements compared using the scale in Table 1.

(3) As presentation, the above mentioned method is used to estimate the relative priority of each requirement. The consistency ratio of the pair-wise comparisons is calculated using the methods mentioned above. The consistency ratio is an indicator of the reliability of the resulting priorities, and thus also an estimate of the judgmental errors in the pair-wise comparisons.

B. Hierarchy AHP

In large-scale development projects the requirements are often structured in a hierarchy of interrelated requirements [21]. The most generalized requirements are placed at the top of the hierarchy and the more specific requirements on levels below. Hierarchies are a common structure in daily use of AHP. But, to separate this hierarchical requirements structure from the flat requirements structure outlined previously, Karlsson et al [3] use the name *Hierarchy AHP*, we also use the same name in this paper.

B-1. How to implement Hierarchy AHP in Requirement Prioritization:

(1) As preparation, all unique pairs of same level requirements are outlined in the hierarchy. Not all requirements are pair-wise compared to each other, but only those at the same level.

http://sites.google.com/site/ijcsis/
ISSN 1947-5500

(2) As execution, all outlined pairs of requirements are compared using the scale in Table 1.
(3) As presentation, methods used for AHP are also applied at each level of the hierarchy. The priorities are then propagated down the hierarchy.

Hierarchy AHP possesses similar characteristics to AHP. Using a hierarchical structure reduces the required number of decisions, but also the amount of redundancy. Thus it is more sensitive to judgmental errors than AHP [3].

C. Minimal Spanning Tree

The pair-wise comparisons in AHP provide interesting relationships to each other. For example, if requirement A is determined to be of higher priority than requirement B, and requirement B is determined to be of higher priority than requirement C, then requirement B should be of higher priority when compared to requirement C. Despite this, AHP lets the decision maker perform the last comparison. Because of this redundancy AHP can indicate inconsistent judgments (such as claiming B to be of higher priority than C in this example) [3].

The redundancy of the comparisons would be unnecessary in case of the decision makers being perfectly consistent. In such a case only (n – 1) comparisons would be enough to calculate the relative intensity of the remaining comparisons. This implies that the least effort required by a decision maker is to create a minimal spanning tree in a directed graph (i.e. the graph is at least minimally connected). In the directed graph which can be constructed by the comparison provided, there is at least one path between the requirements not pair-wise compared [3].

C-1. How to implement Minimal Spanning Tree in Requirement Prioritization:

(1) As preparation, (n – 1) unique pairs of requirements are outlined to construct a minimal spanning tree.
(2) As execution, all outlined pairs of requirements are compared using the scale in Table 1.
(3) As presentation, the missing intensities of importance are computed by taking the geometric mean of the existing intensities of all possible ways in which they are connected. After that, AHP is used as usually.

The minimal spanning tree approach is supposed to be very fast as it dramatically reduces the number of pair wise comparisons. On the other hand, it is more sensitive to judgmental errors since all redundancy has been removed.

D. Bubble Sort

Bubble sort is one of the simplest and most basic methods for sorting elements with respect to a criterion [22]. It is also a candidate method for prioritizing software requirements, since the actual prioritizing process can be viewed as sorting requirements (i.e. the elements) according to their priorities (i.e. the criterion) [3].

Interestingly, bubble sort is closely related to AHP. As with AHP, the required number of pair wise comparisons in bubble sort is $\frac{n(n-1)}{2}$. But, the decision maker only has to determine which of the two requirements is of higher priority, not to what extent [3].

D-1. How to implement Bubble Sort in Requirement Prioritization:

(1) As preparation, all requirements are outlined in a vector.
(2) As execution, all requirements are compared according to bubble sort algorithm.
(3) As presentation, the sorted vector is outlined. The result of the process is a vector where the original order of the requirements has changed. The least important requirement is at the top of the vector, and the most important requirement is at the bottom of the vector.

The result of a bubble sort is requirements are ranked according to their priority on an ordinal scale.

E. Binary Search Tree

A binary tree is a tree in which each node has at most two children. A special case of a binary tree is a binary search tree where the nodes are labeled with elements of a set [22]. Consider the elements of the set as the candidate requirements. This is of interest for prioritizing purposes since an important property of a binary search tree is that all requirements stored in the left sub tree of the node x are all of lower priority than the requirement stored at x, and all requirements stored in the right sub tree of x are of higher priority than the requirement stored in x. If the nodes in a binary search tree are traversed using in order traversing method, then the requirements are listed in sorted order. Consequently creating a binary search tree with requirements representing the elements of a set becomes a method for prioritizing software requirements [3].

Prioritizing n software requirements using the binary search tree approach involves constructing a binary search tree consisting of *n* nodes. The first thing to be done is to create a single node holding one requirement. Then the next requirement is compared to the top node in the binary search tree. If it is of lower priority than the node, it is compared to the node's left child, and so forth. If it is of higher priority than the node, it is compared to the node's right child, and so forth. Finally the requirements are inserted into the proper place and the process continues until all requirements have been inserted into the binary search tree [3].

E-1. How to implement Binary Search Tree in Requirement Prioritization:

(1) As preparation, all candidate requirements are outlined.

http://sites.google.com/site/ijcsis/
ISSN 1947-5500

(2) As execution, select the requirements one at a time and a binary search tree is created.
(3) As presentation, the binary search tree is traversed using in order traversing and nodes are added to a list. The requirements having the lowest priority then come first in the list. Then the list is printed.

Since the average path length from the root to a leaf in a binary search tree is *O(log n)*, inserting a requirement into a binary search tree takes on the average *O(log n)* time. Consequently, inserting all n requirements into a binary search tree takes on the average *O(n log n)* time. In this case, too, the requirements are ranked on an ordinal scale.

F. Priority Groups

In some software development projects, one set of requirements can clearly be of a different kind of importance than another set. One way to reduce the required effort is therefore not to compare the requirements in these distinct sets. Thus another candidate method is to initiate the prioritizing process by dividing the requirements into separate groups based on a rough prioritization. Subsequently, the groups can be internally ranked either by using a suitable approach for ordering the requirement, for example, using AHP or to continue with another grouping of even finer granularity [3].

The primary gain is that, it is not necessary to compare high priority requirements with requirements of low priority, since they are placed in different groups. The actual choice of the number of groups depends on the situation as well as the knowledge of the people performing the prioritization. A simple strategy suggests using three distinct groups: low, medium and high priority. It may even be the case that the high priority requirements must be implemented, and hence there is no need to prioritize between them. In the same way the low-priority requirements may perhaps be postponed to a later release [3].

F-1. How to implement Priority Groups in Requirement Prioritization:

(1) As preparation, all candidate requirements are outlined.
(2) As execution, each of the requirements is put into one of the three groups. In groups with more than one requirement, three new subgroups are created and the requirements are put into these groups. This process is continued recursively to all groups.
(3) As presentation, the requirements are printed from left to right.

To guarantee that the correct ordering of the requirements is obtained, it is necessary to ensure that the tail of one group is having higher priority than the head of the following group. This comparison between tail and head in the groups must continue until the requirements are in the correct order. This is one way of minimizing the risk of ending up with the requirements in the wrong order. The priority grouping approach can hence be divided into two possible approaches: grouping without tail-head comparison and grouping with tail-head comparison [3].

G. Planning Game (PG)

In extreme programming the requirements are written down by the customer on a story card which is then divided into three different piles. According to Beck [23], 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”. At the same time, 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”. Final 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.

G-1. How to implement Planning Game in Requirement Prioritization:

(1) As preparation, all candidate requirements are outlined.
(2) As execution, each requirement is taken and put in the appropriate pile. This process continues until all requirements are sorted.
(3) As presentation, all requirements in the pile “those without which the system will not function” are considered and system is developed with them.

H. 100 Points Method

In this method each stakeholder gets hundred points of some value. With these points they should “purchase ideas”. Each person writes down on a paper how much he/she thinks that one requirement is worth. When all the participants have written down their points, one person calculates, by taking the paper and summing up the points that each requirement has got, and presents the cumulative voting results. The requirement that has got the highest score is the most important requirement.

Theoretically 100P is equally flexible as PG when it comes to the number of comparisons, i.e. *n* requirements takes *n* comparisons. Hence, it should be a really fast and scalable method, also in comparison to both AHP and BST. However, even though it has the same amount of comparisons as PG, i.e. *n*, it probably would take longer time to do the actual comparisons. The reason for this is that, while in PG the decision should be in which pile to put a requirement, i.e. ordinal scale, which is the same scale as BST, in BST the decision should be if requirement *A* is more or less important than requirement *B*. For 100P the scale is ratio, which is the same scale as for AHP. So the person that

http://sites.google.com/site/ijcsis/
ISSN 1947-5500

does the prioritization has to consider to which extent one requirement is more or less important than the other. At the same time he/she has only a small amount of points to distribute, which probably also takes some time to take into account in the distribution of points to the different requirements.

H-1. How to implement 100 Points Method in Requirement Prioritization:

(1) As preparation, all candidate requirements are outlined.
(2) As execution, 100 points are distributed among the requirements, according to their importance.
(3) As presentation, points are summed up for each requirement and requirements sorted according to their total point.

I. Planning Game Combined with AHP (PGcAHP)

Karlsson et al [24] did a study regarding the difference between PG and AHP where in the discussion section; they stated that it would be interesting to combine PG with AHP, so that this combined method would use the strengths in each method and eliminate their weaknesses.

The strength of PG is that it is rather fast [24], i.e. for *n* requirements the total prioritization takes *n* comparisons, but the limitation is that it is not possible to calculate the consistency rate and say how important one requirement is against another, only that it is more important. The strength of the AHP is that it is possible to calculate the consistency rate and know exactly how important one requirement is against the other. The limitation of the method is that the number of pair-wise comparisons among the requirements grows exponentially with the increase in the number of requirements. For *n* requirements the total number of pair-wise comparisons is: $\frac{n(n-1)}{2}$.

The idea of Karlsson et al [24] is to first divide all requirements into three piles with PG and then the most important pile is taken and requirements within the pile are prioritized using AHP. The advantage of this method is: requirement engineer could pay attention to the most important requirements instead of those less important which saves time. The fewest number of comparisons with PGcAHP would be equal to PG, i.e. *n*. However this would indicate that there is at most one requirement that is very important. The other requirements are either important or not important. In that case the idea would be to redo the PG part, or to apply AHP on the middle pile of requirements. The above scenario with maximum one *very important requirement* is only theoretical, logically an application need more than one requirement to be able to function properly.

Now let's take a closer look at the worst and the best case of the method. The best case with the lowest number of comparisons is equal to not putting any requirements in the most important pile, i.e. the number of comparisons is equal to *n*. The worse case is that all the requirements have been placed into the most important pile. That would lead to the longest time to prioritize the requirements, which would be equal to $n+\frac{n(n-1)}{2}$, i.e. the number of comparisons for PG + the number of comparisons for AHP.

I-1. How to implement PGcAHP Method in Requirement Prioritization:

(1) As preparation, all candidate requirements are outlined.
(2) As execution, each requirement is taken and put in the appropriate pile. This process continues until all requirements are sorted. Then all requirements within the pile "those without which the system will not function" are put in a matrix and compared using the scale in Table 1 like AHP.
(3) As presentation, the 'averaging over normalized columns' method (based on the pair wise comparisons) is used to estimate the relative priority of each requirement and consistency ratio of the pair-wise comparisons is calculated using methods provided by AHP. The consistency indicates how trustworthy the results are and also how much judgment error that the analyst has done in the prioritization.

Mathematically it is more suitable to apply PGcAHP, iff, the amount of requirements is more than 21 and if we believe that the number of very important requirements falls within 80% or 90%. If the number of requirements is less than 80% it is better to use AHP. However, this is purely mathematical. In the real life, we do not know how important the requirements are before we prioritize them against each other. Hence, it is not possible to say if the 80% or 90% level is applicable to our problem(s).

IV. Evaluation Frameworks

A. Introduction

The objective is to evaluate the prioritizing methods presented in the previous section to find out which method takes the shortest amount of time in combination to yield the most accurate result and are able to scale up when more requirement are added, from the point of view of prioritizing personnel (personnel could be technical, economical and/or somebody that represent the customer, either by knowledge, behaviors, or any other way that could interesting for the successes for the project) who are going to prioritize the requirements. This section outlines the framework of the evaluation which has been carried out in the form of an experiment. The framework is highly influenced by the experimental approach outlined in [25].

B. Preperation

With the motivation of gaining a better understanding of requirement prioritization, we performed a single project study [25] with the aim of characterizing and evaluating the

http://sites.google.com/site/ijcsis/
ISSN 1947-5500

requirement prioritizing methods under observation from the perspective of users and project managers. This experiment was carried out by 10 persons consist of software developers, project managers, faculty members of computer science and persons without computer science background. They were asked to prioritize 14 features of a mobile set using the prioritization techniques under observation. 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 [3].

Only one method was studied each day to minimize the influence of the order of the methods, and to reduce the influence of the persons remembering the priorities of the requirements using the previous methods. Each day, 15 minutes were allocated for presenting the method which was under observation on that day and after getting the confirmation from each participant that the method 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.

C. Threats to Validity

When reading a result from an experiment, one of the most important questions is: How valid the result is? That makes validity of the result an important question to consider when an experiment is designed [26]. The overall objective of the experiment was evaluation of some requirement prioritization techniques by making some comparisons among them. We do not argue that the results obtained in this evaluation can be generalized and used by any user in any environment for any application. Rather, we tried to illustrate the requirement prioritizing methods to gain a better understanding of them. The following threats have been identified:

C-1. Few persons involved in the experiment:

The significance of the results is limited due to involvement of few persons (10 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.

C-2. 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 14 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 14.

C-3. Hypothesis guessing:

There was a risk that the participants might try to figure out what the intention and the purpose of the experiment were, and then they would answer to satisfy this intention and purpose. That would result in a misleading conclusion.

C-4. Requirements are interdependent:

In practice, the interdependence between the requirements must be considered. None of the prioritizing methods 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.

C-5. Only non functional requirements considered:

This experiment was only concerned with non functional requirements. However, we don't think it to be a major threat to the results from the experiment.

C-6. Offline evaluation:

The evaluation was carried out independently from a 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 potential methods for prioritizing software requirements.

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 [3].

D. Analysis of Collected Data

The testing begins with the first question of every method; followed by the second and third and so on. For each question, participants ranked each method and finally mean value was taken. An important notice is that all the numbers have been rounded to two significant digits.

D-1. Ease of Use:

The first question that the participants were asked was how easy they thought that the method was. The result of the question is shown in Figure 1.

Figure 1 clearly indicates that Planning Game (PG) is the easiest method and AHP is the toughest one.

TABLE IV. Comparison among the methods f

Evaluation Criteria	**AHP**	**Hierarchy AHP**	**Minimal Spanning Tree**	**Bubble Sort**	**Binary Search Tree**
Consistency (Yes / No)	Yes	Yes	No	No	Yes
Scale of Measurement	Ratio	Ratio	Ratio	Ordinal	Ordinal

http://sites.google.com/site/ijcsis/
ISSN 1947-5500

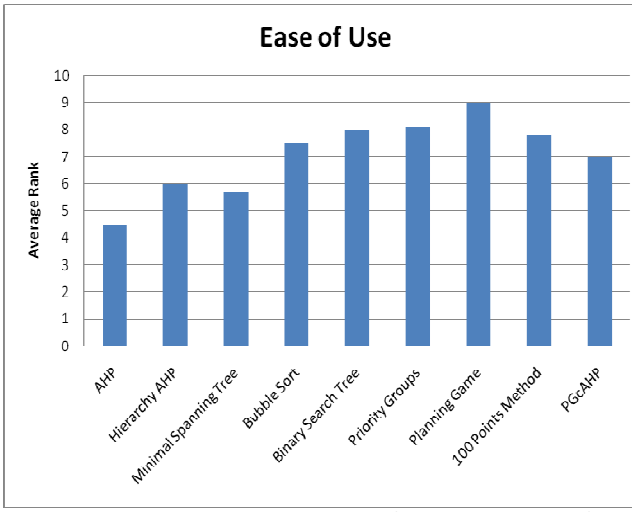

Figure 1: Comparison among the methods for the criteria "Ease of

D-2. Certainty:

The second question that the participants were asked was how certain they were about the end result obtained from the methods under consideration. The result is presented in the Table 4.

Though no statistical data was collected for this criterion, however, most participants think that AHP, Hierarchy AHP, Binary Search Tree, Planning Game (PG) and PGcAHP produce consistent result.

D-3. Total Time Taken:

Next question was how long time it took for the participants to perform the prioritization with the method under consideration. The result is presented in Figure 2.

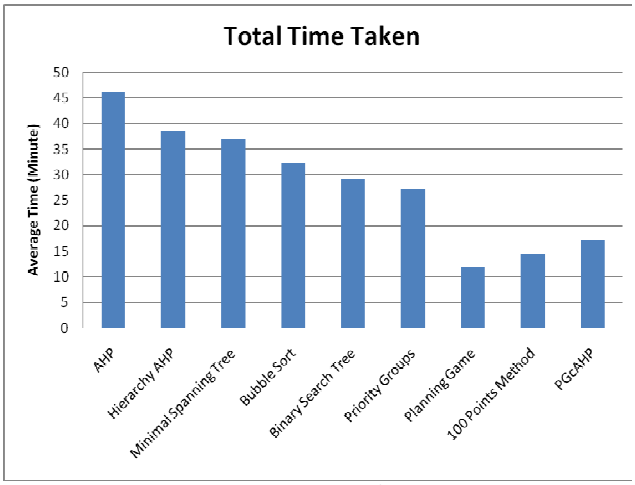

Figure 2: Comparison among the methods for the criteria "Total Time Taken"

The above graph clearly indicates that Planning Game (PG) is the fastest and AHP is the slowest among the prioritization techniques under consideration.

D-4.Scalability:

In next stage participants were asked to arrange the methods according to how they believe that the methods would work with many more requirements than the 14 considered in the experiment. The result is presented in Figure 3.

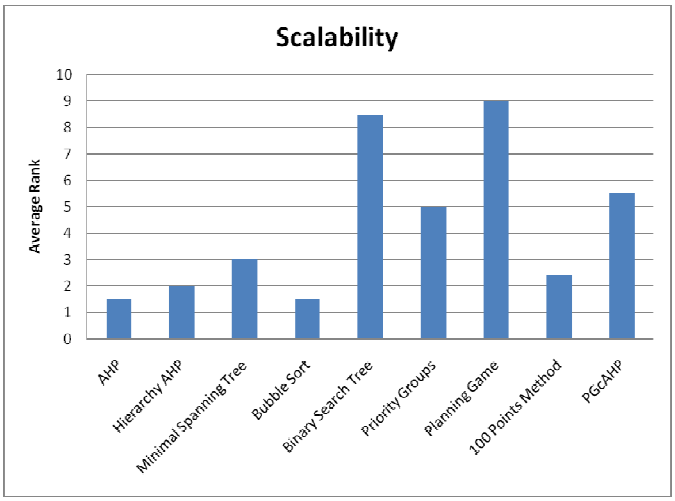

Figure 3: Comparison among the methods for the criteria "Scalability"

This graph indicates that most of the participant think that PG and BST are more scalable than other methods where as AHP, Hierarchy AHP and Bubble Sort are not suitable for large number of requirements.

D-5. Accuracy:

In this stage the participants were asked to arrange the methods under consideration according to their opinion about accuracy of the result produced by each method. However, there was a minor error that was overlooked in the experiment and it was that the words "accuracy" and "certainty" were used as meaning the same thing. Hence, final accuracy was compared with certainty in section *D-2*. The result is shown in figure 4.

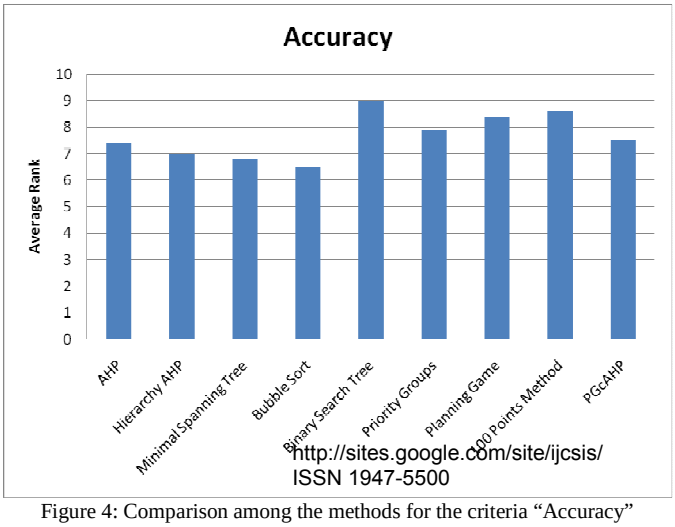

Figure 4: Comparison among the methods for the criteria "Accuracy"

http://sites.google.com/site/ijcsis/
ISSN 1947-5500

From figure 4 in can be imagined that BST and 100 Points Method yield the best result. 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 could be that the participants did not understand how to read out the matrix that presented the prioritization results.

D-6. Total Number of Comparisons:

The participants were asked to keep record of how many comparisons were required for each method. The result is shown in Figure 5.

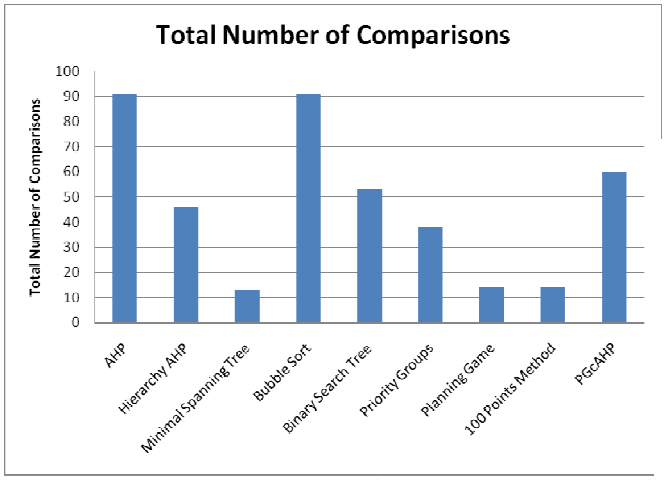

Figure 5: Comparison among the methods for the criteria "Total Number of Comparisons"

Figure 5 shows that Minimal Spanning Tree requires lowest number of comparisons where as AHP and Bubble Sort require highest number of comparisons.

V. FINDING THE BEST TECHNIQUE

After collecting data based on above motioned criteria, we assigned weight for each criterion and they applied a formula (8), (9) and (10) to find out the best technique for requirement prioritization. Each of the evaluation criteria was assigned weight according to Table 5.

TABLE V. Weight Table for Each Criterion

Name of the Criterion	Weight
Ease of Use	10
Certainty	3
Total Time Taken	9
Scalability	7
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.

$C_{ij} = W(C_i) * ((N + 1) – Rc_i(M_j))$ [Except C_2]
(8)

$C_{2j} = W(C_2) * ((N+1) * IsCertain(M_j))$
(9)

where IsCertain = 1 if M_j has Certainity else IsCertain = 0

$$OS(M_j) = \frac{\sum_{i=1}^{NC} C_{ij}}{NC}$$
(10)

Here,

N = Number of Techniques used
NC = Number of Criteria
C_{ij} = Score of Technique j in Criteria i
C_2 = Certainty
$W(C_i)$ = Weight of C_i
Rc_i (Mj) = Ranking of Technique j in Criteria i
$OS(M_j)$ = Overall Score of Technique j

The result after calculation is shown in Figure 6.

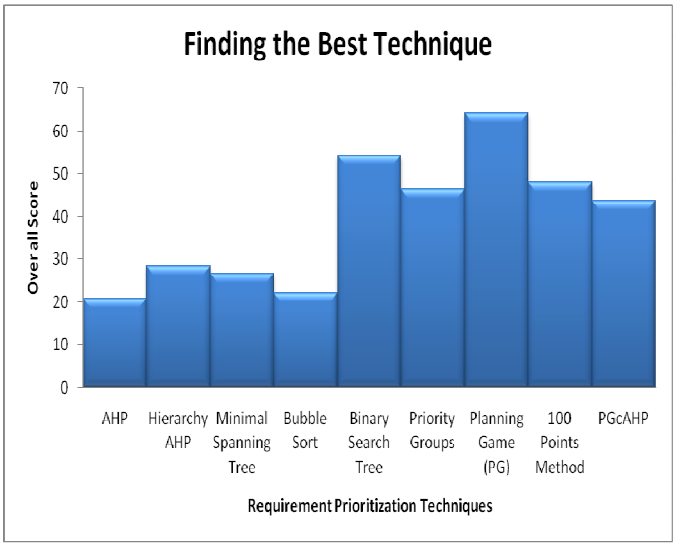

Figure 6: Comparison among the methods on the basis of weighted value

Figure 6 clearly indicates that among all the requirement prioritization techniques under consideration, Planning Game (PG) is supposed to be the best one on the basis of 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 methods can be used in any scenario with appropriate criterion weights suitable for that scenario.

http://sites.google.com/site/ijcsis/
ISSN 1947-5500

VI. DISCUSSION

Final outcome of the experiment says that Planning Game (PG) is supposed to be the best method for prioritizing software requirements. It is an easy method which produces one of the most accurate results and it is rather easy to handle even if there are many more requirements. The worst candidate according to the result is Analytical Hierarchy Process (AHP) as according to the participants it is the toughest method and it is difficult to scale up for a large number of requirements. Although the results have indicates that PG is the best and AHP is the worst among the candidate techniques of this experiment, there are some variables that may have some impact on the result.

(1) Most of the participants had previous experience and knowledge about the working mechanism of Planning Game (PG) technique. Hence, they found it comfortable while prioritizing requirements.

(2) One the other hand, some of the participant had no previous experience about some methods such as AHP, Hierarchy AHP and hence they may not find it convenient to apply these methods to prioritize requirement.

However, these are some minor issues having impact on the obtained result, and it would be interesting to evaluate these issues in various experiments in future.

VII. CONCLUSION

Methods for establishing priorities are of great importance in software development, since the developers' best effort can more easily be focused on the issues which matter most for the success of the system [3]. Therefore we have evaluated and characterized nine different methods for establishing priorities. In our evaluation we found PG to be the most promising approach as it yields one of the most trustworthy results by taking least time and it also works fine when the number of requirements increases.

Of course, the other methods considered in this experiment could be combined in many different ways to provide a more efficient approach for prioritizing requirements where the required number of comparisons could be reduced in an efficient manner.

Interestingly, after this experiment, we found that evaluating the requirement prioritization techniques in group sessions to be a means of communicating knowledge, achieving consensus and effective for identifying potential problems in the requirements. Using a group rather than individuals in this regard, helps the participants to bring out their particular knowledge on which they judge the requirements and that's why, as soon as the results were made visible to the participants, they immediately felt that the results reflected their judgments.

REFERENCES

[1] Sawyer P (2000) Packaged software: challenges for RE. Proc Int Workshop on Req Eng: Foundations of Software Quality. Stockholm, Sweden, pp 137–142

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

[3] Karlsson, J., Wohlin, C., & Regnell, B. (1998). An evaluation of methods for prioritizing software requirements. Information and Software Technology, 39(14-15), 939-947.

[4] Hatton, S. (2007). Early prioritisation of goals. In Advances in conceptual modeling – Foundations and applications (pp. 235-244).

[5] Ngo-The, A., & Ruhe, G. (2005). Decision support in requirements engineering. In A. Aurum & C. Wohlin (Eds.), Engineering and managing software requirements (pp. 267-286): Springer Berlin Heidelberg.

[6] H. In, D. Olson, and T. Rodgers. Multi-Criteria Preference Analysis for Systematic Requirements Negotiation. In 26th Annual International Computer Software and Applications Conference, Oxford, England, August 2002.

[7] J. Karlsson, Software requirements prioritizing, in: Proc. of 2nd IEEE Int. Conf. on Requirements Eng. (1996) pp. 110–116.

[8] F.Moisiadis. The fundamentals of prioritising requirements. In System Engineering, Test and Evaluation Conference, Sydney, Australia, 2002.

[9] A. Ngo-The and G. Ruhe. Requirements Negotiation under Incompleteness and Uncertainty. In Software Engineering Knowledge Engineering 2003 (SEKE 2003), San Francisco, CA, USA, July 2003.

[10] A. Sivzattian and B. Nuseibeh. Linking the selection of requirements to market value: A portfolio-based approach. In REFS 2001, 2001.

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

[12] M. Lubars, C. Potts, C. Richter, A review of the state of the practice in requirements modeling, in: Proc. of the IEEE Int. Symp. on Requirements Eng. (1993) pp. 2–14.

[13] T.L. Saaty, The Analytic Hierarchy Process, McGraw-Hill, Inc. (1980).

[14] G.R. Finnie, G.E. Wittig, D.I. Petkov, Prioritizing software development productivity factors using the analytic hierarchy process, J. Systems Software 22 (2) (1993) 129–139.

[15] C. Douligeris, I.J. Pereira, A telecommunications quality study using the analytic hierarchy process, IEEE J. Selected Areas Commun. 12(2) (1994) 241–250.

[16] J. Karlsson, Towards a strategy for software requirements selection. Licentiate thesis 513, Department of Computer and Information Science, Linko¨ping University, 1995.

[17] Paetsch F., Eberlein A., Maurer F., "Requirements Engineering and Agile Software Development", Proceedings of the 12 IEEE International workshop, IEEE Computer society, 2003, pp 1-6

[18] Regnell B., Höst M., Natt och Dag J., Beremark P., Hjelm T., "An Industrial Case Study on Distributed Prioritisation in Market-Driven Requirements Engineering for Packaged Software", Requirements Engineering, Feb 2001, vol. 6, number 1, pp 51-62

[19] Saaty T. L.,"That Is Not the Analytic Hierarchy Process: What the AHP Is and What It Is Not", Journal of Multi-Criteria Decision Analysis, 1997, vol. 6, No: 6, pp 324 – 335

[20] Gass S., Rapcsák T.," Singular value decomposition in AHP", European Journal of Operational Research, 2004, Vol. 2004, No. 3, pp 573-584

[21] A. Davis, Software Requirements: Objects, Functions and States. Prentice-Hall International, Englewood Cliffs, New Jersey, 1993.

[22] A.V. Aho, J.E. Hopcroft, J.D. Ullman, Data Structures and Algorithms. Addison-Wesley, Reading, MA, 1983.

http://sites.google.com/site/ijcsis/
ISSN 1947-5500

[23] Beck K., ”Extreme programming: explained”, Addison-Wesley, USA, December 2001, 7th edition.

[24] Karlsson L., Berander P., Regnell B., Wohlin C., ”Requirements Prioritisation: An Exhaustive Pair-Wise Comparison versus PG Partitioning”, EASE ’04- Empirical Assessment in Software Engineering, 2004

[25] V.R. Basili, R.W. Selby, D.H. Hutchens, Experimentation in software engineering, IEEE Trans. Soft. Eng. 12 (7) (1986) 733–743.

[26] Wohlin C., Runeson P., Höst M., Ohlsson M., Regnell B., Wesslén A., “Experimentation in software engineering: An introduction”, Kluwer Academic Publishers, USA, 2000

The Standish Group International Inc, “CHAOS Summary 2009: The 10 Laws of CHAOS”, found at http://www.statelibrary.state.pa.us/portal/server.pt/document/690719/chaos_summary_2009_pdf

AUTHORS’ PROFILE

Mohammad Shabbir Hasan received his B.Sc. (Engg.) in Computer Science and Engineering from Khulna University of Engineering and Technology (KUET), Bangladesh in 2008. His research interest includes different areas of Software Engineering like Requirement Engineering, Software Metric, Software Security and Software Maintenance. He has coauthored numerous research papers published in International Journals and Conference Proceedings. Currently he is working as a researcher of Panacea Research Lab, Dhaka, Bangladesh. He is also a lecturer of Department of Computer Science and Engineering in Institute of Science, Trade and Technology, Dhaka, Bangladesh.

Abdullah Al Mahmood received his B.Sc. (Engg.) in Computer Science and Engineering from Khulna University of Engineering and Technology (KUET), Bangladesh in 2009. His research interest includes Robotics, Artificial Intelligence, Internet Security and various areas of Software Engineering like Requirement Gathering, Requirement Prioritization and Software Security. He has coauthored a good number of research papers published in International Journals and Conference Proceedings. Currently he is working as a researcher of Panacea Research Lab, Dhaka, Bangladesh. He is also a lecturer at Institute of Science, Trade and Technology, Dhaka, Bangladesh and a Project Coordinator of Technocrats BD.

Md. Jahangir Alam received his B.Sc. (Engg.) and M.Sc. in Computer Science and Engineering from Dhaka International University (DIU), Bangladesh. His research interests include Software Engineering and Data Structure. Currently he is working as a Senior Lecturer of Department of Computer Science and Engineering, Dhaka International University, Dhaka, Bangladesh.

Sk. Md. Nahid Hasan received his B.Sc. (Engg.) in Computer Science and Engineering from Khulna University of Engineering and Technology (KUET), Bangladesh in 2010. His research interest includes different areas of Software Engineering like Software Metric, Requirement Gathering, Software Security and Software Maintenance. Digital Image Processing is also one of his research concerns. Currently he is working as a researcher of Panacea Research Lab, Dhaka, Bangladesh. He is also a lecturer of Department of Computer Science and Engineering in Institute of Science, Trade and Technology, Dhaka, Bangladesh.

Farin Rahman received her B. Sc. (Engg.) in Computer Science and Engineering from Khulna University of Engineering and Technology (KUET), Bangladesh in 2009. Her research interest encompasses with different sectors of Software Engineering like Requirement Engineering, Software Security and Software Maintenance. She is working as a researcher in Panacea Research Lab and also as a lecturer in Daffodil Institute of Information Technology, Dhaka, Bangladesh.

http://sites.google.com/site/ijcsis/
ISSN 1947-5500