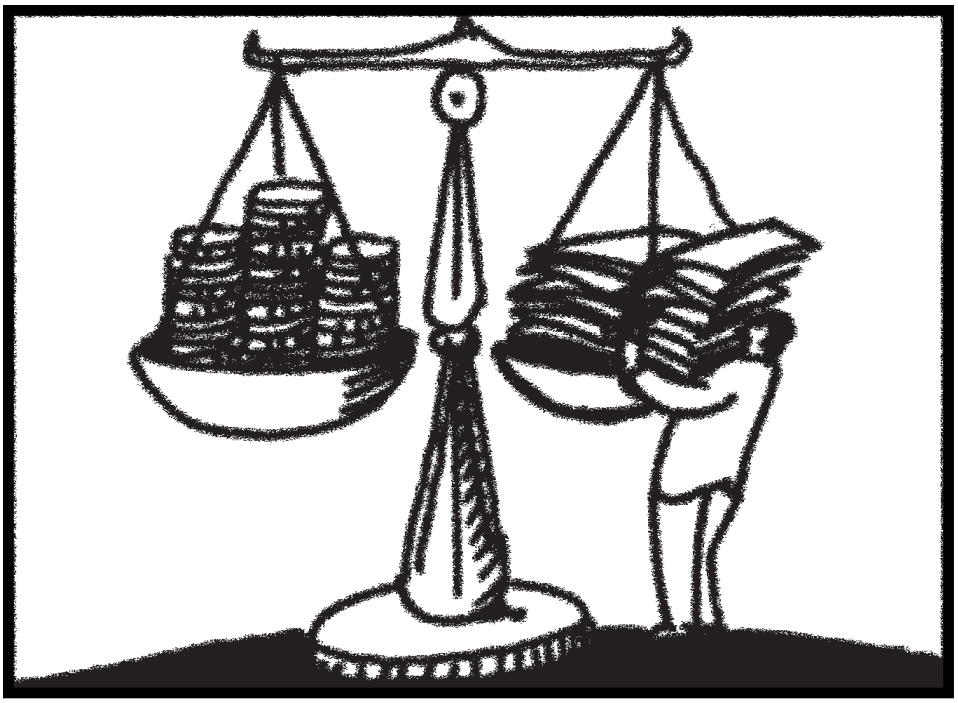
A Cost–Value Approach for Prioritizing Requirements

JOACHIM KARLSSON
Focal Point AB

KEVIN RYAN
University of Limerick



Deciding which requirements really matter is a difficult task and one increasingly demanded because of time and budget constraints. The authors developed a cost–value approach for prioritizing requirements and applied it to two commercial projects.

Developing software systems that meet stakeholders' needs and expectations is the ultimate goal of any software provider seeking a competitive edge. To achieve this, you must effectively and accurately manage your stakeholders' system requirements: the features, functions, and attributes they need in their software system.[1] Once you agree on these requirements, you can use them as a focal point for the development process and produce a software system that meets the expectations of both customers and users. However, in real-world software development, there are usually more requirements than you can implement given stakeholders' time and resource constraints. Thus, project managers face a dilemma: How do you select a subset of the customers' requirements and still produce a system that meets their needs?

0740-7459/97/$10.00 © 1997 IEEE

Authorized licensed use limited to: Norges Teknisk-Naturvitenskapelige Universitet. Downloaded on August 19, 2009 at 06:20 from IEEE Xplore. Restrictions apply.

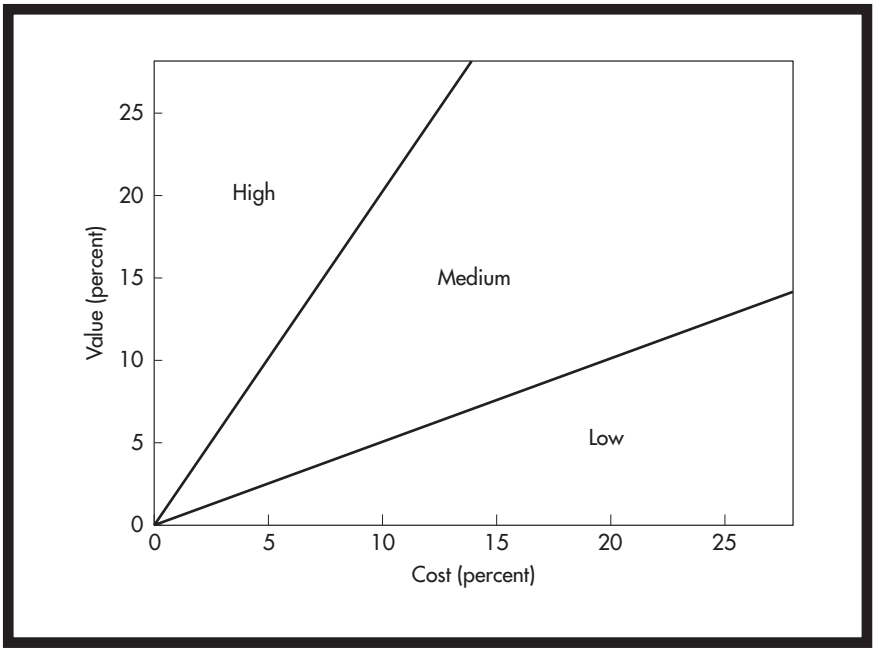


Figure 1. *Using AHP, you can calculate each candidate requirement's relative value and implementation cost and plot them on a cost-value diagram, such as the one shown here.*

Most software organizations carry out this selection process informally, and quite frequently produce software systems that developers, customers, and users view as suboptimal. Without techniques to make these crucial choices, this outcome is hardly surprising. Indeed, despite the recent rapid and welcome growth in requirements engineering research, managers still don't have simple, effective, and industrially proven techniques for prioritizing requirements. A recent survey[2] shows that few companies know how to establish and communicate requirements priorities; another[3] identified prioritization as a key but neglected issue in requirements engineering research.

Clear, unambiguous knowledge about requirement priorities helps you focus the development process and more effectively and efficiently manage projects. It can also help you

- make acceptable tradeoffs among sometimes conflicting goals such as quality, cost, and time-to-market;[4] and
- allocate resources based on the requirement's importance to the project as a whole.

Finally, when time-to-market is particularly important, knowing how to rank your requirements can help you plan releases by indicating which functions are critical and which can be added (and in what order) over successive releases.

We have developed an analytical tool for prioritizing requirements based on a cost–value approach. This tool can help you rank candidate requirements in two dimensions: according to their value to customer and users, and according to their estimated cost of implementation. Our method has been successfully applied in two commercial telecommunications software development projects.

A COST–VALUE APPROACH

A process for prioritizing software requirements must, on one hand, be simple and fast, and, on the other, yield accurate and trustworthy results. If both of these conditions are not met, the process is unlikely to be used in commercial software systems development. The prioritizing process also must hold stakeholder satisfaction as both the ultimate goal and the guiding theme. Shoji Shiba and his colleagues argue that there are three main factors in stakeholder satisfaction: quality, cost, and delivery.[5] For a software system to succeed, quality must be maximized, cost minimized, and time-to-delivery be as short as possible. Our cost-value approach prioritizes requirements according to their relative value and cost. Based on this information, software managers can make decisions such as which requirements to be excluded from the first release to keep the time-to-market at minimum.

We interpret *quality* in relation to a candidate requirement's potential contribution to customer satisfaction with the resulting system. *Cost* is the cost of successfully implementing the candidate requirement. In practice, software developers often calculate costs purely in terms of money. However, we found that prioritizing based on relative rather than absolute assignments is faster, more accurate, and more trustworthy.[6]

Why pairwise? To investigate candidate requirements, we use the Analytic Hierarchy Process,[7] which compares requirements pairwise according to their relative value and cost. The pairwise comparison approach includes much redundancy and is thus less sensitive to judgmental errors common to techniques using absolute assignments. The AHP actually indicates inconsistencies by calculating a consistency ratio of judgmental errors. The smaller the consistency ratio, the fewer the inconsistencies, and thus the more reliable the results. The box "The Analytic Hierarchy Process" on page 69 describes the AHP in more detail.

Process. There are five steps to prioritizing requirements using the cost–value approach.

1. Requirements engineers carefully review candidate requirements for completeness and to ensure that they are stated in an unambiguous way.
2. Customers and users (or suitable substitutes) apply AHP's pairwise comparison method to assess the relative value of the candidate requirements.
3. Experienced software engineers use AHP's pairwise comparison to estimate the relative cost of implementing each candidate requirement.
4. A software engineer uses AHP to calculate each candidate requirement's relative value and implementation cost, and plots these on a cost–value diagram. As Figure 1 shows, value is depicted on the y axis and estimated cost on the x axis.
5. The stakeholders use the cost–value diagram as a conceptual map for analyzing and discussing the candidate

Authorized licensed use limited to: Norges Teknisk-Naturvitenskapelige Universitet. Downloaded on August 19, 2009 at 06:20 from IEEE Xplore. Restrictions apply.

THE ANALYTIC HIERARCHY PROCESS

To make decisions, you identify, analyze, and make tradeoffs between different alternatives to achieve an objective. The more efficient the means for analyzing and evaluating the alternatives, the more likely you'll be satisfied with the outcome. To help you make decisions, the Analytic Hierachy Process compares alternatives in a stepwise fashion and measures their contribution to your objective.[1]

AHP in action. Using AHP for decision making involves four steps. We'll assume here that you want to evaluate candidate requirements using the criterion of value.

Step 1. *Set up the* n *requirements in the rows and columns of an* n × n *matrix.* We'll assume here that you have four candidate requirements: Req1, Req2, Req3, and Req4, and you want to know their relative value. Insert the *n* requirements into the rows and columns of a matrix of order *n* (in this case we have a 4 × 4 matrix).

Step 2. *Perform pairwise comparisons of all the requirements according to the criterion*. The fundamental scale used for this purpose is shown in Table A.[1] For each pair of requirements (starting with Req1 and Req2, for example) insert their determined relative intensity of value in the position (Req1, Req2) where the row of Req1 meets the column of Req2. In position (Req2, Req1) insert the reciprocal value, and in all positions in the main diagonal insert a "1." Continue to perform pairwise comparisons of Req1–Req3, Req1–Req4, Req2–Req3, and so on. For a matrix of order *n*, $n \cdot (n-1)/2$ comparisons are required. Thus, in this example, six pairwise comparisons are required; they might look like this:

	Req1	Req2	Req3	Req4
Req1	1	1/3	2	4
Req2	3	1	5	3
Req3	1/2	1/5	1	1/3
Req4	1/4	1/3	3	1

Step 3. *Use averaging over normalized columns to estimate the eigenvalues of the matrix* (which represent the criterion distribution). Thomas Saaty proposes a simple method for this, known as averaging over normalized columns.[1] First, calculate the sum of the *n* columns in the comparison matrix. Next, divide each element in the matrix by the sum of the column the element is a member of, and calculate the sums of each row:

	Req1	Req2	Req3	Req4	*Sum*
Req1	0.21	0.18	0.18	0.48	*1.05*
Req2	0.63	0.54	0.45	0.36	*1.98*
Req3	0.11	0.11	0.09	0.04	*0.34*
Req4	0.05	0.18	0.27	0.12	*0.62*

Then normalize the sum of the rows (divide each row sum with the number of requirements). The result of this computation is referred to as the *priority matrix* and is an estimation of the eigenvalues of the matrix.

$$\frac{1}{4} \cdot \begin{pmatrix} 1.05 \\ 1.98 \\ 0.34 \\ 0.62 \end{pmatrix} = \begin{pmatrix} 0.26 \\ 0.50 \\ 0.09 \\ 0.16 \end{pmatrix}$$

Step 4. *Assign each requirement its relative value based on the estimated eigenvalues.* From the resulting eigenvalues of the comparison matrix, the following information can be extracted:

- Req1 contains 26 percent of the requirements' total value,
- Req2 contains 50 percent,
- Req3 contains 9 percent, and
- Req4 contains 16 percent.

Result consistency. If we were able to determine precisely the relative value of all requirements, the eigenvalues would be perfectly consistent. For instance, if we determine that Req1 is much more valuable than Req2, Req2 is somewhat more valuable than Req3, and Req3 is slightly more valuable than Req1, an inconsistency has occurred and the result's accuracy is decreased. The redundancy of the pairwise comparisons makes the AHP much less sensitive to judgment errors; it also lets you measure judgment errors by calculating the consistency index of the comparison matrix, and then calculating the consistency ratio.

Consistency index. The consistency index (CI) is a first indicator of result accuracy of the pairwise comparisons. You calculate it as $CI = (\lambda\max - n)/(n-1)$. $\lambda\max$ denotes the maximum principal eigenvalue of the comparison matrix. The closer the value of $\lambda\max$ is to *n* (the number of requirements), the smaller the judgmental errors and thus the more consistent the result. To estimate $\lambda\max$, you first multiply the comparison matrix by the priority vector:

$$\begin{pmatrix} 1 & 1/3 & 2 & 4 \\ 3 & 1 & 5 & 3 \\ 1/2 & 1/5 & 1 & 1/3 \\ 1/4 & 1/3 & 3 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0.26 \\ 0.50 \\ 0.09 \\ 0.16 \end{pmatrix} = \begin{pmatrix} 1.22 \\ 2.18 \\ 0.37 \\ 0.64 \end{pmatrix}$$

Then you divide the first element of the resulting vector by the first element in the priority vector, the second element of the resulting vector by the second element in the priority vector, and so on:

$$\begin{pmatrix} 1.22 / 0.26 \\ 2.18 / 0.50 \\ 0.37 / 0.09 \\ 0.64 / 0.16 \end{pmatrix} = \begin{pmatrix} 4.66 \\ 4.40 \\ 4.29 \\ 4.13 \end{pmatrix}$$

Authorized licensed use limited to: Norges Teknisk-Naturvitenskapelige Universitet. Downloaded on August 19, 2009 at 06:20 from IEEE Xplore. Restrictions apply.

To calculate $\lambda \max$, average over the elements in the resulting vector:

$$\lambda \max = \frac{4.66+4.40+4.29+4.13}{4} = 4.37$$

Now the consistency index can be calculated:

$$CI = \frac{\lambda \max - n}{n-1} = \frac{4.37-4}{4-1} = 0.12$$

To find out if the resulting consistency index (CI = 0.12) is acceptable, you must calculate the consistency ratio.

Consistency ratio. The consistency indices of randomly generated reciprocal matrices from the scale 1 to 9 are called the random indices, RI.[1] The ratio of CI to RI for the same-order matrix is called the consistency ratio (CR), which defines the accuracy of the pairwise comparisons. The RI for matrices of order *n* are given below. The first row shows the order of the matrix, and the second the corresponding RI value.

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0.00	0.00	0.58	0.90	1.12	1.24	1.32	1.41	1.45	1.49	1.51	1.48	1.56	1.57	1.59

According to Table A, the RI for matrices of order 4 is 0.90. Thus, the consistency ratio for our example is

$$CR = \frac{CI}{RI} = \frac{0.12}{0.90} = 0.14 .$$

As a general rule, a consistency ratio of 0.10 or less is considered acceptable.[1] This means that our result here is less than ideal. In practice, however, consistency ratios exceeding 0.10 occur frequently.

REFERENCES

1. T.L. Saaty, *The Analytic Hierarchy Process*, McGraw-Hill, New York, 1980.

TABLE A
SCALE FOR PAIRWISE COMPARISONS

Relative intensity	**Definition**	**Explanation**
1	Of equal value	Two requirements are of equal value
3	Slightly more value	Experience slightly favors one requirement over another
5	Essential or strong value	Experience strongly favors one requirement over another
7	Very strong value	A requirement is strongly favored and its dominance is demonstrated in practice
9	Extreme value	The evidence favoring one over another is of the highest possible order of affirmation
2, 4, 6, 8	Intermediate values between two adjacent judgments	When compromise is needed
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*.	

Authorized licensed use limited to: Norges Teknisk-Naturvitenskapelige Universitet. Downloaded on August 19, 2009 at 06:20 from IEEE Xplore. Restrictions apply.

requirements. Based on this discussion, software managers prioritize the requirements and decide which will actually be implemented. They can also use the information to develop strategies for release planning.

CASE STUDY 1: THE RAN PROJECT

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 to identify, apply, and evaluate ways to improve the early phases of the software engineering process. As part of this collaboration, in January 1994 we were invited in to use the industry-as-laboratory approach,[8] performing in-depth case studies in an industrial environment. For the first study, we selected Ericsson's Radio Access Network project.

The goal of the RAN project was to identify and specify requirements for a system that would give managers information about mobile telephony system operation.[9] The project started small, with a staff of five, but as a result of our study it grew considerably and is now an umbrella for a portfolio of both research and development projects.

First steps. We identified 14 high-level requirements (services) that covered the main system functionality. These high-level requirements were intended to give managers information about issues such as capacity, coverage, and quality in a mobile communications system. Once we'd defined the 14 requirements, the project members reviewed and agreed on them. The prioritizing technique in use at that time was to rank-order the requirements on an ordinal scale ranging from 1 to 3, where 1 denotes highest priority. In practice, the requirements belonging to category 1 were then implemented and the rest discarded or postponed to future releases. Because we'd used this technique before and found it far from optimal,[6] we decided to prioritize RAN requirements using the cost–value approach.

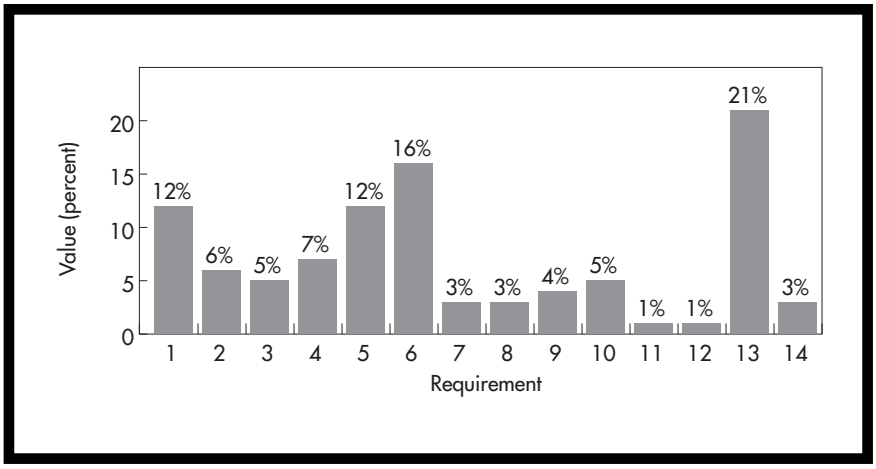


Figure 2. The value distribution of the 14 requirements in the RAN project.

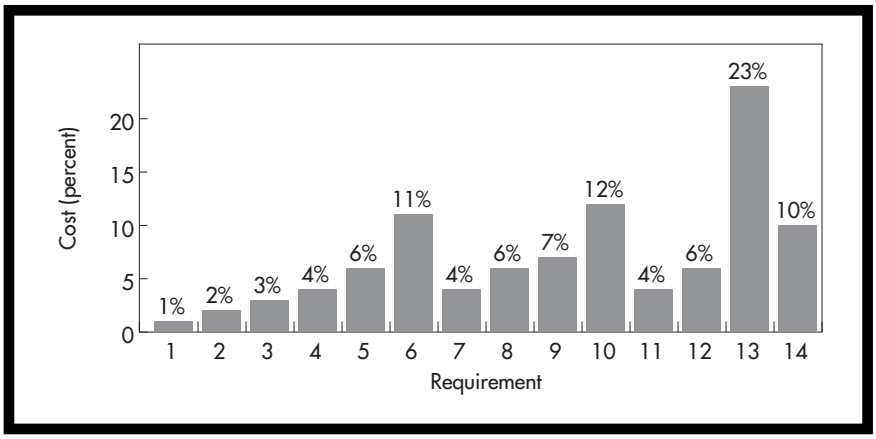


Figure 3. Estimated cost of requirements implementation in the RAN project. Requirements 6, 10, 13, and 14 constitute 56 percent of the total implementation costs.

We asked a group of experienced project members to represent customers' views and carefully instructed them on prioritizing requirements, making pairwise comparisons, choosing the scale to be used, and deciding how many comparisons would be needed. We also explained the importance of carrying out the pairwise comparisons carefully.

To begin, the project manager explained each candidate requirement and discussed it with project participants. He did this to make the requirements more clear and reduce subsequent misinterpretations. We distributed sheets outlining the 91 unique pairs of requirements, including the fundamental scale (as shown in Table A in "The Analytic Hierarchy Process" on page 70). Participants then performed pairwise comparisons of the candidate requirements, first according to value and later, in a separate session, according to the estimated implementation cost.

We let the participants work with the requirement pairs in any order they chose, allowing for retraction during the comparison process. The session was not moderated and participants worked at their own pace. Discussions were allowed, though in fact there were very few. Completing the cost–value approach took about an hour. When all 14 requirements had been pairwise compared, we calculated the value distribution and the cost distribution, as well as the consistency indices and ratios of the pairwise comparisons. There were some judgmental errors, since the consistency ratios for both value and cost were computed as 0.18. Based on the resulting distributions, we outlined the candidate requirements in a cost–value diagram and presented the results to the project members.

Requirements' value. Each requirement's determined value is relative and based on

Authorized licensed use limited to: Norges Teknisk-Naturvitenskapelige Universitet. Downloaded on August 19, 2009 at 06:20 from IEEE Xplore. Restrictions apply.

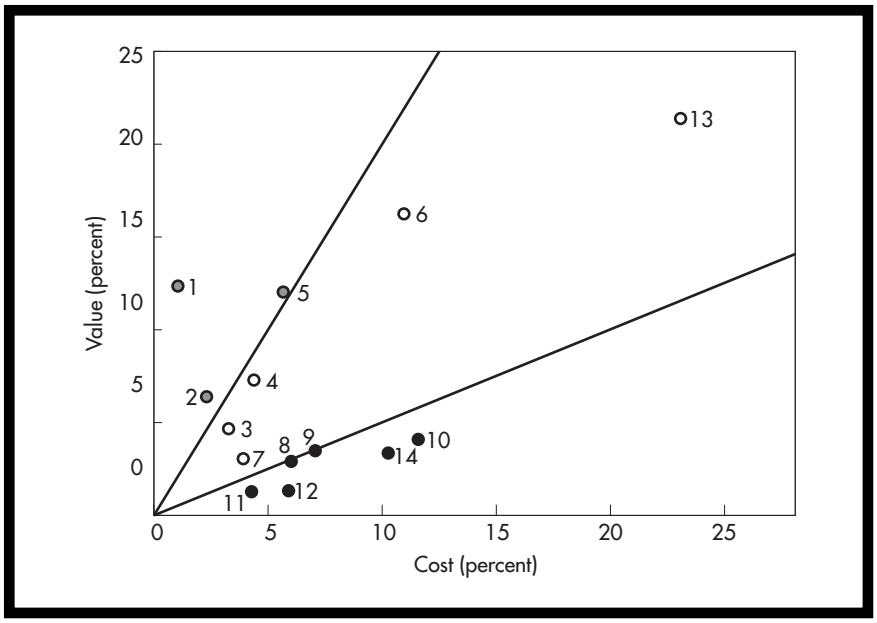


Figure 4. *Cost–value diagram for the RAN project requirements. By not implementing the requirements that contribute little to stakeholder satisfaction, such as 10, 11, and 12, you can significantly reduce the cost and duration of development.*

a ratio scale. This means that a requirement whose determined value is 0.10 is twice as valuable as a requirement with a determined value of 0.05. Moreover, the sum of all requirements value measures is always 1. Thus, a requirement with a determined value of 0.10 represents 10 percent of the total value of the requirements set. Figure 2 shows the value distribution of the 14 requirements in the RAN project. As the figure shows, the value of individual requirements can vary by orders of magnitude. The four most valuable requirements—1, 5, 6, and 13—constitute 61 percent of the total value; the four least valuable requirements—7, 8, 11, and 12—contribute a mere 8 percent. At the extremes, requirement 13 is about 20 times as valuable as requirement number 11.

Requirements' cost. A requirement's estimated cost is also relative and based on a ratio scale; the sum of all costs is again 1. Figure 3 shows the estimated cost of RAN's 14 requirements, which can again vary by orders of magnitude. The four most expensive requirements—6, 10, 13, and 14—constitute 56 percent of the total cost; the four least expensive requirements—1, 2, 3, and 11—account for only 10 percent. Looking again at the extreme values, requirement number 13 is about 20 times as expensive to implement as requirement number 1.

Requirements cost–value analysis. Figure 4 shows the cost–value diagram of the 14 requirements. For discussion purposes, we divide cost–value diagrams into three distinct areas:

- requirements with a high ratio of value to cost (a value–cost ratio exceeding 2),
- requirements with medium ratio of value to cost (a value–cost ratio between 0.5 and 2), and
- requirements with a low ratio of value to cost (a value–cost ratio lower than 0.5).

As such, we infer that requirements 1, 2, and 5 fall into the high ratio category; requirements 3, 4, 6, 7, and 13 into the medium ratio category; and requirements 8, 9, 10, 11, 12, and 14 into the low ratio category. Based on these categories, the software managers were able to effectively and accurately prioritize their requirements.

The cost–value diagram clearly facilitates requirements selection. If, hypothetically, you chose to implement all requirements except numbers 10, 11, and 12, the software system's value for its customers would be 94 percent of the possible maximum, while the cost would be reduced to 78 percent of the cost for implementing all requirements. In general, by not implementing the requirements that contribute little to stakeholder satisfaction, you can significantly reduce the cost and duration of development.

CASE STUDY 2: THE PMR PROJECT

Encouraged by the apparent usefulness and effectiveness of the cost–value approach in the RAN project, we undertook a second case study. We picked a project that was developing a fourth release: the Performance Management Traffic Recording project. PMR is a software system that enables recording and analysis of mobile telecommunications traffic. The project began in 1992 with a full-time staff of 15 people, and has delivered 10 releases of varying sizes thus far.

At the time we joined the project, the system's third release was installed and running at customer sites. Many new requirements had emerged that had to be taken into account in planning the next release. We divided these new requirements into three categories: those demanding traditional defect correction, those requiring performance enhancement, and those suggesting added functionality. We decided to prioritize only the last category because both the project managers and the customers agreed that all defects had to be corrected and performance had to be enhanced. However, the exact functions to be added were up for negotiation.

The 11 high-level functional requirements dealt with issues such as presentation, sorting, and structuring new types of information. To prioritize these requirements each project member had to complete 55 pairwise comparisons for each criterion using the cost–value ap-

Authorized licensed use limited to: Norges Teknisk-Naturvitenskapelige Universitet. Downloaded on August 19, 2009 at 06:20 from IEEE Xplore. Restrictions apply.

proach; this required slightly over 30 minutes. This rate is in line with the effort on the RAN project.

Requirements' value. Figure 5 shows the value distribution of the 11 requirements in the PMR project. Once again, the three most valuable requirements carry most of the value: requirements 4, 5, and 6 account for 63 percent.

Requirements' cost. Figure 6 shows the cost distribution of the 11 requirements. The three most expensive requirements—4, 5, and 9—account for 57 percent of implementation costs.

Requirements' cost–value analysis. Figure 7 shows the candidate requirements in the PMR project in a cost–value diagram. Of the 11 candidate requirements in the PMR project, two fall into the high ratio, six into the medium ratio, and three into the low ratio category. This illustrates how the management task of release planning, for example, is aided by the cost–value diagram. If the requirements with high and medium ratios were selected for implementation, 95 percent of the value would be obtained at 75 percent of the cost. Again, this suggests that you can deliver a software system with substantial customer satisfaction at a significant reduction in cost.

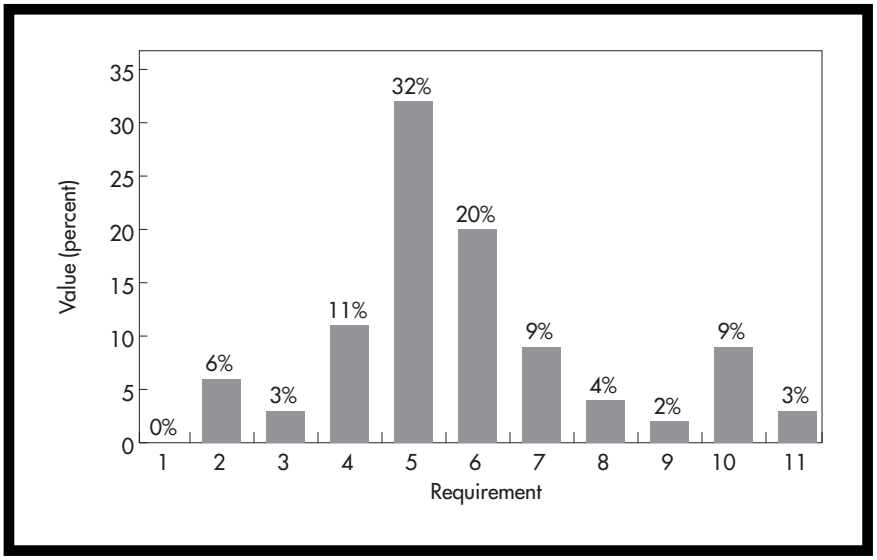


Figure 5. *The requirements' value in the PMR project. Requirements 4, 5, and 6 account for 63 percent of the value.*

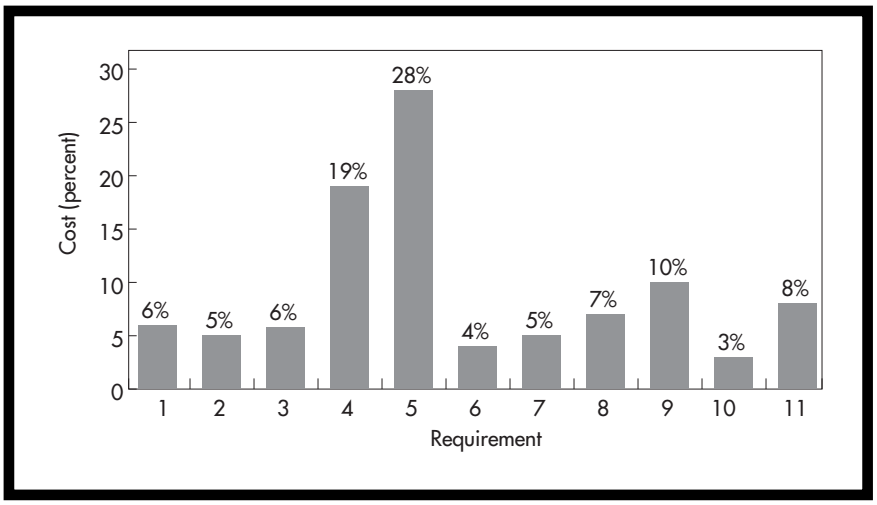


Figure 6. *The estimated cost of implementing the 11 requirements in the PMR project.*

Software engineering has been criticized for lacking the trade-off analysis that is always part of multidisciplinary systems engineering.[10] We believe the cost–value approach is a useful first step in filling this need. In some respects, our approach is similar to that of the Quality Attribute Requirements and Conflict Consultant tool within Barry Boehm's Win-Win system.[11] However, because of the mathematical basis of AHP, our approach yields more concrete results and could be a useful complement within the Win-Win environment. Similarly, recent work in the field of software architecture provides mathematical analyses of the quantitative design space,[12] such as spectrum analysis and contribution analysis. The cost–value approach is an important addition to these approaches as well, as it is more visible, more robust, and easier to use.

There are still a number of problems to be overcome if our approach is to be easily adopted by practitioners. Although our initial users found the cost–value approach intuitive and more useful than traditional approaches, they also found carrying out all the required pairwise comparisons tedious. They sometimes got distracted and had to backtrack to check the consistency of earlier pairwise comparisons.

We have identified several additional issues to be resolved. First, the method takes no account of interdependencies between requirements, leaving software managers to deal with them. For example, to implement a low-cost, high-value requirement, you might have to implement a high-cost, low-value one as well. This situation did not arise in our case studies, where the requirement sets were quite small. With a larger number of requirements, this could be a major consideration. We are continuing to work on these issues. A more detailed account of this work, including our experience with a prototype tool, will be published in *Requirements Engineering Journal*, Vol. 2, No. 1 (Springer-Verlag, 1997). More requirements will also raise the problem of complexity, since the number of pairwise comparisons is of $O(n^2)$. Thus, we are now developing adequate tools to support the pairwise comparison process, to minimize the number of comparisons required, and to cater to the interdependen-

Authorized licensed use limited to: Norges Teknisk-Naturvitenskapelige Universitet. Downloaded on August 19, 2009 at 06:20 from IEEE Xplore. Restrictions apply.

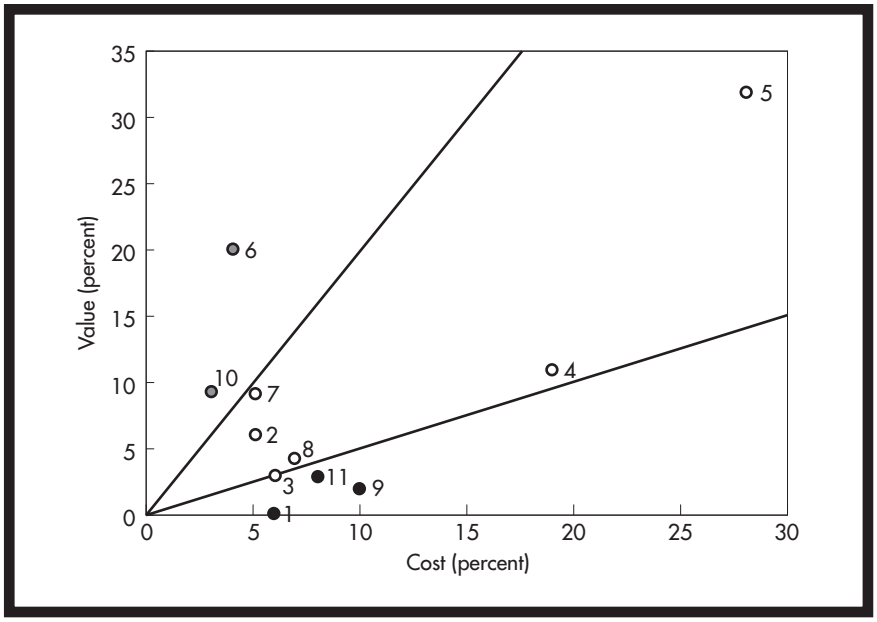


Figure 7. *Cost–value diagram depicting the PMR project requirements.*

cies that will inevitably arise. This tool will also be capable of storing information about the pairwise comparisons such as the people responsible, their rationale, and their assumptions.

Our cost–value method is based on a well-established analytical technique and, with reasonable effort, provides a clear indication of the relative costs and values of all candidate requirements. The case studies have shown that, even in its present form, the cost–value approach is useful for prioritizing requirements. It may also be especially applicable as an aid for requirements selection, since we have found that both the value and cost of requirements can vary by orders of magnitude. Such differences are effectively visualized through cost–value diagrams, which let management take action to maximize stakeholder satisfaction. The diagrams can also be used to prioritize requirements over several release cycles.

Communications between customers, users, and project managers are always likely to be difficult. A clear understanding of the choices involved in requirements selection can greatly assist this communication. We believe that our cost–value approach lays the foundation for a clear, sound, and usable method for determining requirements priorities. ◆

ACKNOWLEDGMENTS

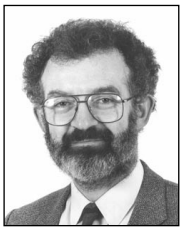
This work was supported by Ericsson Radio Systems AB and The Swedish National Board for Industrial and Technical Development, project number 9303280-2. We also thank the anonymous reviewers for providing valuable comments, and Lena Bjerlöw, Stefan Olsson, and Kristian Sandahl for commenting on a draft of this article.

REFERENCES

1. A. Davis, *Software Requirements: Objects, Functions and States*, Prentice Hall Int'l, Englewood Cliffs, N.J., 1993.
2. M. Lubars, C. Potts, and C. Richter, "A Review of the State of the Practice in Requirements Modeling," *Proc. IEEE Int'l Symp. Req. Eng.*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1993, pp. 2-14.
3. J. Siddiqi and M.C. Shekaran, "Requirements Engineering: The Emerging Wisdom," *IEEE Software*, Mar. 1996, pp. 15-19.
4. B. Curtis, H. Krasner, and N. Iscoe, "A Field Study of the Software Design Process for Large Systems," *Comm. ACM*, Dec. 1988, pp. 1268-1287.
5. S. Shiba, A. Graham, and D. Walden, *A New American TQM: Four Practical Revolutions in Management*, Productivity Press, Portland, Ore., 1993.
6. J. Karlsson, "Software Requirements Prioritizing," *Proc. 2nd IEEE Int'l Conf. Req. Eng.*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1996, pp. 110-116.
7. T.L. Saaty, *The Analytic Hierarchy Process*, McGraw-Hill, New York, 1980.
8. C. Potts, "Software Engineering Research Revisited," *IEEE Software*, Sept. 1993, pp. 19-28.
9. J. Karlsson, *Towards a Strategy for Software Requirements Selection*, Dept. of Computer and Information Science, Linköping Univ., Linköping, Sweden, Licentiate thesis *513*, 1995.
10. S.J. Andriole and P.A. Freeman, "Software Systems Engineering: The Case for a New Discipline," *Software Eng. J.*, May 1993, pp. 165-179.
11. B.W. Boehm and H. In, "Identifying Quality-Requirements Conflicts," *IEEE Software*, Mar. 1996, pp. 25-35.
12. T. Asada et al., "The Quantified Design Space," *Software Architecture*, M. Shaw and D. Garlan, eds., Prentice Hall, Englewood Cliffs, N.J., 1996, pp.116-128.

Joachim Karlsson is cofounder and managing director of Focal Point AB in Linköping, Sweden, and is affiliated with Linköping University. He consults, lectures, and conducts research on software process, software quality, and requirements engineering.

Karlsson received his MSc and Licentiate degree in computer science from Linköping University, Sweden. He is a member of IEEE.



Kevin Ryan is a professor of information technology at the University of Limerick in Limerick, Ireland. His main research interest is requirements engineering. He has lectured and done research on software topics in universities and in industry in Ireland, the US, Africa, and Sweden and has been involved in major Esprit projects on software methods and tools.

Ryan received his BA, BAI engineering, and PhD in computer science from Trinity College, Dublin. He is a member of the IEEE Computer Society, ACM, and the Irish and British computer societies.

Address questions about this article to Karlsson at Focal Point AB, Teknikringen 1E, 58330 Linköping, Sweden; joachim.karlsson@focalpoint.se; or Ryan at kevin.ryan@ul.ie.

Authorized licensed use limited to: Norges Teknisk-Naturvitenskapelige Universitet. Downloaded on August 19, 2009 at 06:20 from IEEE Xplore. Restrictions apply.