NECESSITY OF REQUIREMENTS PRIORITIZATION IN SOFTWARE ENGINEERING

[1]Kothuri Parashu Ramulu, [2]Dr.B.V. Ramana Murthy, [3]Dr. J. Malla Reddy

[1]Research Scholar, [2]Professor , [3]Professor
[1]Department of Computer Science,
[1]Rayalaseeema University, Kurnool, India

Abstract : This paper provides an overview of techniques for prioritization of requirements for software products. Prioritization is a crucial step towards making good decisions regarding product planning for single and multiple releases. Various aspects of functionality are considered, such as importance, risk, cost, etc. Prioritization decisions are made by stakeholders, including users, managers, developers, or their representatives. Methods are given how to combine individual prioritizations based on overall objectives and constraints. A range of different techniques and aspects are applied to an example to illustrate their use. Finally, limitations and shortcomings of current methods are pointed out, and open research questions in the area of requirements prioritization are discussed.

IndexTerms - **Requirements analysis, software product planning, Aspects of Prioritization, requirements prioritization, decision support.**

1. Introduction

In everyday life, we make many decisions, e.g. when buying a DVD-player, food, a telephone, etc. Often, we are not even conscious of making one. Usually, we do not have more than a couple of choices to consider, such as which brand of mustard to buy, or whether to take this bus or the next one. Even with just a couple of choices, decisions can be difficult to make. When having tens, hundreds or even thousands of alternatives, decision-making becomes much more difficult.

One of the keys to making the right decision is to prioritize between different alternatives. It is often not obvious which choice is better, because several aspects must be taken into consideration. For example, when buying a new car, it is relatively easy to make a choice based on speed alone (one only needs to evaluate which car is the fastest). When considering multiple aspects, such as price, safety, comfort, or luggage load, the choice becomes much harder. When developing software systems, similar trade-offs must be made. The functionality that is most important for the customers might not be as important when other aspects (e.g. price) are factored in. We need to develop the functionality that is most desired by the customers, as well as least risky, least costly, and so forth.

Prioritization helps to cope with these complex decision problems. This chapter provides a description of available techniques and methods, and how to approach a prioritization situation. The chapter is structured as follows: First, an overview of the area of prioritization is given (Section 2). This is followed by a presentation and discussion of different aspects that could be used when prioritizing (Section 3). Next, some prioritization techniques and characteristics are discussed (Section 4), followed by a discussion of different stakeholders situations that affect prioritization in Section 5. Section 6 discusses additional issues that arise when prioritizing software requirements and Section 7 provides an example of a prioritization. Section 8 discusses possible future research questions in the area. Finally, Section 9 summarizes the concept.

2. Requirements Prioritization

Complex decision-making situations are not unique to software engineering. Other disciplines, such as psychology, and organizational behavior have studied decision-making thoroughly [1]. Classical decision-making models have been mapped to various requirements engineering activities to show the similarities [1]. Chapter 12 in this book provides a comprehensive overview of decision-making and decision support in requirements engineering. Current chapter primarily focuses on requirements prioritization, an integral part of decision-making [49]. The intention is to describe the current body of knowledge in the requirements prioritization area.

The quality of a software product is often determined by the ability to satisfy the needs of the customers and users [7, 53]. Hence, eliciting (Chapter 2) and specifying (Chapter 3) the correct requirements and planning suitable releases with the right functionality is a major step towards the success of a project or product. If the wrong requirements are implemented and users resist using the product, it does not matter how solid the product is or how thoroughly it has been tested.

Most software projects have more candidate requirements than can be realized within the time and cost constraints. Prioritization helps to identify the most valuable requirements from this set by distinguishing the critical few from the trivial many. The process of prioritizing requirements provides support for the following activities (e.g. [32, 55, 57, 58]):

- for stakeholders to decide on the core requirements for the system.
- to plan and select an ordered, optimal set of software requirements for implementation in successive releases.
- to trade off desired project scope against sometimes conflicting constraints such as schedule, budget, resources, time to market, and quality.
- to balance the business benefit of each requirement against its cost.
- to balance implications of requirements on the software architecture and future evolution of the product and its associated cost.
- to select only a subset of the requirements and still produce a system that will satisfy the customer(s).
- to estimate expected customer satisfaction.
- to get a technical advantage and optimize market opportunity.
- to minimize rework and schedule slippage (plan stability).
- to handle contradictory requirements, focus the negotiation process, and resolve disagreements between stakeholders (more about this in Chapter 7).
- to establish relative importance of each requirement to provide the greatest value at the lowest cost.

The list above clearly shows the importance of prioritizing and deciding what requirements to include in a product. This is a strategic process since these decisions drive the development expenses and product revenue as well as making the difference between market gain and market loss [1]. Further, the result of prioritization might form the basis of product and marketing plans, as well as being a driving force during project planning. Ruhe *et al.* summarize this as: *"The challenge is to select the 'right' requirements out of a given superset of candidate requirements so that all the different key interests, technical constraints and preferences of the critical stakeholders are fulfilled and the overall business value of the product is maximized"* [48].

Of course, it is possible to rectify incorrect decisions later on via change management (more about change impact analysis in Chapter 6), but this can be very costly since it is significantly more expensive to correct problems later in the development process [5]. Frederick P. Brooks puts it in the following words: *"The hardest single part of building a software system is deciding precisely what to build. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later."* [10]. Hence, the most cost effective way of developing software is to find the optimal set of requirements early, and then to develop the software according to this set. To accomplish this, it is crucial to prioritize the requirements to enable selection of the optimal set.

Besides the obvious benefits presented above, prioritizing requirements can have other benefits. For example, it is possible to find requirements defects (e.g misjudged, incorrect and ambiguous requirements) since requirements are analyzed from a perspective that is different from that taken during reviews of requirements [33].

Some authors consider requirements prioritization easy [55], some regard it of medium difficulty [57], and some regard prioritization as one of the most complex activities in the requirements process, claiming that few software companies have effective and systematic methods for prioritizing requirements [40]. However, all these sources consider requirements prioritization a fundamental activity for project success. At the same time, some text books about requirements engineering (e.g [9, 47]) do not discuss requirements prioritization to any real extent.

There is no "right" requirements process and the way of handling requirements differs greatly between different domains and companies [1]. Further, requirements are typically vaguer early on and become more explicit as the understanding of the product grows [50]. These circumstances imply that there is no specific phase where prioritization is made, rather, it is performed throughout the development process [13, 38]. Hence, prioritization is an iterative process and might be performed at different abstraction levels and with different information in different phases during the software lifecycle.

Prioritization techniques can roughly be divided into two categories: methods and negotiation approaches. The methods are based on quantitatively assigning values to different aspects of requirements while negotiation approaches focus on giving priorities to requirements by reaching agreement between different stake holders [39]. Further, negotiation approaches are

based on subjective measures and are commonly used when analyses are contextual and when decision variables are strongly interrelated. Quantitative methods make it easier to aggregate different decision variables into an overall assessment and lead to faster decisions [15, 50]. In addition, one must be mindful of the social nature of prioritization. There is more to requirements prioritization than simply asking stakeholders about priorities. Stakeholders play roles and should act according to the goals of that role, but they are also individuals with personalities and personal agendas. Additionally, many organizational issues like power etc. need to be taken into account. Ignoring such issues can raise the risk level for a project.

3. Aspects of Prioritization

Requirements can be prioritized taking many different aspects into account. An aspect is a property or attribute of a project and its requirements that can be used to prioritize requirements. Common aspects are importance, penalty, cost, time, and risk. When prioritizing requirements based on a single aspect, it is easy to de-cide which one is most desirable (recall the example about the speed of a car). When involving other aspects, such as cost, customers can change their mind and high priority requirements may turn out to be less important if they are very ex-pensive to satisfy [36]. Often, the aspects interact and changes in one aspect could result in an impact on another aspect [50]. Hence, it is essential to know what ef-fects such conflicts may have, and it is vital to not only consider importance when prioritizing requirements but also other aspects affecting software development and satisfaction with the resulting product. Several aspects can be prioritized, and it may not be practical to consider them all. Which ones to consider depend on the specific situation, and a few examples of aspects suitable for software projects are described below. Aspects are usually evaluated by stakeholders in a project (managers, users, developers, etc.)

3.1. Importance

When prioritizing importance, the stakeholders should prioritize which require-ments are most important for the system. However, importance could be an extremely multifaceted concept since it depends very much on which perspective the stakeholder has. Importance could for example be urgency of implementation, importance of a requirement for the product architecture, strategic importance for the company, etc. [38]. Consequently, it is essential to specify which kind of importance the stakeholders should prioritize in each case.

3.2. Penalty

It is possible to evaluate the penalty that is introduced if a requirement is not fulfilled [57]. Penalty is not just the opposite of importance. For example, failing to conform to a standard could incur a high penalty even if it is of low importance for the customer (i.e. the customer does not get excited if the requirement is fulfilled). The same goes for implicit requirements that users take for granted, and whose absence could make the product unsuitable for the market.

3.3. Cost

The implementation cost is usually estimated by the developing organization. Measures that influence cost include: complexity of the requirement, the ability to reuse existing code, the amount of testing and documentation needed, etc. [57]. Cost is often expressed in terms of staff hours (effort) since the main cost in soft-ware development is often primarily related to the number of hours spent. Cost could be prioritized by using any of the techniques presented , but also by simply estimating the actual cost on an absolute or normalized scale.

3.4 Time

As can be seen in the section above, cost in software development is often related to number of staff hours. However, time (i.e. lead time) is influenced by many other factors such as degree of parallelism in development, training needs, need to develop support infrastructure, complete industry standards, etc. [57].

3.5 Risk

Every project carries some amount of risk. In project management, risk management is used to cope with both internal (technical and market risks) and external risks (e.g. regulations, suppliers). Both likelihood and impact must be considered when determining the level of risk of an item or activity [44]. Risk management can also be used when planning requirements into products and releases by identifying risks that are likely to cause difficulties during development [41, 57]. Such risks could for example include performance risks, process risks, schedule risks etc. [55]. Based on the estimated risk likelihood and risk impact for each requirement [1], it is possible to calculate the risk level of a project.

3.6 Volatility

Volatility of requirements is considered a risk factor and is sometimes handled as part of the risk aspect [41]. Others think that volatility should be analyzed separately and that volatility of requirements should be taken into account separately in the prioritization process [36]. The reasons for requirements volatility vary, for example: the market changes, business requirements change, legislative changes occur, users change, or requirements become more clear during the software life cycle [18, 50]. Irrespective of the reason, volatile requirements affect the stability and planning of a project, and presumably increase the costs since changes during development increase the cost of a project. Further, the cost of a project might increase because developers have to select an architecture suited to change if volatility is known to be an issue [36].

3.7. Other Aspects

The above list of aspects has been considered important in the literature but it is by no means exhaustive. Examples of other aspects are: financial benefit, strategic benefit, competitors, competence/resources, release theme, ability to sell, etc. For a company, we suggest that stakeholders develop a list of important aspects to use in the decision-making. It is important that the stakeholders have the same interpretation of the aspects as well as of the requirements. Studies have shown that it is hard to interpret the results if no guidelines about the true meaning of an aspect are present [37, 38].

3.8. Combining Different Aspects

In practice, it is important to consider multiple aspects before deciding if a requirement should be implemented directly, later, or not at all. For example, in the Cost- Value approach, both value (importance) and cost are prioritized to imple-ment those requirements that give most value for the money [30]. The Planning Game (from XP) uses a similar approach when importance, effort (cost), and risks are prioritized [2]. Further, importance and stability (volatility) are suggested as aspects that should be used when prioritizing while others suggest that dependencies also must be considered [12, 36] (more about dependencies in Chapter 5). In Wiegers' approach, the relative value (importance) is divided by the relative cost and the relative risk in order to determine the requirements that have the most fa-vorable balance of value, cost, and risk [57]. This approach further allows differ-ent weights for different aspects in order to favor the most important aspect (in the specific situation).

There are many alternatives of combining different aspects. Which aspects to consider depend very much on the specific situation and it is important to know about possible aspects and how to combine them efficiently to suit the case at hand.

4. Prioritization Techniques

The purpose of any prioritization is to assign values to distinct prioritization ob-jects that allow establishment of a relative order between the objects in the set. In our case, the objects are the requirements to prioritize. The prioritization can be done with various measurement scales and types. The least powerful prioritization scale is the ordinal scale, where the requirements are ordered so that it is possible to see which requirements are more important than others, but not how much more important. The ratio scale is more powerful since it is possible to quantify how much more important one requirement is than another (the scale often ranges from 0 - 100 percent). An even more powerful scale is the absolute scale, which can be used in situations where an absolute number can be assigned (e.g. number of hours). With higher levels of measurement, more sophisticated evaluations and calculations become possible [20].

Below, a number of different prioritization techniques are presented. Some techniques assume that each requirement is associated with a priority, and others group requirements by priority level. When examples are given, importance is used as the aspect to prioritize even though other aspects can be evaluated with each of the techniques. It should be noted that the presented techniques focus specifically on prioritization. Numerous *methods* exist that use these prioritization techniques within a larger trade- off and decision making framework (e.g. EVOLVE [24], Cost-Value [30] and Quantitative Win-Win [48]).

4.1. Analytical Hierarchy Process (AHP)

The Analytic Hierarchy Process (AHP) is a systematic decision-making method that has been adapted for prioritization of software requirements [45, 51]. It is conducted by comparing all possible pairs of hierarchically classified require-ments, in order to determine which has higher priority, and to what extent (usually on a scale from one to nine where one represents equal importance and nine represents absolutely more important) . The total number of comparisons to perform with AHP are $n \times (n-1)/2$ (where n is the number of requirements) at each hierarchy level, which results in a dramatic increase in the number of comparisons as the number of requirements increases. Studies have shown that AHP is not suit-able for large numbers of requirements [39, 42]. Researchers have tried to find ways to decrease the number of comparisons (e.g. [26, 54]) and variants of the technique have been found to reduce the number of comparisons by as much as 75 percent [31].

In its original form, the redundancy of the pair-wise comparisons allows a consistency check where judgment errors can be identified and a consistency ratio can be calculated. When reducing the number of comparisons, the number of redun-dant comparisons are also reduced, and consequently the ability to identify incon-sistent judgments [33]. When using other techniques (explained below) a consis-tency ratio is not necessary since all requirements are directly compared to each other and consistency is always ensured. Some studies indicate that persons who prioritize with AHP tend to mistrust the results since control is lost when only comparing the requirements pair-wise [34, 39]. The result from a prioritization with AHP is a weighted list on a ratio scale. More detailed information about AHP can be found in [30], [51] and [52].

4.2. Cumulative Voting, the 100-Dollar Test

The 100-dollar test is a very straightforward prioritization technique where the stakeholders are given 100 imaginary units (money, hours, etc.) to distribute be-tween the requirements [37]. The result of the prioritization is presented on a ratio scale. A problem with this technique arises when there are too many requirements to prioritize. For example, if you have 25 requirements, there are on average four points to distribute for each requirement. Regnell *et al.* faced this problem when there were 17 groups of requirements to prioritize [45]. In the study, they used a fictitious amount of $100,000 to have more freedom in the prioritizations. The subjects in the study were positive about the technique, indicating the possibility to use amounts other than 100 units (e.g. 1,000, 10,000 or 1 000,000). Another possible problem with the 100-dollar test (especially when there are many re-quirements) is that the person performing the

prioritization miscalculates and the points do not add up to 100 [3]. This can be prevented by using a tool that keeps count of how many points have been used.

One should only perform the prioritization once one the same set of requirements, since the stakeholders might bias their evaluation the second time around if they do not get one of their favorite requirements as a top priority. In such a situa-tion, stakeholders could put all their money on one requirement, which might in-fluence the result heavily. Similarly, some clever stakeholders might put all their money on a favorite requirement that others do not prioritize as highly (e.g. Mac compatibility) while not giving money to requirements that will get much money anyway (e.g. response time). The solution could be to limit the amount spent on individual requirements [37]. However, the risk with such an approach is that stakeholders may be forced to not prioritize according to their actual priorities.

4.3. Numerical Assignment (Grouping)

Numerical assignment is the most common prioritization technique and is suggested both in RFC 2119 [8] and IEEE Std. 830-1998 [29]. The approach is based on grouping requirements into different priority groups. The number of groups can vary, but in practice, three groups are very common (e.g. [37, 55]). When using numerical assignment, it is important that each group represents something that the stakeholders can relate to (e.g. critical, standard, optional), for a reliable classification. Using relative terms such as high, medium, and low will confuse the stakeholders [57]. This seems to be especially important when there are stake-holders with different views of what high, medium and low means. A clear definition of what a group really means minimizes such problems.

A further potential problem is that stakeholders tend to think that everything is critical [36, 55]. If customers prioritize themselves, using three groups; *critical*, *standard*, and *optional*, they will most likely consider 85 percent of the requirements as critical, 10 percent as standard, and 5 percent as optional [4, 57]. One idea is to put restrictions on the allowed number of requirements in each group (e.g. not less than 25 percent of the requirements in each group) [34]. However, one problem with this approach is that the usefulness of the priorities diminishes because the stakeholders are forced to divide requirements into certain groups[32].However, no empirical evidence of good or bad results with such restrictions exists. The result of numerical assignment is requirements prioritized on an ordinal scale. However, the requirements in each group have the same priority, which means that each requirement does not get a unique priority.

4.4. Ranking

As in numerical assignment, ranking is based on an ordinal scale but the requirements are ranked without ties in rank. This means that the most important requirement is ranked 1 and the least important is ranked *n* (for *n* requirements). Each requirement has a unique rank (in comparison to numerical assignment) but it is not possible to see the relative difference between the ranked items (as in AHP or the 100-dollar test). The list of ranked requirements could be obtained in a variety of ways, as for example by using the bubble sort or binary search tree algorithms [33]. Independently of sorting algorithm, ranking seems to be more suitable for

a single stakeholder because it might be difficult to align several different stakeholders' views. Nevertheless, it is possible to combine the different views by taking the mean priority of each requirement but this might result in ties for requirements which this method wants to avoid.

6. Future Research in the Area of Requirements Prioritization

Requirements engineering is a field with much research activity. One journal, several workshops, and one large annual international conference are devoted to requirements engineering. Nevertheless, the existing work in the area of requirements prioritization is limited even though the need for prioritizing software requirements is acknowledged in the research literature [32]. Especially, few empirical validations of different prioritization techniques and methods exist. Instead, it is common that new techniques and methods are introduced and they seem to work well, but the scalability of the approach has not been tested (e.g. [48]). How-ever, there exist some studies that have evaluated different prioritization techniques (e.g. [33, 34]). Unfortunately, such empirical evaluations most often focus on toy systems with a few requirements (seldom more than 20). This is not really providing any evidence of whether one technique is better than another even though some preliminary evidence could be found. One of the few industry studies, for example, found that AHP was not usable with more than 20 requirements since the number of comparisons became too many for the practitioners [39]. Hence, more studies are needed when prioritization methods are used in industry.

A further question that seldom is addressed in requirements prioritization re-search is the question of how much sophistication is actually needed. Many techniques and methods are developed and they become more and more complex with the goal to provide more help for practitioners but the results are seldom used in industry. Instead, professionals use simple methods such as numerical assignment. Practitioners live in a different environment than experimental subjects (often students) and are more limited by time and cost constraints [4]. Hence, an important question to answer is how much sophistication (and thereby complexity) is actu-ally necessary and desirable by practitioners?

The above issues lead to another open question about when a technique or method is suitable. Existing empirical studies seldom discuss factors such as company size, time-to -market limitations, number of stakeholders, domain, etc. In-stead, focus is on whether a technique or method is better than another one. A more sound approach would be to test different approaches in various environ-ments to get some understanding when different prioritization techniques, aspects, etc. are suitable. In [21] a framework for evaluating pair programming is suggested and independent (e.g. technique), dependent (e.g. quality), and context variables (e.g. type of task) are proposed for evaluating programming techniques. A similar framework for requirements prioritization would be beneficial.

Another important question in the area of requirements prioritization concerns dependencies between requirements. Nevertheless, the impact of dependencies can be tremendous. For example, prioritization techniques (such as AHP) assume that requirements are independent even though we know that they seldom are [46]. We need to find better ways to handle dependencies in an efficient way.

As could be seen in Section 4.6.3, functional and non- functional requirements are very different even though they have a serious impact on each other. Prioritizing these two entirely together or separately might not be the best solution. Approaches where prioritizations of functional and non-functional could be combined in an efficient way are necessary. Different methods that seem suitable for prioritizing non-functional requirements are available (e.g. Conjoint Analysis [22], and Quality Grid [36]) and it would be interesting to evaluate these empirically in industrial settings. Further, finding ways to combine such approaches with approaches more directed to functional requirements would be a challenge.

7. Summary

This paper has presented a number of techniques, aspects, and other issues that should be thought of when performing prioritizations. These different parts together form a basis for systematically prioritizing requirements during software development. The result of prioritizations suggests which requirements should be implemented, and in which release. Hence, the techniques could be a valuable help for companies to get an understanding of what is important and what is not for a project or a product. As with all evaluation methods, the results should be interpreted and possibly adjusted by knowledgeable decision-makers rather than simply accepted as a final decision.

8. Acknowledgement

We thank all our friends and colleagues for their support.

References

1. Aurum A, Wohlin C (2003) The Fundamental Nature of Requirements Engineering Activities as a Decision-Making Process. Information and Software Technology 45(14): 945-954
2. Beck K (1999) Extreme Programming Explained. Addison-Wesley, Upper Saddle River
3. Berander P, Wohlin C (2004) Differences in Views between Development Roles in Software Process Improvement – A Quantitative Comparison. Proceedings of the 8th International Conference on Empirical Assessment in Software Engineering (EASE 2004). IEE, Stevenage, pp 57-66
4. Berander P (2004) Using Students as Subjects in Requirements Prioritization. Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE'04). IEEE Computer Society, Los Alamitos, pp 167-176
5. Boehm BW (1981) Software Engineering Economics. Prentice Hall, Englewood Cliffs
6. Boehm BW, Ross R (1989) Theory-W Software Project Management: Principles and Examples. IEEE Transactions on Software Engineering 15(7):902-916
7. Bergman B, Klefsjö B (2003) Quality from Customer Needs to Customer Satisfaction. Published by Studentlitteratur AB, Lund
8. Bradner S (1997) RFC 2119. Internet <http://www.ietf.org/rfc/rfc2119.txt> (24 No-vember 2004)

9. Bray IK (2002) An Introduction to Requirements Engineering. Pearson Education, London
10. Brooks FP (1995) The Mythical Man-Month: Essays on Software Engineering. Addi-son-Wesley Longman, Boston
11. Carlshamre P (2001) A Usability Perspective on Requirements Engineering – From Methodology to Product Development. Ph.D. thesis, Linköping Institute of Technol-ogy
12. Carlshamre P, Sandahl K, Lindvall M, Regnell B, Natt och Dag J (2001) An Industrial Survey of Requirements Interdependencies in Software Release Planning. Proceedings of the Fifth IEEE International Symposium on Requirements Engineering (RE'01). IEEE Computer Society, Los Alamitos, pp 84-91
13. Carlshamre P (2002) Release Planning in Market-Driven Software Product Develop-ment: Provoking an Understanding. Reguirements Engineering 7(3):139-151
14. Clements P, Northrop L (2002) Software Product Lines – Practices and Patterns. Addi-son-Wesley, Upper Saddle River
15. Colombo E, Francalanci C (2004) Selecting CRM Packages Based on Architectural, Functional, and Cost Requirements: Empirical Validation of a Hierarchical Ranking Model. Requirements Engineering 9(3):186-203
16. Dahlstedt Å, Persson A (2003) Requirements Interdependencies – Moulding the State of Research into a Research Agenda. Proceedings of the Ninth International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ '03). Univer-sität Duisburg-Essen, Essen, pp. 71-80
17. Davis AM (2003) The Art of Requirements Triage. IEEE Computer 36(3):42-49
18. Ecklund EF, Delcambre LML, Freiling MJ (1996) Change Cases: Use Cases that Iden-tify Future Requirements. Proceedings of the 11th ACM SIGPLAN Conference on Ob-ject-Oriented Programming, Systems, Languages, and Applications (OOPSLA '96). ACM, USA, pp. 342-358
19. Feather MS, Menzies T (2002) Converging on the Optimal Attainment of Require-ments. Proceedings of the IEEE Joint International Conference on Requirements Engi-neering (RE'02). IEEE Computer Society, Los Alamitos, pp. 263-270
20. Fenton, NE, Pfleeger SL (1997) Software Metrics – A Rigorous and Practical Ap-proach, 2nd Edition. PWS Publishing Company, Boston
21. Gallis H, Arisholm E, Dybå T (2003) An Initial Framework for Research on Pair Pro-gramming. Proceedings of the 2003 International Symposium on Empirical Software Engineering (ISESE'03). IEEE Computer Society, Los Alamitos, pp.132-142
22. Giesen, J, Völker A (2002) Requirements Interdependencies and Stakeholders Prefer-ences. Proceedings of the IEEE Joint International Conference on Requirements Engi-neering (RE'02). IEEE Computer Society, Los Alamitos, pp 206-209
23. Gorschek T (2004) Software Process Assessment & Improvement in Industrial Re-quirements Engineering. Licentiate Thesis, Blekinge Institute of Technology
24. Greer D, Ruhe G (2004) Software Release Planning: an Evolutionary and Iterative Ap-proach. Information and Software Technology 46(4):243-253
25. Grudin J, Pruitt J (2002) Personas, Participatory Design and Product Development: An Infrastructure for Engagement. Participation and Design Conference (PDC2002). Computer Professionals for Social Responsibility, Palo Alto, pp. 144-161

26. Harker PT (1987) Incomplete Pairwise Comparisons in the Analytic Hierarchy Proc-ess. Mathematical Modelling 9(11):837-848
27. Hill N, Brierly J, MacDougall R (1999) How to Measure Customer Satisfaction. Gower Publishing, Hampshire
28. Humphrey WS (1989) Managing the Software Process. Addison-Wesley, USA
29. IEEE Std 830-1998 (1998) IEEE Recommended Practice for Software Requirements Specifications. IEEE Computer Society, Los Alamitos
30. Karlsson J, Ryan K (1997) A Cost-Value Approach for Prioritizing Requirements. IEEE Software 14(5):67-74
31. Karlsson J, Olsson S, Ryan K (1997) Improved Practical Support for Large-Scale Requirements Prioritizing. Requirements Engineering 2(1):51-60
32. Karlsson J (1998) A Systematic Approach for Prioritizing Software Requirements. Ph.D. Thesis, Linköping Institute of Technology
33. Karlsson J, Wohlin C, Regnell B (1998) An Evaluation of Methods for Prioritizing Software Requirements. Information and Software Technology 39(14-15):939-947
34. Karlsson L, Berander P, Regnell B, Wohlin C (2004) Requirements Prioritisation: An Experiment on Exhaustive Pair-Wise Comparisons versus Planning Game Partitioning. Proceedings of the 8th International Conference on Empirical Assessment in Software Engineering (EASE 2004). IEE, Stevenage, pp. 145-154
35. Kotler P, Armstron G, Saunders J, Wong V (2002) Principles of Marketing, 3rd European Edition. Pearson Education, Essex
36. Lausen S (2002) Software Requirements – Styles and Techniques. Pearson Education, Essex
37. Leffingwell D, Widrig D (2000) Managing Software Requirements – A Unified Approach. Addison-Wesley, Upper Saddle River
38. Lehtola L, Kauppinen M, Kujala S (2004) Requirements Prioritization Challenges in Practice. Proceedings of 5th International Conference on Product Focused Software Process Improvement, Lecture Notes in Computer Science (vol. 3009), Springer-Verlag, Heidelberg, pp. 497-508
39. Lehtola L, Kauppinen M (2004) Empirical Evaluation of Two Requirements Prioritization Methods in Product Development Projects, Proceedings of the European Software Process Improvement Conference (EuroSPI 2004), Springer-Verlag, Berlin Heidel-berg, pp. 161-170
40. Lubars M, Potts C, Richter C (1993) A Review of the State of Practice in Require-ments Modeling. Proceedings of IEEE International Symposium on Requirements Engineering, IEEE Computer Society, Los Alamitos, pp. 2-14
41. Maciaszek LA (2001) Requirements Analysis and System Design – Developing Information Systems with UML. Addison Wesley, London
42. Maiden NAM, Ncube C (1998) Acquiring COTS Software Selection Requirements. IEEE Software 15(2):46-56
43. Moore G (1991) Crossing the Chasm. HarperCollins, New York
44. Nicholas JM (2001) Project Management for Business and Technology – Principles and Practice, 2nd Edition. Prentice Hall, Upper Saddle River

45. Regnell B, Höst M, Natt och Dag J, Beremark P, Hjelm T (2001) An Industrial Case Study on Distributed Prioritisation in Market-Driven Requirements Engineering for Packaged Software. Requirements Engineering 6(1):51-62
46. Regnell B, Paech B, Aurum A, Wohlin C, Dutoit A, Natt och Dag J (2001) Requirements Mean Decisions! – Research Issues for Understanding and Supporting Decision-Making in Requirements Engineering. First Swedish Conference on Software Engineering Research and Practise (SERP'01): Proceedings, Blekinge Institute of Technology, Ronneby, pp. 49-52
47. Robertson S, Robertson J (1999) Mastering the Requirements Process. ACM Press, London
48. Ruhe G, Eberlein A, Pfahl D (2002) Quantitative WinWin – A New Method for Decision Support in Requirements Negotiation. Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'02), ACM Press, New York, pp. 159-166
49. Ruhe G (2003) Software Engineering Decision Support - A New Paradigm for Learn-ing Software Organizations. Advances in Learning Software Organization, Lecture Notes in Computer Science (vol. 2640), Springer-Verlag, pp. 104-115
50. Ruhe G, Eberlein A, Pfahl D (2003) Trade-off Analysis for Requirements Selection. International Journal of Software Engineering and Knowledge Engineering 13(4): 345-366
51. Saaty TL (1980) The Analytic Hierarchy Process. McGraw-Hill, New York
52. Saaty TL, Vargas LG (2001) Models, Methods, Concepts & Applications of the Analytic Hierarchy Process. Kluwer Academic Publishers, Norwell
53. Schulmeyer GG, McManus JI (1999) Handbook of Software Quality Assurance, 3rd Edition. Prentice Hall, Upper Saddle River
54. Shen Y, Hoerl AE, McConnell W (1992) An Incomplete Design in the Analytical Hierarchy Process. Mathematical Computer Modelling 16(5):121-129
55. Sommerville I, Sawyer P (1997) Requirements Engineering – A Good Practice Guide. John Wiley and Sons, Chichester
56. Sommerville I (2001) Software Engineering, 6th Edition. Pearson Education, London
57. Wiegers K (1999) Software Requirements. Microsoft Press, Redmond
58. Yeh AC (1992) REQUirements Engineering Support Technique (REQUEST) – A Market Driven Requirements Management Process. Proceedings of Second Sympo-sium of Quality Software Development Tools. IEEE Computer Society, Piscataway, pp. 211-223