Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 3, Issue. 4, April 2014, pg.1336 – 1341

RESEARCH ARTICLE

AN EVALUATION OF METHODS TO PRIORITIZE REQUIREMENTS

Prachi Bhandari[1], Rajni Sehgal[2]

[1]CSE, AMITY UNIVERSITY, NOIDA, INDIA

[2]CSE, AMITY UNIVERSITY, NOIDA, INDIA

[1] prachibhandari50@gmail.com, [2] rajnisehgal23@gmail.com

Abstract— This article describes an evaluation of six different methods for prioritizing software requirements. Based on the quality requirements for a telephony system, the authors individually used all six methods on separate occasions to prioritize the requirements. The methods were then characterized according to a number of criteria from a user's perspective. We found the analytic hierarchy process to be the most promising method, although it may be problematic to scale-up. In an industrial follow-up study we used the analytic hierarchy process to further investigate its applicability. We found that the process is demanding but worth the effort because of its ability to provide reliable results, promote knowledge transfer and create consensus among project members. q 1998 Elsevier Science B.V.

Keywords— Requirements; engineering; Requirements; prioritising; Experimental evaluation

1. Introduction

In commercial software systems development there is an increasing need for methods capable of prioritizing candidate requirements. Reasons for this include that not all requirements can usually be met with available time and resource constraints that the customers to a larger extent are demanding systems with the most bang for the buck, or that requirements must be allocated to different releases. Efficient and trustworthy methods for prioritizing requirements are therefore strongly demanded by practitioners. A promising framework for this purpose is the cost value approach [1]. In using this approach, decision makers are provided with guidelines on how to prioritize the requirements based on their relationships of value to cost of implementation.

This paper provides an investigation of six candidate methods for prioritizing requirements: analytic hierarchy process (AHP), hierarchy AHP, spanning tree matrix, bubble sort, binary search tree and priority groups. These methods can either be used as stand-alone utilities or be utilized within the cost value approach. To study these methods, we systematically applied all methods to prioritize 13 well-defined quality requirements on a small telephony system. We then categorized the methods from a user's perspective according to a number of criteria such as ease of use, required completion time and reliability of results.

Despite its problems of scaling-up, we found the analytic hierarchy process to be the most promising method for prioritizing requirements. To further verify this conclusion, we used the method in an industrial case-study. The practitioners were highly inspired by the strength of the method and confirmed its industrial applicability.

This paper is organized as follows. Section 2 motivates this work, and the paper continues in Section 3 by outlining the six different prioritizing methods. Section 4 describes the evaluation framework and Section 5 presents experience from the industrial application. Section 6 concludes the paper with a discussion and provides some recommendations.

2. Motivation

There is a growing acknowledgment in industrial soft-ware development 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 [2]. In a review of the state of the practice in requirements engineering, Lubars et al. [3] 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 communicate these priorities effectively to project members.

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. In industrial projects, this approach has been experienced as being effective, accurate and also to yield informative and trustworthy results [5]. Probably even more important, having used the approach in several commercial projects, we have experienced that practitioners are very attracted by the approach, and continue to use it in other projects. Despite such 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. Prioritizing methods

Prioritizing methods guide decision makers in their task of analyzing requirements in order to assign them numbers or symbols selecting their importance. 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.

(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.

3.1. The analytic hierarchy process (AHP)

The analytic hierarchy process (AHP) is a decision-making method [4]. Using AHP to prioritize software requirements involves comparing all unique pairs of requirements to determine which of the two is of higher priority, and to what extent. In a software project comprising n requirements, $n´(n ¹ 1)/2$ pair-wise comparisons are consequently required by a decision maker.

Prioritizing software requirements using AHP involves all three stages of a prioritizing session (for a comprehensive description of AHP, see Ref. [4]):

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

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

(3) As presentation, use the `averaging over normalized columns' method (based on the pair-wise comparisons) to estimate the relative priority of each requirement. Calculate the consistency ratio of the pair-wise comparisons using methods provided by AHP. 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.

3.2. Hierarchy AHP

In large-scale development projects the requirements are often structured in a hierarchy of interrelated requirements

Table 1
Fundamental scale used for pair-wise comparisons in AHP [4]

Intensity of importance	Description
1	Of 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*

3.3. 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).

If decision makers were perfectly consistent, the redundancy of the comparisons would be unnecessary. In such a case only n ¹ 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.

3.4 Bubblesort

Bubble sort is one of the simplest and most basic methods for sorting elements with respect to a criterion [10]. 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).

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

Using the bubble sort approach involves the following stages of a prioritizing session:

(1) As preparation, outline the requirements in a vector.

(2) As execution, start to compare the requirements at the top with the requirement at the position below the top. If the requirement in the above position is considered of higher priority, swap their positions. Continue the comparison until unique combinations of positions have been compared.

(3) As presentation, outline the sorted vector.

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 are requirements ranked according to their priority on an ordinal scale.

3.5. 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 labelled with elements of a set [10]. 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 in *order*, 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.

Prioritizing *n* software requirements using the binary search tree approach involves constructing a binary search tree consisting of *n* nodes. The ®rst 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.

4. Evaluation framework

4.1. Introduction

The objective is to evaluate the prioritizing methods presented in the previous section. 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 Ref. [11]. The evaluation criteria presented here are based on inherent characteristics, objective measures of the methods and subjective measures by the authors. Inherent characteristics are attributes of the method itself, objective measures are observed during the evaluation, and subjective measures are grades jointly assigned on an ordinal scale after the study by the authors.

4.2. Evaluation definition

With the motivation of gaining a better understanding of requirements prioritizing, we performed a single project study [11] with the aim of characterizing and evaluating the six candidate prioritizing methods from the perspective of users and project managers. The methods were studied by each of the authors by applying them to the 13 quality requirements [13] outlined in Fig. 1 for a small telephony system.

To carefully and systematically evaluate and characterize the prioritizing methods the authors initiated a minor experiment. This experiment was carried out by a single team (the authors). The overall objectives of the experiment are two-fold: (a) to illustrate the prioritizing methods; (b) to evaluate and characterize the prioritizing methods.

4.2.1 Execution

The experiment was divided into two phases:

(1) In phase 1 the requirements were prioritized by the three persons 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. The methods were drawn in random order for each of the three persons, which meant that one person may use method A first and then method B, and a second person may start with method C and then method B. Phase 1 generated the objective measures.

(2) In phase 2, the authors sat down jointly and discussed the results of the evaluation and the methods were judged subjectively. Each person ordered the methods according to the subjective measures, and based on the individual ordering, a joint order was agreed on. The individual orders given by the different persons were mostly very close to each other, and hence it was very easy to come to agreement on the ranking of the methods. Thus, phase 2 provided the subjective measures.

To minimize the risk of the persons remembering their own ordering in phase 1, only one method was studied each week. This was done to further 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.

In phase 1, the methods were evaluated in the order out-lined in Table 2. For example, this implied that person A

Table 2
Method evaluation order

Method	Person A	Person B	Person C
AHP	6	4	1
Hierarchy AHP	4	3	6
Minimal spanning tree	1	2	2
Bubble sort	3	5	5
Binary search tree	5	6	4
Priority groups	2	1	3

used *minimal spanning tree* in the ®rst week, and *priority groups* in week 2. After 6 weeks all six methods had been used by the three persons, and they then met to perform phase 2, and to analyse the data collected.

Table 3
Inherent characteristics of the prioritizing methods

Evaluation criteria	AHP	Hierarchy AHP	Spanning tree	Bubble sort	Binary search	Priority groups
Consistency index (yes/no)	Yes	Yes	No	No	No	No
Scale of measurement	Ratio	Ratio	Ratio	Ordinal	Ordinal	Ordinal

Table 4
Objective measures during the evaluations

Evaluation criteria	AHP	Hierarchy AHP	Spanning tree	Bubble sort	Binary search	Priority groups
Required number of decisions	78	26	12	78	29,33,38	34,35,36
Total time consumption (ordinal scale 1±6)	6	2	1	3	5	4
Time consumption per decision (ordinal scale 1±6)	2	4	5	1	6	3

Table 5
Subjective measures after the evaluations

Evaluation criteria	AHP	Hierarchy AHP	Spanning tree	Bubblesort	Binary search	Priority groups
Ease of use	3	4	2	1	5	6
Reliability of results	1	3	6	2	4	5
Fault tolerance	1	3	6	2	4	5

4.6. Evaluation analysis

Table 3 shows that the first three methods provide a more powerful scale. Methods based on AHP also allow the possibility of checking the consistency of the priorities. The conclusion from the inherent properties is thus that AHP and hierarchy AHP are most powerful.

The objective measures in Table 4 show that AHP and bubble sort require the highest number of decisions (around 80) and spanning tree requires the fewest (around 10). The number of decisions required for binary search and priority groups depend on the decisions taken during method execution, hence the three different values for each evaluator. Binary search, priority groups and hierarchy AHP all required around 30 decisions.

The total time consumption and the time consumption per decision are presented on an ordinal scale, as here we are only interested in the ranking of the methods. The results from the evaluation showed that AHP and binary search need the longest time to execute, while hierarchy AHP and spanning tree were the fastest methods. If we divide the total time by the number of decisions, we see that binary search and spanning tree required most time per decision, while AHP and bubble sort were, on average, fastest per decision.

Our subjective evaluation, as reported in Table 5, resulted in good marks for AHP and bubble sort, with respect to ease of use, reliability of results and fault tolerance.

5. Industrial application

The results from our evaluation of prioritizing methods indicate that AHP is both useful and trustworthy but may be problematic to scale up. In order to gain more understanding about the industrial applicability of AHP and how to make the best use of it, we applied the method to an early phase of the development of industrial software system. Since a cross-functional team was involved in the project, we also had the opportunity to investigate the effects of letting the team perform the prioritizing together, rather than letting the developers carry out the prioritizing individually. Prior to the prioritizing session, the cross-functional team gathered and brainstormed all options to be prioritized. After analysing and reviewing the options, the team settled for 11 options. Since time and resources did not allow full utilization of all options, the prioritizing session was considered of high importance.

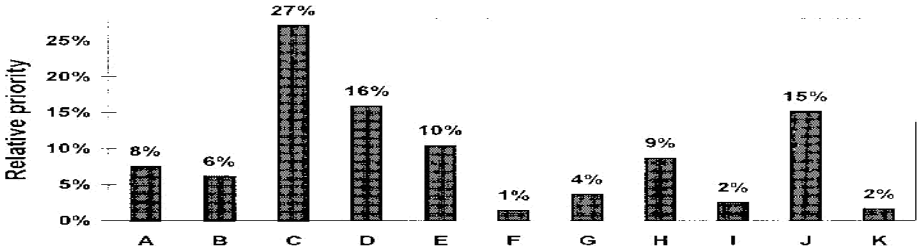

Fig. 2. Relative priority of the 11 options.

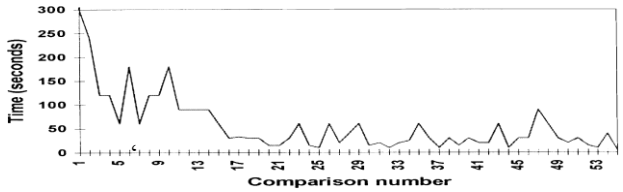

Fig. 3. Required time for pair-wise comparisons

Such diagrams provide a graphical illustration on which developers can reason, for example, about how to best utilize the scarce resources in software development. In this project, we divided the options into logical categories according to their priorities. Relatively, options C, D and J are of high priority, options A, B, E and H of medium priority, and options F, G, I

and K of low priority.

On the average one pair-wise comparison took the cross-functional team about 1 min to perform. The time distribution of the pair-wise comparisons is outlined in Fig. 3.

In using AHP and the resulting diagram, the notion of consensus becomes very tangible. When reviewing the diagram all participants directly agreed that the result mirrored their exact views based on the knowledge gained from the discussion. Interestingly, no participant believed he would have reached the same quality in the result by using any other method known to them. More-over, making decisions based on consensus is generally advantageous since the often tedious discussions on details are left out

Conclusions

The research which is formulated in this paper develops an innovative approach so as to evaluate the requirement's quality in software projects which are basically based on multiple quality evaluation method. It also presents a method that will be using desirability functions for creating a unified measure that represents meets quality attributes and then the importance of the quality attributes for the application. We can now further enhance this application to work on private cloud environment where several users are furnishing their requirements from different locations. We may also include the principles of combining software engineering techniques like the estimating different software metrics like Effort, Time, People, and Cost etc

ACKNOWLEDGMENT

We would like to thank Dr. Abhay Bansal, Head of Department (CSE) and Mr. Abhishek Singhal (Programme Leader) at Amity University for their constant guidance and support in our research project.

REFERENCES

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

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

[3]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 Require-ments Eng. (1993) pp. 2±14.

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

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

[6]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.

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

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

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

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

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

[12] S.S. Stevens, On the theory of scales of measurement, Science 103 (2684) (1946) 677±680.

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

