Requirements Eng (1997) 2: 51–60

Requirements Engineering

Improved Practical Support for Large-scale Requirements Prioritising

Joachim Karlsson[a,1], Stefan Olsson[a] and Kevin Ryan[b]
[a]Focal Point AB, Teknikringen 1E, Linköping, Sweden; [b]College of Informatics and Electronics, University of Limerick, Limerick, Ireland

An efficient, accurate and practical process for prioritising requirements is of great importance in commercial software developments. This article improves an existing cost–value approach in which stakeholders compare all unique pairs of candidate requirements according to their value and their cost of implementation. Techniques for reducing the required number of comparisons are suggested, thus making the process more efficient. An initial approach for managing requirements interdependencies is proposed. A support tool for the improved process has been developed to make the process more practical in commercial developments. The improved process and its support tool have been applied and evaluated in an industrial project at Ericsson Radio Systems AB. The results indicate a pressing need for mature processes for prioritising requirements, and the work presented here is an important step in that direction.

Keywords: Requirements engineering; Requirements prioritising; Cost–value approach

1. Introduction

A primary objective of requirements engineering is to permit the development of software systems that satisfy all of their stakeholders. Not only must the requirements engineers identify the stakeholders and their requirements, but they must also manage conflicting preferences and expectations. This is important since it is frequently the case that not all stakeholder requirements can be fully implemented, especially when expectations conflict or when resources are limited. In addition, requirements engineers will frequently have to distinguish between those requirements which will have a major impact on stakeholder satisfaction and those which will not. This is crucially important since the value of candidate requirements, as well as the cost of meeting those requirements, have been shown to vary by orders of magnitude [1]. To deal with this aspect we need an effective and practical process for prioritising software requirements.

This article justifies, describes and evaluates an effective process for prioritising software requirements. In previous work [2], a promising approach was identified which employs a pair-wise comparison method to establish requirements priorities in two dimensions: value to the stakeholders, and cost of implementation. This cost–value approach has its roots in the Analytic Hierarchy Process (AHP), which is an analytical tool useful for multi-criteria decision making [3]. When evaluated in industry the basic the cost–value approach was liked by practitioners but some practical drawbacks were identified which made it problematic for large-scale use. In addition there was a clear need for tool support to ease the clerical and computational burden on the practitioner.

In response, we have revised and improved the process and provided a tool to support it. For example, we have incorporated techniques to reduce the required number of pair-wise comparisons, thus extending the usefulness of the process to industrial software projects with many requirements. We have developed an initial approach for coping with requirements interdependencies, where stakeholders can observe how the selection of one requirement has an impact on

Correspondence and Offprint requests to: Joachim Karlsson, Focal Point AB, Teknikringen 1E, SE-583 30 Linköping, Sweden. Tel: +46 13 21 37 05; fax: +46 13 21 37 25; email: Joachim.Karlsson@focalpoint.se
[1]Also at Department of Computer and Information Science, Linköping University, Sweden

the value and cost of implementing another. These, and other improvements, have been implemented in a support tool which reduces much of the burden on the practitioners by, for instance, automating calculations, plotting cost–value diagrams and indicating consistency errors. Finally, we have evaluated the improved process and its supporting tool in a commercial telecommunications software development project. The overall results were very positive, although further improvements are possible.

Section 2 outlines the background and our research approach. Section 3 discusses requirements prioritising, outlines the cost–value approach and details difficulties with the basic approach. Section 4 outlines the process improvements which include reducing the number of comparisons, managing requirements interdependencies and allowing hierarchical representation of requirements. Section 5 presents the support tool for the improved process, and section 6 recounts the results and observations from the industrial application and evaluation. Finally, section 7 concludes the paper and suggests further steps.

2. Background

Since 1992, Ericsson Radio Systems AB and the Department of Computer and Information Science at Linköping University have been involved in a joint research program with the aim of improving the software engineering process, especially the early phases of the software life-cycle. In this collaboration the researchers participated in Ericsson's commercial development projects, with the aim of identifying immature activities, suggesting improvement opportunities, evaluating them in practice, and studying the outcomes. This collaboration has given us the opportunity to use the industry-as-laboratory approach [4]. We do not primarily perform academic experiments but instead participate as researchers in commercial projects using an approach termed *action research*. This is a form of self-reflective inquiry undertaken by a researcher, in an organisational context, so as to improve his or her own practices or understanding of these practices [5]. Two important characteristics of action research can be identified [6]. Firstly, the researcher seeks to gain more knowledge, but is also concerned to apply this knowledge. Secondly, the problem to be solved is defined by the practitioners, not by the researcher.

Performing such studies in industrial projects involves both a challenge and a risk. It is a challenge since long-term industrial studies are necessary and important in requirements engineering [7]. But it is also a risk since the commercial objectives of the projects must take precedence and therefore projects can change direction drastically, be moved to other geographical locations or be cancelled at short notice, any of which can be very difficult for the researcher.

Our action research is to a large extent motivated by the comprehensive field study carried out by Lubars et al. [8]. Their study shows that many organisations believe it is important to assign priorities to requirements and make decisions about them according to rational, quantitative data. Despite this, no company really knows how to assign priorities or how to communicate those priorities effectively to project members. It is all the more surprising therefore that there also seems to be a lack of research in the area of requirements prioritising [9].

To address the lack of industrially useful prioritising methods, we first developed and applied a numeral assignment method to prioritise requirements, but the method was neither accurate nor trustworthy enough. In a comparative study we contrasted this approach with a pair-wise comparison method based on the AHP. We found that the latter approach yields more accurate and more trustworthy results, and is also faster [10]. After subsequent discussions with management we also realised the need for trading off value and cost when prioritising requirements. A requirement should be given priority mainly guided by its value in relation to its cost of implementation. The cost–value approach for prioritising requirements was therefore developed.

3. Requirements Prioritising

In commercial software development there are usually few requirements that are considered absolutely mandatory and, for our purposes, these will be treated as part of the given implementation context. All of the others are negotiable both as to whether they should be implemented or not, and as to the degree to which they should be met. Moreover, in commercial projects, the desirability of requirements is strongly linked to their cost of implementation. Requirements estimated to be much more expensive than initially expected may lose their attraction to the stakeholders. On the other hand, requirements estimated to be cheaper than initially expected may, as a result, be more attractive to the stakeholders.

3.1. The Cost–Value Approach

In previous work the cost–value approach for prioritising software requirements was developed, applied in commercial projects and later evaluated [2]. When using this approach the stakeholders quantify the value and cost of candidate requirements before, for instance, selecting requirements for implementation or planning what to include in later releases. A requirement's value is defined as its ability to contribute to customer satisfaction with the overall system, when successfully implemented. A requirement's cost is an estimate of the additional cost required to meet that requirement alone. By relating each requirement's value to its cost, stakeholders have a measure of that requirement's ability to contribute to customer satisfaction.

The cost–value approach uses techniques from the AHP, for the quantification of value and cost of implementation. In the AHP, stakeholders compare pairs of candidate requirements, according to the two criteria value and cost of implementation, using the fundamental scale in Table 1. Based on the pair-wise comparisons, the value and cost distributions are calculated using techniques provided by the AHP. If stakeholders were perfectly able to judge each requirement's relative cost and value, the resulting distributions would be perfectly consistent. In practice, however, errors of judgement do occur, but the redundancy of pair-wise comparisons is very useful since it makes the AHP less sensitive to these errors. In addition, the AHP provides techniques to measure inconsistency by calculating consistency indices and consistency ratios.

Another important advantage of the AHP is the fact that the scale used for measuring the requirements priorities is a *ratio* scale. The four most common measurement scales are, in increasing order of strength: nominal, ordinal, interval, and ratio scale [11]. In the nominal scale, numerals are used as labels, where words or letters would serve as well. A typical example is the numbering of soccer players for the identification of the participants. The ordinal scale is used for rank-ordering purposes, such as assigning students grades on a scale ranging from 1 to 5, where 5 is best. The difference between a 5 and a 4, however, is not necessarily the same as the difference between a 4 and a 3, so the ordinal scale is not invariant when it comes to equality of differences. In the interval scale, however, there is an equality of differences. This can be exemplified by the Centigrade scale, where the difference between 20°C and 21°C is exactly the same as the difference between 29°C and 30°C. The strongest measurement scale is the ratio scale, which not only satisfies equality of intervals, but also equality of ratios. That is, if the distance between two cities is measured to be 10 miles, this distance is exactly twice the distance of 5 miles. It does not, for instance, make sense to claim that 20°C is twice as warm as 10°C. Since requirements priorities using the AHP are based on a ratio scale, interesting and useful assessments of requirements' priorities can be made.

Table 1. Fundamental scale for pair-wise companions [3]

Relation intensity	Description
1	Of equal value (of equal cost of implementation)
3	Moderate value (cost) difference
5	Essential or strong value (cost) difference
7	Very strong value (cost) difference
9	Extreme value (cost) difference
2, 4, 6, 8	Compromise numbers between the two adjacent judgements
Reciprocals	If requirement i has one of the above numbers assigned to it when compared with requirement j, then j has the reciprocal value when compared with i

The cost–value approach for prioritising requirements is carried out in five consecutive steps, given in detail in [2].

1. The requirements engineers carefully review the candidate requirements for completeness and to ensure that the requirements are stated as clearly as possible.
2. A representative set of customers and users apply the pair-wise comparison method in the AHP to assess the relative value of the full set of candidate requirements. If the actual or potential customers and users are not available, experienced software analysts or marketing personnel use the AHP to estimate the requirements' values.
3. Experienced software engineers use the pair-wise comparison method in the AHP to estimate the relative cost of implementation of each member of the set of candidate requirements.
4. A software engineer uses the AHP to calculate each candidate requirement's relative value and cost of implementation, and plots these on a cost–value diagram, where value is depicted on the y-axis and cost on the x-axis.
5. The different stakeholders use the cost–value diagram as a conceptual map for analysing and discussing the candidate requirements. This map provides a

useful focal point for the stakeholders. Based on this discussion software managers can effectively prioritise the requirements and, for instance, decide which requirements to select for actual implementation, or develop strategies for release planning.

It should be noted that currently the assessments of value and cost of implementation are based on the decision makers' experience and judgement, but could of course be supplemented by any other method.

3.2. Difficulties with the Current Process

Although practitioners were enthusiastic about the cost–value approach and viewed it as very useful in the requirements engineering process, it was clear that before it could be widely adopted some practical problems needed to be addressed:

- *Explosive growth in the number of comparisons.* For n requirements in a software development project, AHP requires $n(n-1)/2$ pair-wise comparisons (since all requirements are compared to all others). For the approach to effectively scale up, techniques for reducing the required number of pair-wise comparisons must be found. To illustrate the problem, a project involving 200 requirements would require a stakeholder to perform $200(200-1)/2 = 19{,}900$ pair-wise comparisons. At the rate observed in practice, which is about one to four comparisons a minute, this would take a number of — very boring — weeks. Besides, experience shows that later comparisons can be affected by fatigue.
- *Lack of support for requirements interdependencies.* In most projects requirements are linked to one another in various ways which can affect the actual selection process. For example in our context, implementing one requirement may affect the cost or value of implementing another, or two requirements may be mutually exclusive or, indeed, totally independent. To better support practitioners, a prioritising process should be able to manage requirements interdependencies, so that the impact of including or excluding one requirement can be observed by the decision makers, ideally in the cost–value diagram.
- *Lack of flexibility in structuring requirements.* In its initial form, the cost–value approach treats all requirements as belonging to a single level. Structuring the requirements into a hierarchy is worthwhile for at least three reasons. Firstly, in large projects the set of requirements is likely to be elaborated as a layered hierarchy. According to Davis [12], 'in a typical complex application, the best way to organise requirements is probably a multilevel hierarchy where each level corresponds to a different grouping criteria'. Secondly, layering aids comprehension by stakeholders since they can view requirements at an appropriate level of abstraction. Finally, as will be seen, layering facilities reducing the number of comparisons required in the cost–value approach. Therefore, it is desirable that the approach allows for hierarchical representation of requirements.
- *Lack of tool support.* The initial industrial use of the cost–value approach had highlighted the clerical burden involved in managing even modest numbers of requirements. If the approach is to be used, routinely, in large-scale projects some computer-based support is essential.

4. Process Improvement

4.1. Reducing the Required Number of Pair-wise Comparisons

The most pressing problem in the current version of the cost–value approach is the number of pair-wise comparisons required in large-scale development projects. As the number of candidate requirements increases, the number of comparisons required grows polynomially. Techniques to reduce the number of comparisons, while still producing results with sufficient quality, have been developed [13–15]. The generic name for these techniques is Incomplete Pair-wise Comparisons (IPC).

For a development project having n candidate requirements, $n-1$ pair-wise comparisons would be enough to rank the requirements. However, the AHP requires $n(n-1)/2$ pair-wise comparisons to be carried out. The redundancy is useful since stakeholders can never be perfectly consistent in their judgements. The question to be asked is how much redundancy really is needed to cope with the inconsistent judgements. The IPCs aim to reduce the number of pair-wise comparisons, while still keeping enough redundancy to produce high-quality results. This is possible since it is often a waste of time to perform all pair-wise comparisons. The quality improvement due to the last few pair-wise comparisons is minor and will hardly affect the results at all. Ideally, a support tool for the prioritising process should provide *stopping rules* indicating when additional pair-wise comparisons are no longer necessary. Such stopping rules can be divided into two separate categories: local stopping rules (LSR) and global stopping rules (GSR). Local stopping rules operate only on a single node in the requirements hierarchy, whereas global stopping rules operate with respect to the complete requirements hierarchy.

4.1.1. Local Stopping Rule

An LSR is a means of reducing the required effort in the AHP. The basic principle of these local stopping rules is that they require a stakeholder to create at least a spanning tree in a directed graph (i.e., the graph is at least minimally connected). To create such a spanning tree a minimum of $n-1$ pair-wise comparisons must be completed. Thus the stakeholder leaves a number of pair-wise comparisons 'blank'. Let us assume that all comparisons are carried out, except for that between requirements A and D, which is left 'blank'. In the directed graph which can be constructed by the completed comparisons, there is at least one path from requirement A to D. The 'blank' value (or cost) can be computed by taking the geometric mean of the intensities of all possible paths connecting requirements A and D. To illustrate the principle, if requirement A is determined to be twice as expensive as requirement B, which in turn is estimated to be three times as expensive as requirement C, the relationship of A to C can easily be calculated. Once the blanks have been filled, the AHP is then used, as usual, for the computation of the value (or cost) distribution.

Harker [13] proposes an LSR to reduce the number of redundant pair-wise comparisons at a node by ordering the comparisons in decreasing informational value and by stopping the process when the expected added value of the next comparisons decreases below a certain threshold. This local stopping rule is very effective in large-scale applications since it may require as few as $n-1$ pair-wise comparisons. The rule requires further comparisons only until the maximum absolute difference in the value (or cost) of any requirement from one comparison to the next is $<a\%$, where a is a given constant. Then a stakeholder can stop making comparisons since the remaining comparisons will not contribute significantly.

Another approach for reducing the number of pair-wise comparisons at a node is proposed by Shen et al. [15]. In their approach requirements at a node in the hierarchy are divided into several subsets, such that these subsets have one requirement in common. The pair-wise comparisons are then performed for each of these subsets, where the common requirement serves as a benchmark. This requirement determines how much value (or cost) is contained in each of the subsets. For example, assume requirements A, B, C, D and E are to be prioritised. A, B and C are compared in one subset and C, D and E in another. If A is twice as valuable as C, and E is half as valuable as C in its subset, we can easily compute the relative value of A and E.

4.1.2. Global Stopping Rule

A GSR goes further than reducing the number of pair-wise comparisons at a node. Instead it reduces the number of pair-wise comparisons for a whole hierarchy [14]. A motivating concept for this approach is the use of global value (or cost) as input to forthcoming comparisons. It is reasonable to assume that fewer comparisons are needed for those nodes which have a low overall impact on the final results of the process. This means that the greatest effort is put into those nodes which matter most. To illustrate the concept, assume there are four branches in the hierarchy with the relative values: 20%, 50%, 28% and 2%. Then the effort required to further analyse the '2%' branch may not be worthwhile.

One global stopping rule which can be used to terminate further comparisons is the following:

$$Maxdif < \frac{Kstop}{\sqrt{N} \cdot GW}$$

where N is the number of nodes at a given level in the hierarchy. *Maxdif* is the maximum absolute difference in the value (or cost) from one comparison to the next. *GW* is the global value (or cost) of the node, and *Kstop* is a stopping constant supplied by the stakeholder.

4.2. Managing Requirements Interdependencies

In almost all large-scale projects, some requirements are interdependent. When using prioritising as a means for selecting among candidate requirements some kinds of interdependency are especially relevant. For example, given that a project manager has selected requirement A for implementation, then this may change the cost for selecting requirement B; either B may be more expensive to implement or may be less expensive. A reasonable list of interdependencies for our purposes would be as follows:

- *Cannot-exist.* Given that requirement A has been chosen for implementation, then another requirement, say requirement B, cannot be implemented. For example, if the requirement 'stand-alone system' is chosen, then the requirement 'support for electronic mail features' cannot be met.
- *Must-exist.* Given that requirement A has been selected for implementation, then requirement B has to be chosen too. For example, if the requirement 'Internet access' shall be met, then the requirement 'providing network facilities' must also be met in order for the former requirement to make sense.

• *Positive cost.* Given that requirement A has been chosen for implementation, then the cost of implementing requirement B falls. For example, if 'English spell checker' has been chosen for implementation, it is likely that 'Swedish spell checker' will be cheaper.

• *Negative cost.* Given that requirement A has been chosen for implementation, then the cost of realising requirement B increases. For example, if 'long stand-by time' is chosen for implementation for a cellular phone, then 'light weight' is likely to be more expensive since the first requirement demands large batteries, which may conflict with the second requirement.

• *Positive value.* Given that requirement A has been chosen for implementation, then requirement B becomes more valuable. For example, if 'standard interface alignment' is chosen for implementation, then 'a cut-and-paste feature' increases in value.

• *Negative value.* Given that requirement A has been chosen for implementation, then requirement B becomes less valuable. For example, if 'on-line help' is chosen for implementation, then having a 'detailed manual' may decrease in value.

A first step towards supporting the interdependencies in the cost–value approach would therefore be to provide means for defining these relations between requirements. While it is clear that a more comprehensive treatment of requirements interdependencies would be very desirable, it was felt by practitioners that, at least initially, these simple relations would be adequate.

4.3. Allowing Hierarchical Representation

In large-scale developments, requirements are structured in different ways, such as hierarchies, in order to get a better overview of the requirements, but also to minimise the impact of changes to the requirements. Moreover, as its name implies, the AHP was designed to support such hierarchical structures. Hierarchies are useful in the prioritising process since they reduce the required number of pair-wise comparisons. In a hierarchical structure, only those requirements at the same node are pair-wise compared. Moreover, the global stopping rule described above depends on a hierarchical structure in order to function. An effective process for prioritising requirements must consequently be capable of managing hierarchical structures of requirements as well as the more simple flat structures of requirements.

5. Tool Support for the Improved Process

The primary objective of the support tool was to reduce the clerical overhead of the process. In addition, the major improvements in the process, which have been detailed above, imply an additional clerical and computational load so, if these were to be used in practice, a support tool was essential.

5.1. Support Tool Requirements

After discussion with actual and potential users, the following set of general requirements for the tool was selected from a wider candidate set, using the cost–value approach.

1. *Storage and editing.* Allow stakeholders to enter, store, edit and display the requirements structure (either hierarchical or flat).
2. *Facilitate pair-wise comparisons.* Traverse the requirements structure and present pairs of requirements for the decision makers to make comparisons according to value or cost.
3. *Carry out the AHP.* Calculate the relative value and cost distributions based on the pair-wise comparisons and display them as separate lists or as a cost–value diagram.
4. *Estimate consistency.* Compute and display the consistency indices and ratios as prescribed by the AHP.
5. *Support selection of requirements.* Allow the user to select a subset of requirements from the cost–value diagram and provide the user with feedback on the resulting cost and value of the subset.
6. *Support incomplete pair-wise comparison.* Implement at least one global stopping rule and at least one local stopping rule. Allow the user to switch the use of the stopping rules on and off.
7. *Support requirement interdependencies.* Implement means for specifying and using requirement interdependencies. The cost–value diagram should reflect the interdependencies between those requirements selected for implementation. Allow the user to switch the use of the interdependencies on and off.

5.2. Support Tool Implementation

The support tool was implemented through a series of prototypes [16]. A first version of the support tool covering the initial cost–value approach was implemented. Once the first version was working, tests showed that, despite its simplicity, it speeded up the

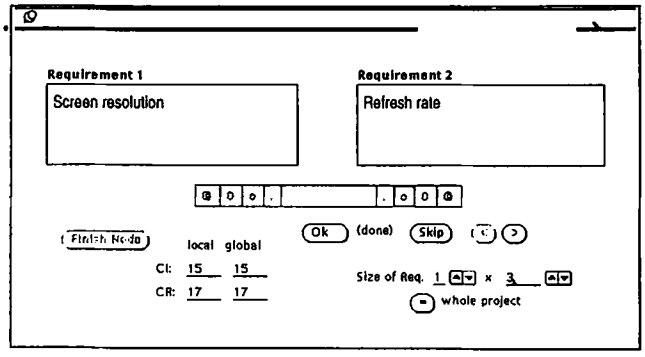

Fig. 1. An example of a pair-wise comparison of candidate requirements.

prioritising process and was welcomed by the practitioners. Additional features were added successively.

An editor was designed and implemented which stores structures of requirements and presents them for comparisons in the format shown in Fig. 1. The tool carries out the AHP and can produce cost–value diagrams such as that shown in Fig. 2. Clicking on the information button will display the consistency indices and ratios, indicating the level of judgmental error in the pair-wise comparisons. The user can select a subset of requirements, shown as filled squares in the figure, and the tool displays the relative value and cost of the selected subset. One global stopping rule and two local stopping rules, as described earlier, were implemented in the support tool, and the user is free to switch them on and off at any time.

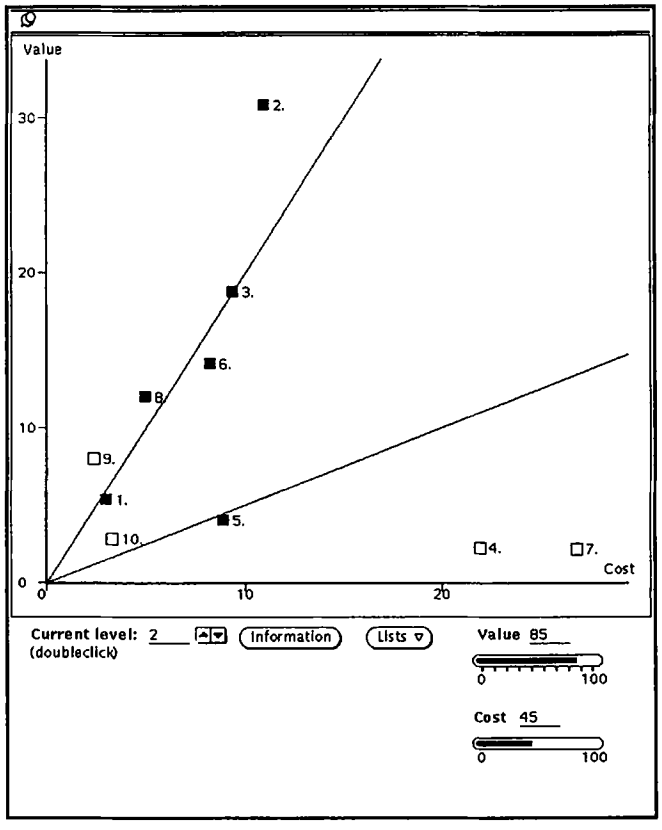

Fig. 2. An example of a cost–value diagram where filled boxes represent selected requirements.

Implementing the interdependencies was the most complex aspect. In its present very basic form, the support tool allows a user to express simple interdependencies such as the following:

- Requirement A makes requirement B necessary.
- Requirement C makes requirement D impossible.
- Requirement A makes requirement C cheaper (the cost for requirement C is decreased by either 30% or 60%).
- Requirement E makes requirement F more valuable (the value for requirement F is increased by either 30% or 60%).

The interdependencies are taken into account when selecting requirements on the cost–value diagram. If, for instance, the interdependencies listed above were used, requirement F is moved to a new position on the cost–value diagram corresponding to its new estimated value when requirement E is selected or deselected for implementation. In this way the decision makers can see immediately the approximate impact of the interdependency.

6. Industrial Application and Evaluation

Having developed the support tool, we wanted to have it evaluated further by experienced developers who had not previously been involved with the cost–value approach. Senior developers in a commercial telecommunications project at Ericsson Radio Systems AB, who were developing additional functions for Base Stations Controllers, applied the support tool in their requirements work.

A total of more than 200 requirements were identified but, to minimise the developers' time commitment, the study was restricted to the 23 requirements dealing with product improvement. Furthermore, we emphasised the evaluation of the process and the usefulness of the process and its support tool rather than studying other details such as the user interface. Instead of producing test cases and documenting the behaviour of the users, we provided help for them to get started, and interviewed them after the sessions, discussing the outcome and their experiences.

We introduced the concept of pair-wise comparisons and the use of the support tool to the developers by running a simple example. This instruction and subsequent discussions required about 30 minutes. The developers then used the support tool under our supervision to pair-wise compare the 23 requirements according to the criteria value and cost of implementation. Performing pair-wise comparisons of 23 requirements required $23(23-1)/2 = 253$ comparisons for each criterion. By using one of the implemented local stopping rules, that provided by Harker [13], the number of comparisons was greatly reduced, from 253 to 60 per criterion.

6.1. Results

The developers completed their task in about 60 minutes. Interestingly, at the same pace a total of 4 hours would have been required without the local stopping rule. The resulting cost–value diagram of the 23 requirements is shown in Fig. 3. By selecting for implementation all requirements except numbers 7, 10, 13 and 18 the total value is only decreased to 91%, but the cost would be reduced significantly, to 73% of the cost of implementing all the candidate requirements.

6.2. Observations

After the evaluation session we spent 30 minutes with each test person discussing the results. In general the reactions were very positive. The developers approved of our objectives and were convinced that the cost–value approach was very attractive. They were impressed by the speed with which they did the rankings, supported by the tool, and felt that the results were both quicker and more accurate than the ordinal scale they were used to. In summary, they judged that the process and the support tool had excellent potential to save time in prioritising requirements and that, if a suitable commercial version was available, it could be adopted for use at Ericsson. The main obstacle to this at present was the lack of flexibility in the tool. It should cater fully for changing both the contents and the structure of the requirements and, in particular, it should allow the input and use of actual costings. For example, when the developers had good estimates of the implementation cost of some of the requirements, they found it tedious to compare these costs to one another, rather than enter the 'known' costs once and for all so that they could concentrate on the relationship between more problematic costs.

As final note, the improved process and its support tool were highly appreciated and regarded as useful in commercial software development projects. The developers agreed that this effective and accurate approach to visualising requirements value and cost is very beneficial in commercial projects.

7. Conclusions and Further Steps

In this article we have outlined directions towards an improved process for prioritising software requirements in large-scale development projects. This approach allows requirements engineers effectively and accurately to prioritise requirements based on pair-wise comparisons in two dimensions: the requirements' value, and their cost of implementation. Practical improvements have been implemented which include global and local stopping rules for reducing the required number of pair-wise comparisons and an initial strategy for dealing with requirements interdependencies. The stopping rules proved to be particularly useful in the application project. They reduced

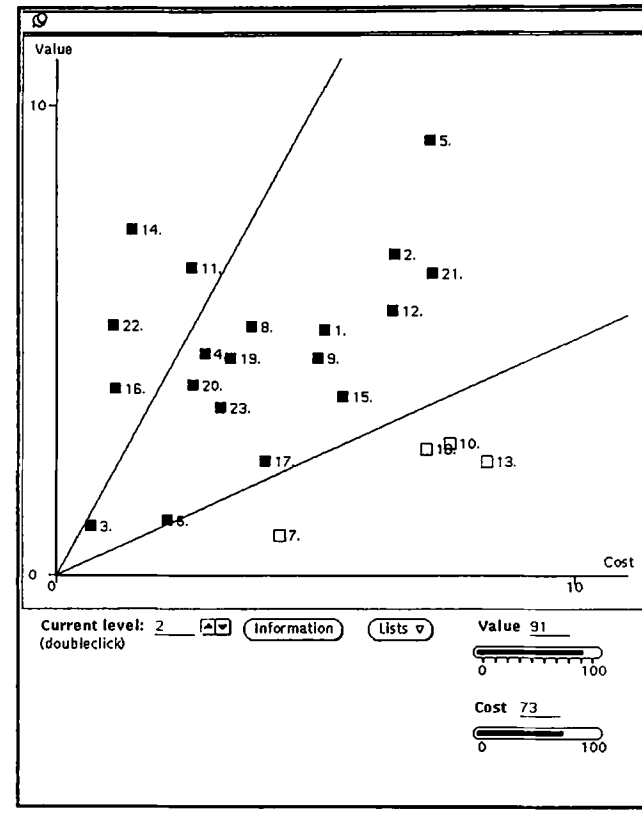

Fig. 3. Cost-value diagram for the 23 candidate requirements in the evaluation study.

the required number of pair-wise comparisons by as much as 75%. A support tool covering the cost–value approach and a number of additional features was developed and applied in a commercial project at Ericsson Radio Systems AB. Qualitative responses indicated a strong need for effective processes for prioritising requirements. That need was, to a large extent, fulfilled by the improved cost–value approach but, of course, further improvements are always possible.

7.1. Future Work

We have identified two important aspects for future work. The first is to evaluate the improved cost–value approach in other commercial projects. It is planned to perform in-depth interviews and to collect and analyse qualitative data. The second aspect is to further improve the support tool. The following features are regarded as desirable by the practitioners using the current version:

- *Pair-wise comparison rationale.* Someone might want to look at the pair-wise comparisons done a long time ago, or by someone else. It would be easier to understand the pair-wise comparisons if the relevant stakeholder could add a comment explaining each decision.
- *Indication of contradictory comparisons.* Whenever a user is inconsistent, i.e., provides pair-wise comparisons that contradict each other, the support tool should give an indication as to where the inconsistency is to be found. By showing the user the contradicting pair-wise comparisons it could pinpoint the requirements involved. Occasionally a decision maker can provide contradictory circular pair-wise comparisons. This implies that a user, for example, claims requirement A to be of higher value than requirement B, which in turn, the user claims to be of higher value than requirement C, which is deemed to be of higher value than requirement A. Such circular, inconsistent judgements should be detectable by a support tool.
- *Organisational adaptability.* The support tool should help its users within the context of an organisation's existing development process. It may be the case that the organisation's process must be adapted so as to include the cost–value approach, but it must be possible to tailor the support tool to fit the context.
- *Allowing multi-user features.* More than one decision maker can be involved in the requirements prioritising process. It is thus important that the input be taken from multiple sources. At least three ways can be identified to allow multi-user features. The possibility is that the stakeholders prioritise different parts of the requirements hierarchy, and together these constitute the complete distribution. Alternatively, all stakeholders prioritise all parts of the requirements hierarchy, and then the project manager uses his or her judgement to reconcile the priorities. The third option is that more than one stakeholder prioritises the entire requirements hierarchy. In this case the geometric mean of the stakeholders' pair-wise comparisons is used to reconcile their priorities. The decision as to which option to use will depend on the range of expertise available to make the evaluations.
- *Supporting requirements evolution.* As commercial projects progress, the requirements are likely to change. It is also possible that later releases of software systems will have differing requirements. If, in these cases, all requirements were to be re-prioritised, the effort involved is likely to be overwhelming. It would be preferable that, when requirements are added or changed, these must be pair-wise compared to each other, and to some of the other existing requirements. But they must only be compared to a minimum of other requirements in order to gain sufficient information. In the case where a requirement is removed, re-prioritising ought to be completely avoided. Instead, the remaining requirements' values (or costs) should be recalculated.
- *Linking absolute costs to relative costs.* In the evaluation session of the improved process and its support tool the developers asked for more flexibility regarding known costs for requirements. Such knowledge can be very useful for assessment purposes in the requirements work as well as in the planning work. Consider the case where a stakeholder estimates the cost of meeting one requirement as $10,000. Moreover, assume that if, after all pair-wise comparisons have been carried out, this requirement stands for 17% of the total cost. Then a first, rough estimate of meeting all requirements can be calculated as $50,000/0.17 which is almost $300,000. In addition, each of the other requirements' costs can be calculated by multiplying their relative cost by the total estimated cost. This approach must of course be used with great care. We do believe, however, that the speed and simplicity of this approach can be very useful and important for practitioners as an early cost estimation. It should also be possible to reduce the number of comparisons by not requiring the use of the support tool to make redundant comparisons that involve absolute costs.
- *Comparing and evaluating techniques for reducing the number of required pair-wise comparisons.* Reducing the number of pair-wise comparisons is important in commercial software development, due to time and

resource constraints. It is therefore of great interest to implement and evaluate techniques for this purpose. Only a small number of the available approaches have been explored to date, and a fuller evaluation would be well worthwhile.

• *Managing requirements interdependencies*. The initial approach for dealing with requirement interdependencies proposed in this article needs further development and evaluation.

Acknowledgements

This work has been supported by Ericsson Radio Systems AB and the Swedish National Board for Industrial and Technical Development, project number 9303280-2.

References

1. Karlsson J, Ryan K. Supporting the selection of software requirements. In 8th International workshop on software specification and design, 1996, pp. 146–149
2. Karlsson J, Ryan K. Prioritising requirements using a cost–value approach. IEEE Software, to appear
3. Saaty TL. The analytic hierarchy process. McGraw-Hill, New York, 1980
4. Potts C. Software engineering research revisited. IEEE Software 1993; 10(5): 19–28
5. Carr W, Kemmis S. Becoming critical: educational, knowledge and action research. Falmer Press, 1986
6. Mansell G. Action research in information systems development. J Inform Syst 1991; 1(1): 29–40
7. Bubenko JA Jr. Challenges in requirements engineering. In 2nd IEEE international symposium on requirements engineering, 1995, pp. 160–162
8. Lubars M, Potts C, Richter C. A review of the state of the practice in requirements modeling. In IEEE international symposium on requirements engineering, 1993, pp. 2–14
9. Zave P. Classification of research efforts in requirements engineering. In 2nd IEEE international symposium on requirements engineering, 1995, pp. 214–216
10. Karlsson J. Software requirements prioritising. In 2nd IEEE international conference on requirements engineering, 1996, pp. 110–116
11. Stevens SS. On the theory of scales of measurement. Science 1946; 103: (2648) 677–680
12. Davis A. Software requirements: objects, functions and states. Prentice-Hall, Englewood Cliffs, NJ, 1993
13. Harker P. Incomplete pair-wise comparisons in the analytic hiearchy process. Math Modelling 1987; 9(11): 837–848
14. Millet I, Harker P. Globally effective questioning in the analytic hierarchy process. Eur J OR 1990; 48(1): 88–97
15. Shen Y, Hoerl AE, McConnell W. An incomplete design in the analytic hierarchy process. Math Comput Modelling 1992, 16(5): 121–129
16. Olsson S. Improving and supporting the contribution-based method (CBM), MSc thesis 9597, Department of Computer and Information Science, Linköping University, 1996