J. Karlsson, C. Wohlin and B. Regnell, "An Evaluation of Methods for Prioritizing Software Requirements", Information and Software Technology, Vol. 39, No. 14-15, pp. 939-947, 1997-98.

ELSEVIER

Information and Software Technology 39 (1998) 939–947

INFORMATION AND SOFTWARE TECHNOLOGY

An evaluation of methods for prioritizing software requirements

Joachim Karlsson[a,b,*], Claes Wohlin[b], Björn Regnell[c]

[a]Focal Point AB, Teknikringen 1E, SE-583 30 Linköping, Sweden
[b]Department of Computer and Information Science, Linköping University, SE-581 83 Linköping, Sweden
[c]Department of Communication Systems, Lund University, SE-221 00 Lund, Sweden

Received 7 February 1997; revised 5 November 1997; accepted 13 November 1997

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. © 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 software 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 how to

* Corresponding author. Tel.: +46 13 213705; fax: +46 13 213725; e-mail: Joachim.Karlsson@focalpoint.se

0950-5849/98/$19.00 © 1998 Elsevier Science B.V. All rights reserved
PII S0950-5849(97)00053-0

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.

To our knowledge, AHP has only been used in few applications in the software industry. Finnie et al. [6], for example, used AHP to prioritize software development factors. They found AHP to be relatively easy to apply, but do not elaborate on the problems arising if the number of prioritizing objects grows. Other applications of AHP include a telecommunications quality study performed by Douligeris and Pereira [7], and software requirements prioritizing in a commercial development project by Karlsson [8]. In these cases the number of prioritizing objects was very low and the required number of pair-wise comparisons did not consequently cause unmanageable problems in the process.

Since AHP may be problematic for large-scale projects, we have identified five complementary approaches to challenge AHP. All of these method 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 [5]. We have focused on methods which may reduce the required effort, but are still able to produce high-quality results which are considered trustworthy by its users.

3. Prioritizing methods

Prioritizing methods guide decision makers in their task of analyzing requirements in order to assign them numbers or symbols reflecting 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, $n \cdot (n-1)/2$ pair-wise comparisons are consequently required by a decision maker. On the one hand AHP is a demanding method due to the dramatically increasing number of required pair-wise comparisons when the number of requirements grows. On the other hand AHP is very trustworthy since the huge amount of redundancy in the pair-wise comparisons makes the process fairly insensitive to judgmental errors. Another advantage is that the resulting priorities are relative and based on a ratio scale, which allows for useful assessments of requirements.

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

[9]. 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, we use the name *hierarchy* AHP in this paper.

Using hierarchy AHP also involve all three stages of a prioritizing session:

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

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

(3) As presentation, do the same as for AHP 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.

3.3. 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 $n - 1$ 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.

Using the minimal spanning tree approach involves all three stages of a prioritizing session:

(1) As preparation, outline $n - 1$ unique pairs of requirements so that a minimal spanning tree can be constructed.

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

(3) As presentation, compute the missing intensities of importance by taking the geometric mean of the existing intensities of all possible ways in which they are connected. Then use AHP as usual.

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

3.4. Bubblesort

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

Using the bubblesort 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 bubblesort 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 labeled 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 subtree of the node x are all of lower priority than the requirement stored at x, and all requirements stored in the right subtree of x are of higher priority than the requirement stored in x. If the nodes in a binary search tree are traversed in *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 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.

Using the binary search tree approach involves all three stages of a prioritizing session:

(1) As preparation, outline the candidate requirements.

(2) As execution, select the requirements one at a time and create a binary search tree.

(3) As presentation, traverse the binary search tree in inorder and add them to a list. The requirements having the lowest priority then come first in the list. Print the list.

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.

3.6. 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.

The primary gain is that we do not have to compare high priority requirements with low priority requirements, since they are placed in different groups. The actual choice of the number of groups depends on the situation and the knowledge of the people performing the prioritization. A simple strategy would be to use 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.

Using the priority groups approach involves all three stages of a prioritizing session. Assume three distinct groups are used:

(1) As preparation, outline the candidate requirements.

(2) As execution, put each of the requirements into one of the three groups. In groups with more than one requirement, create three new subgroups and put the requirements into these groups. Continue to apply this process recursively to all groups.

(3) As presentation, just read the requirements from left to right.

A potential additional action, to ensure that the correct ordering of the requirements is obtained, is to compare the lowest ranked requirement in one group with the highest ranked requirement in the next group. This could be done to ensure that the tail of one group should have 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.

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 twofold: (a) to illustrate the prioritizing methods; (b) to evaluate and characterize the prioritizing methods.

4.3. Evaluation criteria

4.3.1. Inherent characteristics

Two inherent characteristics of the prioritizing methods were identified:

(1) *Consistency indication*: this characteristic indicates whether the prioritizing method is able to indicate consistency in the decision maker's judgment. This ability requires redundancy in the judgments.

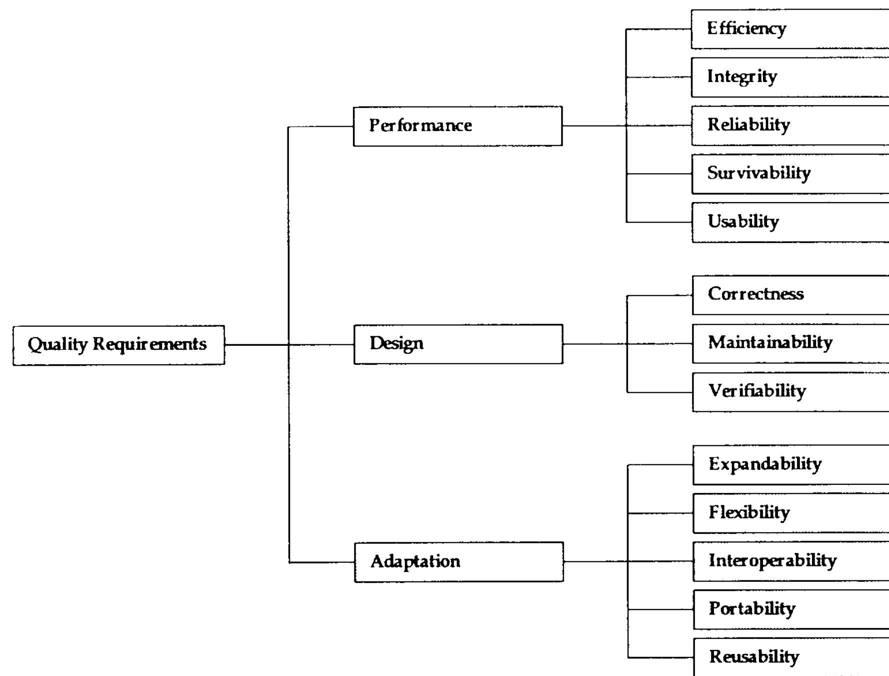

Fig. 1. Quality requirements subject to prioritizing [13].

(2) *Scale of measurement*: this characteristic describes the scale on which the resulting requirements priorities are based. The scale used for ranking the requirements is an important attribute of goodness. The more powerful the scale, the more useful the assessments of the requirements can be carried out. These measurement scales in increasing order of strength are: nominal, ordinal, interval, and ratio scales [12].

4.3.2. Objective measures

The following objective measures were assessed during the evaluation sessions:

(1) *Required number of decisions*: for the first four methods the number of decisions is pre-defined, but for the two last methods the number is based on how the specific session was carried out. This measure refers to the number of pairwise comparisons required by a decision maker to complete all stages of the method.

(2) *Total time consumption*: this is an objective measure of the average time required by a decision maker to complete all stages of the method. This measure is different from the measure of required number of decisions, since comparisons are of a different nature and can thus take different times to accomplish. When it comes to the cost of using a prioritizing method, this measure can serve as a rough estimation.

(3) *Time consumption per decision*: this is an objective measure of the time consumption required per decision.

4.3.3. Subjective measures

After the evaluation, it was decided that it was essential to judge some aspects of the prioritizing methods based on the experience gained by using the methods. Initially, the objective was to grade the methods on an ordinal scale from 1 to 5, but after experiencing some difficulties it was determined to rank the methods from 1 to 6, where 1 is the best method for that particular criterion. The following features were judged by the authors after the evaluation:

(1) *Ease of use*: this measure describes how easy it is to use the prioritizing method.

(2) *Reliability of results*: this measure describes how reliable the results are judged to be.

(3) *Fault tolerance*: this measure describes how insensitive the method is to judgmental errors.

4.4. Evaluation operation

4.4.1. Preparation

The preparation, in terms of learning the methods for prioritization, was performed as follows. One of the authors, knowledgeable in all the methods, informed and explained the methods to the other two authors in a similar way to how we could expect the methods to be explained for potential users in practice. In total about 4 h were needed to become familiar with the methods in order to use them for prioritizing requirements.

The experiment was carried out by the three authors, using the quality requirements proposed by Keller et al. [13], which are depicted in Fig. 1. The requirements are prioritized for a telephony system. The system is a private branch exchange (PABX) for a small company with about 50 employees. The company is a consultancy firm in software and computer engineering, and is currently about to purchase a PABX. Thus, the company is keen to evaluate

different available PABX systems based on their quality characteristics.

The PABX will offer ordinary telephony, together with facsimile and computer communication. Additional operator services are required for managing local numbers and secretarial functions. Common services, such as call forwarding and conferencing, are also required. The system must be easy to extend with more subscribers and new services. The PABX will be used for communication within the company and with the customers. The system is thus a vital tool for the company's ability to perform its aims.

We will not go into further detail about the configuration of the PABX, and assume that the reader is familiar with the usage of and demands on such a system.

4.4.2. 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 outlined in Table 2. For example, this implied that person A used *minimal spanning tree* in the first 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 analyze the data collected.

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
Bubblesort	3	5	5
Binary search tree	5	6	4
Priority groups	2	1	3

4.5. Threats to validity

The overall objective of the evaluation was to illustrate the methods and to make a first comparison and evaluation. We will 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 want to illustrate the prioritizing methods and to gain a better understanding of them. In addition, choosing a prioritizing method must be based on goals and application area. This evaluation does, however, provide useful insights into advantages and disadvantages of the different methods.

The following threats have been identified:

4.5.1. Requirements are interdependent

In practice, the interdependence between the requirements must be taken into account. In particular, the tradeoffs between different requirements must be dealt with. None of the prioritizing methods described in this article provides means for handling interdependence, hence this limitation of the experiment is not believed to influence the actual evaluation of the different methods.

4.5.2. Few persons involved in the experiment

The significance of the results is limited since only three persons have been involved in the experimentation. It is, however, not regarded to be overly critical as the authors were very much in agreement when it came to the subjective measures. The results from the objective measures are more inconclusive, and hence can be regarded as a partial threat to the evaluation.

4.5.3. Only quality requirements considered

The study presented is only concerned with quality requirements, hence focusing on non-functional requirements. This limitation is, however, not believed to be a major threat to the results from the experiment.

4.5.4. Off-line evaluation

The evaluation was carried out independently from a software project. This is always considered a potential problem in experimentation. It is, however, not regarded as being a major threat as the main objective is to perform a first experiment to gain understanding and illustrate a number of potential methods for prioritizing 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.

Table 3
Inherent characteristics of the prioritizing methods

Evaluation criteria	AHP	Hierarchy AHP	Spanning tree	Bubblesort	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	Bubblesort	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 bubblesort 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 bubblesort were, on average, fastest per decision.

Our subjective evaluation, as reported in Table 5, resulted in good marks for AHP and bubblesort, with respect to ease of use, reliability of results and fault tolerance. Priority groups were given the lowest ranking.

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 analyzing and reviewing the options, the team finally settled for 11 options. Since time and resources did not allow full utilization of all options, the prioritizing session was considered of high importance. The prioritizing session of 55 pair-wise comparisons required a total effort of 52 min for the cross-functional team. The resulting priority distribution of the options is shown in Fig. 2. Notice the useful feature of AHP that the priorities always add up to 100%.

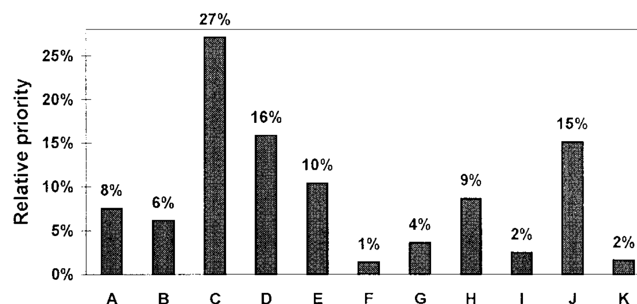

Fig. 2. Relative priority of the 11 options.

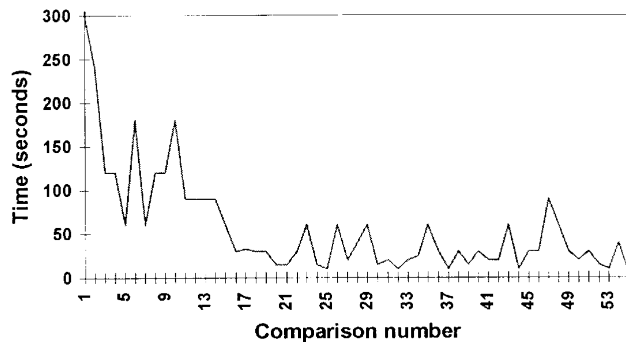

Fig. 3. Required time for pair-wise comparisons.

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.

The diagram in Fig. 3 indicates that the execution stage of a prioritizing session goes through two separate parts. The first part, from the first comparison to about comparison number 15, is a *knowledge transfer* part. During this part the participants have thorough discussions about each option and gain a deeper understanding of each of them. They also acquaint themselves with the notion of pair-wise comparisons. We experienced the knowledge transfer between the members of the cross-functional team as being very obvious. An explanation for this can be that knowledge transfer is more likely to be effective when the participants know they are going to need the knowledge involved [14]. This was certainly the case in this project. When the cross-functional team have reached a reasonably even level of knowledge about the options, the session enters a *fulfilment* part. During this part the required time for the pair-wise comparisons decreases dramatically due of the knowledge transfer in the previous part. The average time for a pair-wise comparison was reduced by one half during this stage.

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. Moreover, making decisions based on consensus is generally advantageous since the often tedious discussions on details are left out.

6. Discussion and recommendation

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. Therefore we have evaluated and characterized six different methods to establish priorities. In our evaluation as well as in a practical evaluation we have found AHP to be the most promising approach. It yields the most trustworthy results which are based on a ratio scale, it is fault-tolerant and it includes a consistency check. Other promising approaches, such as bubblesort, lack these important attributes. In using AHP, the priority distance between requirements becomes very tangible, the other methods only providing the correct order. The consistency check is very important since human judgment is far from perfect. The absence of a consistency index may make the process unreliable since nobody can assess the potential judgmental errors being made. On the other hand AHP may be problematic to scale up for larger projects (the same is true for bubblesort). Therefore, tools supporting the process as outlined in Ref. [15] are needed. In that process the required number of pair-wise comparisons in AHP can be reduced to more manageable numbers.

Of course, the methods could be combined in many different ways to provide a more efficient approach to prioritizing requirements. For example, the requirements could be grouped into three different groups (cf. the priority groups method) and then AHP could be applied to the three groups. In such an approach the required number of comparisons could be reduced in an efficient manner.

Using prioritization in industrial projects, we have found the idea very important and useful for better utilizing scarce resources. Interestingly, we have also found that establishing priorities in group sessions to be a means for both communicating knowledge, achieving consensus and for identifying potential problems in the requirements. Using a group rather than individuals to prioritize forces the participants to bring out their particular knowledge on which they judge the requirements. Consequently, the other participants learn as the process proceeds. Moreover, as the results were made visible to the group, they immediately felt that the results reflected their judgments.

By performing the pair-wise comparisons, misjudged, incorrect and ambiguous requirements were identified by the participants. We believe the idea of pair-wise comparisons forces the participants to analyze the requirements from a view not being used in traditional reviews, and thus finding complementary problems. Such positive side effects of prioritizing sessions were highly appreciated by the practitioners.

Acknowledgements

Ivan Rankin, Kevin Ryan and Kristian Sandahl provided valuable comments on a draft of this paper.

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 Requirements 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, Linköping 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.
[14] D.A. Garvin, Building a learning organization, Harvard Business Review, July–August, 1993, pp. 78–91.
[15] J. Karlsson, S. Olsson, K. Ryan, Improved practical support for large-scale requirements prioritizing, Requirements Eng. J. 2 (1) (1997) 51–60.