4 Requirements Prioritization

Patrik Berander and Anneliese Andrews

Abstract: This chapter 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 for combining 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.

Keywords: Requirements analysis, Software product planning, Requirements prioritization, Decision support, Trade offs.

4.1 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 (Sect. 4.2). This is followed by a presentation and discussion of different aspects that could be used when prioritizing (Sect. 4.3). Next, some prioritization techniques and characteristics are discussed (Sect. 4.4), followed by a discussion of different stakeholders' situations that affect prioritiza-

tion in Sect. 4.5. Section 4.6 discusses additional issues that arise when prioritizing software requirements and Section 4.7 provides an example of a prioritization. Section 4.8 discusses possible future research questions in the area. Finally, Sect. 4.9 summarizes the chapter.

4.2 What is 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 (Chap. 2) and specifying (Chap. 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 [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 Chap. 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 Chap. 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 [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 (more about this in Sect. 4.6.2) [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 stakeholders [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. Negotiation and goal modeling are described in detail in Chaps. 7 and 9, respectively, while this chapter focuses primarily on quantitative methods for prioritizing requirements.

4.3 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 decide 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 expensive 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 effects 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.)

4.3.1 Importance

When prioritizing importance, the stakeholders should prioritize which requirements 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.

4.3.2 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.

4.3.3 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 software development is often primarily related to the number of hours spent. Cost (as well as time, cf. Sect. 4.3.4.) could be prioritized by using any of the techniques presented in Sect. 4.4, but also by simply estimating the actual cost on an absolute or normalized scale.

4.3.4 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].

4.3.5 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.

4.3.6 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 clearer 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 (see more about this issue in Chap. 6). 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].

4.3.7 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].

4.3.8 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 implement those requirements that give most value for the money [30]. The Planning Game (PG) from eXtreme Programming (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 Chap. 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 favorable balance of value, cost, and risk [57]. This approach further allows different 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 depends 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.

4.4 Prioritization Techniques

The purpose of any prioritization is to assign values to distinct prioritization objects 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].

4.4.1 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 requirements, 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 suitable 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 redundant comparisons are also reduced, and consequently the ability to identify inconsistent judgments [33]. When using other techniques (explained below) a consistency 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].

4.4.2 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 between 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 requirements) 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 situation, stakeholders could put all their money on one requirement, which might influence 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.

4.4.3 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 [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 stakeholders 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.

4.4.4 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.

4.4.5 Top-Ten Requirements

In the top-ten requirements approach, the stakeholders pick their top-ten requirements (from a larger set) without assigning an internal order between the requirements. This makes the approach especially suitable for multiple stakeholders of equal importance [36]. The reason to not prioritize further is that it might create unnecessary conflict when some stakeholders get support for their top priority and others only for their third priority. One could assume that conflicts might arise anyway if, for example, one customer gets three top-ten requirements into the product while another gets six top-ten requirements into the product. However, it is important to not just take an average across all stakeholders since it might lead to some stakeholders not getting any of their top requirements [36]. Instead, it is crucial that some essential requirements are satisfied for each stakeholder. This could obviously result in a situation that dissatisfies all customers instead of satisfying a few customers completely. The main challenge in this technique is to balance these issues.

4.4.6 Which Prioritization Technique to Choose

Table 4.1 summarizes the presented prioritization techniques, based on measurement scale, granularity of analysis, and level of sophistication of the technique.

Table 4.1 Summary of presented technique

Technique	Scale	Granularity	Sophistication
AHP	Ratio	Fine	Very Complex
Hundred-dollar test	Ratio	Fine	Complex
Ranking	Ordinal	Medium	Easy
Numerical Assignment	Ordinal	Coarse	Very Easy
Top-ten	-	Extremely Coarse	Extremely Easy

A general advice is to use the simplest appropriate prioritization technique and use more sophisticated ones when a more sensitive analysis is needed for resolving disagreements or to support the most critical decisions [42]. As more sophisticated techniques generally are more time consuming, the simplest possible technique ensures cost effective decisions. The trade-off is to decide exactly how "quick and dirty" the approach can be without letting the quality of the decisions suffer. It should also be noted that there exist several commercial tools that facilitate the use of more sophisticated techniques (e.g. AHP) and that it is possible to construct simple home-made tools (e.g. in spreadsheets) to facilitate the use of different prioritization techniques.

4.4.7 Combining Different Techniques

The techniques in Table 4.1 represent the most commonly referenced quantitative prioritization techniques. It is possible to combine some of them to make prioritization easier or more efficient. Some combinations of the above techniques exist and probably the best known example is Planning Game (PG) in eXtreme Programming (XP) [2] (more about agile methods in requirements engineering in Chap. 14). In PG, numerical assignment and ranking are combined by first dividing the different requirements into priority groups and then ranking requirements within each group [34]. Requirements triage is an approach where parallels are drawn to medical treatment at hospitals [17]. Medical personnel divide victims into three categories: those that will die whether treated or not, those who will resume normal lives whether treated or not, and those for whom medical treatment may make a significant difference. In requirements prioritization, there are requirements that must be in the product (e.g. platform requirements), requirements that the product clearly need not satisfy (e.g. very optional requirements), and requirements that need more attention. This means that the requirements are assigned to one of three groups (numerical assignment) and requirements that need more attention are prioritized by any of the other techniques (AHP, ranking, 100 points etc.). In this approach, not all requirements must be prioritized by a more sophisticated technique, which decreases the effort.

The two examples above show that it is possible to combine different techniques for higher efficiency or to make the process easier. Which method or combination of methods is suitable often depends on the individual project.

4.5 Involved Stakeholders in the Prioritization Process

In Chap. 13, market-driven software development is discussed and similarities and differences between market-driven and bespoke software development are presented. As can be seen in Chap. 13, similarities and differences also apply when prioritizing software requirements. In a bespoke project, only one or a few stakeholders must be taken into consideration while everyone in the whole world might serve as potential customers in market-driven development. Table 4.2 outlines some of the differences between market-driven and bespoke development that affects requirements prioritization.

Table 4.2 Differences between market-driven and bespoke development [11]

Facet	Bespoke Development	Market-driven Development
Main stakeholder	Customer organization	Developing organization
Users	Known or identifiable	Unknown, may not exist until product is on market
Distance to users	Usually small	Usually large
Requirements Conception	Elicited, analyzed, validated	Invented (by market pull or technology push)
Lifecycle	One release, then maintenance	Several releases as long as there is a market demand
Specific RE issues	Elicitation, modeling, validation, conflict resolution	Steady stream of requirements, prioritization, cost estimating, release planning
Primary goal	Compliance to specification	Time-to-market
Measure of success	Satisfaction, acceptance	Sales, market share

As can be seen in Table 4.2, there are large differences between these two extremes and different projects have to consider different ways to handle, and hence prioritize, requirements. Table 4.2 shows the two extremes in software development; a real case probably falls somewhere in between. For example, it is possible that a company delivers for a market, but the market is limited to a small number of customers (e.g. telecommunication systems are only bought by telephone operators). The discussion here focuses on three different "general" scenarios: one customer, a number of "known" customers, and a mass-market.

4.5.1 One Customer

In a one customer situation, there is only one customer's priorities that need to be considered (from the customer/user perspective). Many of the present software

development processes are based on one customer and assume that this customer is available throughout the project [11]. For example, eXtreme Programming has an "on-site customer" as one of the core practices (the focus is on having one customer even though this customer could represent a market) [2]. One important issue to consider when having a one-customer situation is that the customer and the end-user(s) are not always the same. In this case, the person who prioritizes and the persons who will use the system may not have the same priorities [24]. Such situations are of course undesirable since it may result in reduced use of the product. In this case, it would be better to involve the end-users in prioritizing the requirements since they are the ones who know what they need. For example, if the customer is an employer, and the user is an employee of the company buying the product, this may result in conflicts. It is possible to imagine features that are desirable to an employer, but not an employee.

4.5.2 Several Known Customers

When having several customers, the issue of prioritization becomes more difficult since the customers may have conflicting viewpoints and preferences [1]. This introduces the challenge of drawing these different customer views together [38]. The ultimate goal in these situations is to create win-win conditions and make every stakeholder a "winner" [6]. If one perspective is neglected the system might be seen as a failure by one or several of the stakeholders [1]. Hence, it is of tremendous importance that all stakeholders are involved in this process since the success of the product ultimately is decided in this step. A discussion on how to make trade-offs between different stakeholders is provided in Sect. 4.5.5.

4.5.3 Mass-Market

When developing for a mass-market, it is not possible to get all customers to prioritize. When eliciting information for prioritization in a mass-market situation, different sources exist [35]: internal records (e.g. shipments, sales records), marketing intelligence (e.g. information from sales force, scientists), competitor intelligence (e.g. information about competitors' strategies, benchmarking competitors' products) and marketing research (e.g. surveys, focus groups). When conducting marketing research, the sample must be representative for the intended market segment (group of consumers with similar needs) [35]. For example, if developing products for large companies, it is meaningless to involve small companies in the focus groups or the surveys. Hence, it is very important to decide which market segments should be the focus of the product before performing the prioritization.

The result from a prioritization for a mass-market product could provide a good base for analyzing which requirements are high priorities for all different market segments. By using this information, it is possible to identify which parts of a system should be common for all market segments and which parts should be specifi-

cally developed for specific market segments. This way of dealing with requirements is valuable when developing software product lines [14].

One way of dealing with the problem that all possible users are not known or accessible is to use the concept of "personas" that originated in marketing and has been used in system design [25]. These personas are fictional persons, representing market segments. They have names, occupations, possessions, age, gender, socioeconomic status, etc. They are based on and inspired by real people that are supposed to use the developed product. This information is gathered from ethnographies, market research, usability studies, interviews, observations, and so forth. The intention is to help the developing organization focus the attention on personas that the system is and is not designed for, and to give an understanding of these target personas. Further, personas enhance engagement and reality by providing fictional users of the system. The developing organization can use the personas in decision-making (and prioritization) by asking questions like: Why are we building this feature (requirement)? Why are we building it like this? When having such explicit but fictitious users of the system, the organization can get an understanding of which choices the personas would make in different situations.

4.5.4 Stakeholders Represented in the Prioritization

Since requirements can be prioritized from several different aspects, different roles must also be involved in the prioritization process to get the correct views (e.g. product managers prioritize strategic importance and project managers prioritize risks). At least three perspectives should always be represented: customers, developers, and financial representatives [17]. Each of these stakeholders provides vital information that the other two may neglect or are unable to produce since customers care about the user/customer value, developers know about the technical difficulties, and financial representatives know and care for budgetary constraints and risks [17]. Nevertheless, it is of course suitable to involve all perspectives (beside these three) that have a stake in the project or product.

4.5.5 Trade-Off between Different Stakeholders

In both market-driven and bespoke projects, there can be several different stakeholders with different priorities and expectations of the system. How to make trade-offs between several stakeholders with different priorities is an issue that is commonly mentioned as a problem by product managers in software organizations. First, this could be a problem when having one or a few very strong stakeholders since their wishes are often hard to neglect (i.e. when the big customer says jump, the company jumps). Second, "squeaky wheel" customers often get what they want [38, 58].

In such situations, it is important to have a structured way of handling different stakeholders. Regnell et al. adjust the influence of each stakeholder by prioritize for different aspects [45]. This can be done by weighting market segments based

on for example: revenue last year, profit last release, size of total market segment, number of potential customers, etc. The weighting aspect depend on the strategy most suitable in the current market phase ([43], cited in [45]). Priorities are then used to weigh each stakeholder in the prioritization process. This approach is also possible when dealing with specific stakeholders even though the aspects on which the priorities are based might be different. The weighting of the stakeholders could be performed in the same way as ordinary prioritization, and the techniques described in Sect. 4.4 could be used to provide the weights (preferably the techniques based on a ratio scale since these will provide distances of importance between the stakeholders).

4.6 Using Requirements Prioritization

Requirements prioritization needs to consider several different aspects, techniques, and stakeholder situations. This section presents additional issues to consider and ways of dealing with such issues.

4.6.1 Abstraction Level

Requirements are commonly represented at different levels of abstraction [23], which causes problems when prioritizing requirements. One reason is that requirements on higher abstraction levels tend to get higher priority in pair-wise comparisons [39]. For example, if prioritizing requirements in a car, a lamp in the dashboard cannot be compared with having a luggage boot. Most customers would probably prefer a luggage boot over a lamp in the dashboard but if one had to compare a lamp in the luggage boot and a lamp in the dashboard, the lamp in the dashboard might have higher priority. Hence, it is really important that the requirements are not mixed at different abstraction levels [57].

Deciding on the level of abstraction can be difficult and depend very much on the number of requirements and their complexity. With a small number of requirements, it might be possible to prioritize the requirements at a low level of abstraction while it might be a good idea to start with requirements at a high level and prioritize lower levels within the higher levels later when having many requirements to prioritize [57]. AHP supports this approach of decomposing requirements into different hierarchical levels in order to decrease the number of comparisons. In other cases, it might even be a good idea to just prioritize the high level requirements, and then letting the subordinate requirements inherit the priorities. If choosing this approach, it is important that all stakeholders are aware of this inheritance [57].

Regnell et al. discuss the problem of having a lot of requirements to prioritize [45]. They grouped the requirements to make the prioritization easier. The requirements were divided into a low level (original requirements) and a higher level (requirements were grouped based on relationships). This approach not only

reduces the number of requirements to prioritize but also deals with dependencies of requirements [50]. Grouping requirements based on requirements dependencies (e.g. which requirements must be implemented together) would make further analysis of the requirements easier since requirements that are grouped together would not compete for priorities (issues related to dependencies are further discussed in Chap. 5). According to the result of the study, forming coherent groups was easy and the stakeholders successfully prioritized at both levels.

4.6.2 Reprioritization

When developing software products, it is likely that new requirements will arrive, requirements are deleted, priorities of existing requirements change, or that the requirements themselves change [24, 39]. Hence, it is of tremendous importance that the prioritization process is able to deal with changing requirements and priorities of already prioritized requirements. When prioritizations are on an ordinal (e.g. ranking and numerical assignment) or absolute scale (estimating cost) this does not introduce any major problems since the new or changed requirement just need to be assigned a value, or a correct priority. Such iterations of the numerical assignment technique have been used successfully [17].

When using prioritization on a ratio scale (such as AHP), the situation becomes more complex since all requirements should be compared to all others to establish the correct relative priorities. However, it is possible to tailor this process by comparing new or modified requirements with certain reference requirements and thereby estimating the relative value. For example, when using the 100-dollar test it is possible to identify the two requirements with higher and lower *ranking*, and then establish the relative value in comparison to these and normalize the weights (of the complete requirements set). However, this means that the original process is not followed and the result might differ from a complete reprioritization even though the cost versus benefit of such a solution might be good enough. Cost and benefit must be taken into consideration when choosing a prioritization technique.

Further, it is important to not forget that priorities of already implemented requirements can change; especially non-functional requirements. Techniques such as gap-analysis (see Sect. 4.6.5) could be successfully used to prioritize already implemented requirements in order to take these into account in a reprioritization.

4.6.3 Non-Functional Requirements

Previously in this chapter, no differences in analyzing functional and non-functional (quality attributes) requirements have been discussed. The previously presented methods can be used with both kinds of requirements and sometimes it is preferable to prioritize them together. Nevertheless, it is not *always* advisable to prioritize functional and non-functional requirements together, for the same reasons that requirements at different abstraction levels should not be prioritized to-

gether. Differences between functional and non-functional requirements include, but are not limited to [36, 47, 56]:

- Functional requirements usually relate to specific functions while non-functional requirements usually affect several functions (from a collection of functions to the whole system).
- Non-functional requirements are properties that the functions or system must have, implying that non-functional requirements are useless without functional requirements.
- When implemented, functional requirements either work or not while non-functional requirements often have a "sliding value scale" of good and bad.
- Non-functional requirements are often in conflict with each other, implying that trade-offs between these requirements must be made.

Thus, it is not always possible or advisable to prioritize both types of requirements together. For example, if there is one functional requirement about a specific function and one non-functional requirement regarding performance, it could be hard to prioritize between them. In such cases, it is possible to prioritize them separately with the same or even with different techniques. Some techniques are especially suitable for prioritizing non-functional requirements. One such approach (originating from marketing) is conjoint analysis where different product alternatives are prioritized based on the definition of different attribute levels [22]. It should be noted that there does not seem to be a need to include all levels of all attributes (e.g. faster response time is always preferable). Since trade-offs often are present with such attributes (e.g. maintainability vs. performance), one idea is to only include comparisons where trade-offs are taken into consideration.

4.6.4 Introducing Prioritization into an Organization

As with other technology transfer situations, it is recommended to start small with one or a few of the practices (e.g. using numerical assignment to prioritize importance and cost) and then add more sophistication (and thereby complexity) as need and knowledge increase. Since introducing and improving prioritization is a form of process improvement, rules and guidelines for software process improvement should be applied (e.g. changes should be done in small steps and should be tested and adjusted accordingly [28]). A good idea could be to monitor future extensions by measuring process adherence and satisfaction of the involved stakeholders (both internally and externally). This way, it is possible to continuously measure the process and thereby determine when the process gets too heavy by calculating the cost versus benefit of each extension.

4.6.5 Evaluating Prioritization

Both for the reasons of improving and adjusting the prioritization process, and for improving and adjusting a product, it is necessary to evaluate the result of prioriti-

zations in retrospect. For both purposes, it is important that information about the priorities is kept since these provide the best information for analyzing both the product and the process [38]. This includes information about both selected and discarded requirements from a release [46]. When having access to this information, it is possible to do post mortem analysis to evaluate if the correct requirements were selected and if they fulfilled the stakeholders' expectations. If they did not, it is possible to change the process and the product for subsequent products/releases to get better prioritizations and more satisfied stakeholders. One way of evaluating if the correct priorities were assigned is through gap-analysis where the "gap" between perceived levels of fulfillment of a requirement and the importance of the requirement is calculated [27]. The result shows how well each requirement, or type of requirement, is fulfilled according to how important the stakeholders think the requirements are. In this case, the requirements with the largest gaps get the highest priorities for improvement (PFI) [27]. This makes it possible to improve parts of the product with a low level of fulfillment, but it could also be used to tune the process to avoid such situations again.

4.6.6 Using the Results of Requirements Prioritization

The results of a prioritization exercise must be used judiciously [39]. Dependencies between requirements should be taken into consideration when choosing which requirements to include. Dependencies could be related to cost, value, changes, people, competence, technical precedence, etc. [16, 49]. Such dependencies might force one requirement to be implemented before another, implying that it is not possible to just follow the prioritization list (dependencies are further discussed in Chap. 5). Another reason for not being able to solely base the selected requirements on the priority list is that when the priority list is presented to the stakeholders, their initial priority might have emerged incorrectly [39]. This means that when the stakeholders are confronted with the priority list, they want to change priorities. This is a larger problem in techniques where the result is not visible throughout the process (e.g. AHP).

The product may have some naturally built-in constraints. For example, projects have constraints when it comes to effort, quality, duration, etc. [50]. Such constraints makes the selection of which requirements to include in a product more complex than if the choice were solely based on the importance of each requirement. A common approach to make this selection is to propose a number of alternative solutions from which the stakeholders can choose the one that is most suitable based on all implicit context factors [24, 38, 48, 50, 57]. By computerizing the process of selecting nominated solutions, it is possible to focus the stakeholders' attention on a relatively small number of candidate solutions instead of wasting their time by discussing all possible alternatives [19]. In order to automate and to provide a small set of candidate solutions to choose from, it is necessary to put some constraints on the final product. For example, there could be constraints that the product is not allowed to cost more than a specific amount, the

time for development is not allowed to exceed a limit, or the risk level is not allowed to be over a specific threshold.

4.7 An Example of a Requirements Prioritization

To illustrate the different aspects, prioritization techniques, trade-offs between stakeholders, and combinations of prioritization techniques and aspects, an example of a prioritization situation is given. The method used in this example is influenced by a model proposed by Wiegers but is tailored to fit this example [57]. The example analyses 15 requirements (R1-R15) in a situation with three known customers (see 4.5.2). The analysis is rather sophisticated to show different issues in prioritization but still simple with a small amount of requirements. While many more requirements are common in industry, it is easier to illustrate how the techniques work on a smaller example. Each of the 15 requirements is prioritized according to the different aspects presented in Sect. 4.3. Table 4.3 presents the aspects that are used in the example together with the method that is used to prioritize the aspect and from which perspective it is prioritized.

Table 4.3 Aspects to prioritize

Aspect	Prioritization Technique	Perspective
Strategic importance	AHP	Product Manager
Customer importance	100-dollar / Top-ten[1]	Customers
Penalty	AHP	Product Manager
Cost	100-dollar	Developers
Time	Numerical Assignment (7)	Project Manager
Risk	Numerical Assignment (3)	Requirements Specialist
Volatility	Ranking	Requirements Specialist

As can be seen in Table 4.3, all prioritization techniques presented in Sect. 4.4 are used. However, two clarifications are in order. First, numerical assignment for time (7) and risk (3) uses a different number of groups to show varying levels of granularity. The customer importance is prioritized both by the top-ten technique and the 100-dollar technique depending how much time and cost the different customers consider reasonable.

To make the prioritizations more effective, requirements are further refined. First, requirements R1 and R2 are requirements that are absolutely necessary to get the system to work at all. Hence, they are not prioritized by the customers but they are estimated when it comes to cost, risk, etc. since R1 and R2 influence these variables no matter what. This is a way of using the requirements triage approach presented in Sect. 4.4.7. Further, two groups of requirements have been identified as having high dependencies (must be implemented together) and

[1] The top-ten technique is modified to a top-four technique in this example due to the limited number of requirements.

should hence be prioritized together. Requirements R3, R4, and R5 are grouped together as R345, and requirements R6 and R7 are grouped into R67.

Table 4.4 Prioritization results of strategic and customer importance. Priority, $P(R_X) = RP_{C1} \times W_{C1} + RP_{C2} \times W_{C2} + RP_{C3} \times W_{C3} + RP_{PM} \times W_{PM}$, where RP is the requirement priority, and W is the weight of the stakeholder

Requirement	C1 (0.15)	C2 (0.30)	C3 (0.20)	PM (0.35)	Priority:
R8	0.25	0.24	0.16	0.15	0.19
R9		0.07	0.14	0.03	0.06
R10	0.25	0.05	0.13	0.29	0.18
R11		0.05	0.01	0.02	0.02
R12		0.16	0.04	0.01	0.06
R13		0.05	0.16	0.02	0.05
R14	0.25	0.02	0.10	0.10	0.10
R15		0.03	0.04	0.05	0.03
R345		0.04	0.18	0.17	0.11
R67	0.25	0.29	0.04	0.16	0.19
Total:	1	1	1	1	1

Table 4.5 Descending priority list based on importance and penalty (IP). $IP(R_X) = RP_I \times W_I + RP_P \times W_P$, where RP is the requirement priority, and W is the weight of Importance (I) and Penalty (P)

Requirement	Importance (0.7)	Penalty (0.3)	IP	Cost	Time	Risk	Volatility
R1	1	1	1	0.11	3	1	2
R2	1	1	1	0.13	4	2	1
R8	0.19	0.2	0.20	0.07	1	3	7
R67	0.19	0.09	0.16	0.10	6	3	5
R10	0.18	0.01	0.13	0.24	2	3	11
R14	0.10	0.16	0.12	0.01	1	3	10
R345	0.11	0.02	0.08	0.03	3	2	8
R9	0.06	0.12	0.08	0.09	3	2	9
R15	0.03	0.17	0.08	0.05	5	1	4
R12	0.06	0.06	0.06	0.11	4	2	6
R11	0.02	0.14	0.06	0.02	3	1	3
R13	0.05	0.03	0.05	0.04	7	1	12
Total / Median:	3	3	3	1	3	2	

The next step is to prioritize the importance of the requirements. In the case at hand, the three known customers and the product manager prioritize the requirements. Furthermore, these four stakeholders are assigned different weights depending on how important they are deemed by the company. This is done by using the 100-dollar test to get the relative weights between the stakeholders (see Sect. 4.5.5). Table 4.4 presents the result of the prioritization. In the table, the three customers are denoted C1–C3 and the product manager is denoted PM.

As can be seen in this table, the different stakeholders have different priorities, and it is possible to combine their different views to an overall priority. The

weights (within parenthesis after each stakeholder) represent the importance of each customer and in this case, the product manager is assigned the highest weight (0.35). This is very project dependent. In this case, the mission of this product release is to invest in long-term requirements and attract new customers at the same time as keeping existing ones. As also can be seen, C1 used the top-ten technique and hence the priorities were evenly divided between the requirements that this customer regarded as most important. The list to the far right presents the final priority of the requirements with the different stakeholders and their weights taken into consideration. This calculation is possible since a ratio scale has been used instead of an ordinal scale.

The next step is to prioritize based on the other aspects. In this case, the Priority from Table 4.4 is used to express Importance in Table 4.5. It should also be noted that requirements R1 and R2 (absolutely necessary) have been added in Table 4.5.

Table 4.6 Selected requirements based on IP and cost

Requirement	IP	Cost	IP/Cost	Time	Risk	Volatility
R1	1	0.11	9.09	3	1	2
R2	1	0.13	7.69	4	2	1
R8	0.20	0.07	2.80	1	3	7
R67	0.16	0.1	1.59	6	3	5
R10	0.13	0.24	0.54	2	3	11
Total / Median:	2.48	0.65	21.71	3	3	

Table 4.5 shows a prioritized list of the requirements (based on IP). With this information there are two options: 1) pick prioritized items from the top of the list until the cost constraints are reached, 2) analyze further based on other prioritized aspects, if prioritizations of additional aspects are available. The example has two major constraints: 1) the project is not allowed to cost more than 65% of the total cost of the elicited requirements, and 2) the median risk level of the requirements included is not allowed to be higher than 2.5. Based on this, we first try to include the requirements with the highest IP. The result of this is presented in Table 4.6 where the list was cut when the sum of costs reached 65% of the total cost of elicited requirements.

Table 4.6 shows that we managed to fit within the cost constraints but could not satisfy the risk constraint. As a result, the project becomes too risky. Instead, another approach is taken to find a suitable collection of requirements. In this approach, we take the IP/Cost ratio into consideration. This shows which requirements provide most IP at the least cost. In this case, we try to set up a limit of only selecting requirements that have an IP/Cost-ratio higher than 1.0. The result is presented in Table 4.7. Table 4.7 shows the cost constraints are still met (even nine percent less cost) while also satisfying the risk constraint. Comparing tables 4.6 and 4.7 shows that the IP-value of the second candidate solution is higher which indicates that the customers are more satisfied with the product and the IP/Cost ratio is almost doubled. The second candidate solution satisfies 91 percent (2.73/3) of the IP aspect, compared to 83 percent in the first candidate solution. The fact that the second alternative costs less and is less risky also favors this choice. Nev-

ertheless, the above example is not optimal since cost was constrained at 0.65 and other combinations of requirements may be more optimal for the selection.

Table 4.7 Selected requirements based on cost and IP/cost ratio.

Requirement	IP	Cost	IP/Cost	Time	Risk	Volatility
R1	1	0.11	9.09	3	1	2
R2	1	0.13	7.69	4	2	1
R8	0.20	0.07	2.80	1	3	7
R67	0.16	0.1	1.59	6	3	5
R14	0.12	0.01	11.70	1	3	10
R345	0.08	0.03	2.71	3	2	8
R15	0.08	0.05	1.50	5	1	4
R11	0.06	0.02	2.94	2	1	3
R13	0.05	0.04	1.17	7	1	12
Total / Median:	2.73	0.56	41.19	3	2	

This type of release planning is known in operational research as the binary knapsack problem [13]: maximize value when the selection is bounded by an upper limit. However, the difference between a classical knapsack problem and the problem faced above is that release planning is a "wicked problem" [13]. This means that an optimal solution may not exist, that every release planning is unique, and that no objective measure of success exists, etc. [13]. In addition, the values of the aspects in the above example are estimates and subjective measures in comparison to objective measures such a length, weight, and volume. Instead of finding the optimal set, different alternative solutions should be discovered and the alternative that seems most suitable should be chosen [13]. This implies that the purpose with prioritization is not to come up with a list of final requirements, but rather to provide support for good decisions. In comparison to the above example, real projects generally have more requirements, and more complex dependencies [13]. However, this example was meant to show how different aspects can be used to handle trade-offs between different (sometimes conflicting) aspects. It is also possible, as illustrated, to fine-tune an existing technique or method to suit a company specific situation.

4.8 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 [48]. However, there exist some studies that have evaluated different prioritization techniques [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 research 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 actually 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. Instead, 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 environments 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. Dependencies are not covered in this chapter since Chap. 5 discusses this in detail. 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 Sect. 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.

4.9 Summary

This chapter 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.

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. In: 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. In: 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, Sweden
8. Bradner S (1997) RFC 2119. http://www.ietf.org/rfc/rfc2119.txt (24 November 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. Addison-Wesley Longman, Boston
11. Carlshamre P (2001) A usability perspective on requirements engineering – From methodology to product development. Ph.D. thesis, Linköping Institute of Technology, Sweden
12. Carlshamre P, Sandahl K, Lindvall M, Regnell B, Natt och Dag J (2001) An industrial survey of requirements interdependencies in software release planning. In: Proceedings of the 5th 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 development: provoking an understanding requirements engineering 7(3):139–151
14. Clements P, Northrop L (2002) Software product lines – Practices and patterns. Addison-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 – Molding the state of research into a research agenda. In: Proceedings of the 9th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ '03). Universität 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 identify future requirements. In: Proceedings of the 11th ACM SIGPLAN Conference on Object-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 requirements. In: Proceedings of the IEEE Joint International Conference on Requirements Engineering (RE'02). IEEE Computer Society, Los Alamitos, pp. 263–270
20. Fenton, NE, Pfleeger SL (1997) Software metrics – A rigorous and practical approach, 2nd Edition. PWS Publishing Company, Boston
21. Gallis H, Arisholm E, Dybå T (2003) An initial framework for research on pair programming. In: 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 preferences. In: Proceedings of the IEEE Joint International Conference on Requirements Engineering (RE'02). IEEE Computer Society, Los Alamitos, pp.206–209
23. Gorschek T (2004) Software process assessment & improvement in industrial requirements engineering. Licentiate Thesis, Blekinge Institute of Technology
24. Greer D, Ruhe G (2004) Software release planning: An evolutionary and iterative approach. 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 process. Mathematical Modeling 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. In: 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. In: 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. In: Proceedings of the European Software Process Improvement Conference (EuroSPI 2004), Springer-Verlag, Berlin Heidelberg, pp.161–170
40. Lubars M, Potts C, Richter C (1993) A review of the state of practice in requirements modeling. In: 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. 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 prioritization 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. In: Proceedings of 1st Swedish Conference on Software Engineering Research and Practise (SERP'01). 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. In: 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 learning software organizations. Advances in learning software organization, Lecture Notes in Computer Science, Springer-Verlag, Vol. 2640, 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 modeling 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. In: Proceedings of 2nd Symposium of Quality Software Development Tools. IEEE Computer Society, Piscataway, pp.211–223

Author Biography

Patrik Berander is a Ph.D. student in Software Engineering at the School of Engineering at Blekinge Institute of Technology in Sweden. He received his degree of Master of Science with a major in Software Engineering – with a specialization in Management in 2002. His research interests are requirements engineering in general and decisions related to requirements and products in particular. Further research interests include software product management, software quality, economic issues in software development, and software process management.

Dr. Anneliese Amschler Andrews is the Huie Rogers Endowed Chair in Software Engineering at Washington State University. Dr. Andrews is the author of a textbook and over 130 articles in the area of Software Engineering, particularly software testing and maintenance. Dr. Andrews holds an MS and PhD from Duke University and a Dipl.-Inf. from the Technical University of Karlsruhe. She served as Editor in Chief of the IEEE Transactions on Software Engineering. She has also served on several other editorial boards including the IEEE Transactions on Reliability, the Empirical Software Engineering Journal, the Software Quality Journal, the Journal of Information and Software Technology, and the Journal of Software Maintenance. She was Director of the Colorado Advanced Software Institute from 1995 to 2002. CASI's mission was to support technology transfer research related to software through collaborations between industry and academia.