8 Quality Assurance in Requirements Engineering

Christian Denger and Thomas Olsson

Abstract: This chapter presents a survey of the state of the art for quality assurance for requirements. The meaning of quality in the requirements context is discussed, as is the influence of the quality assurance during requirements on other parts of the development. Different quality assurance approaches are categorized as either constructive (e.g., standards, guidelines, elicitation techniques) or analytical (e.g., inspections) and discussed with respect to their impact on the requirements quality. Based on the approaches, future challenges are discussed. The main future challenges lie in investigating the return on investment of quality assurance in the requirements context and to provide more empirical results which approach that effectively prevent or detect which problems.

Keywords: Quality assurance, Requirements, Quality characteristics, Inspections, Analytical approaches, Constructive approaches.

8.1 The Importance of Early Quality Assurance

Continuously increasing complexity, ever-increasing market pressure, and customers' demands for higher quality require a combination of carefully selected validation and verification techniques to deliver a software product on time, within budget and with the desired quality. Requirements engineering is the initial part of a software development process, and all later steps of the development are influenced by the requirements, making the quality of the requirements an important factor for the overall quality of the developed system.

Independent of the domain, quality assurance (QA) is an important but elusive part of software development. Traditionally, QA techniques have mainly focused on the later development phases such as the implementation phase and the related testing activities. However, QA can and should start earlier. This chapter addresses exactly this aspect by discussing QA activities that can be applied in the requirements engineering phase.

Why it is important to detect defects as early as possible? An issue that originates in the requirements runs the risk of affecting not only other requirements but also later phases and can cause follow-up defects in architecture, design, coding and testing, see Fig. 8.1.

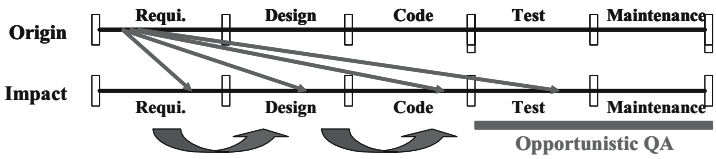

Fig. 8.1 Impact of requirements issues

If the quality assurance is only performed in the test and maintenance phase, one is dependent on the ability of the requirements engineers, designers and programmers to produce good working products, suitable for the rest of the development. That is, you rely on their ability not to make any crucial mistakes. However, this would reflect an ideal case that in almost all cases cannot be achieved (it is natural that humans make errors). Having no intermediate QA, i.e. a quality gate for the intermediate work products, it is most likely that the design and implementation are based on the wrong requirements. This, in consequence, leads to high rework effort as not only the code but most often the overall system architecture and design have to be revised due to requirements defects. Nevertheless, it seems to be quite common to do QA only by means of testing (and maintenance approaches), which, therefore, is an opportunistic approach.

Many studies show that late, opportunistic QA leads to a stressful and costly test and maintenance phases. Issues should be resolved in the phase of their origin to avoid costly testing and rework. Testing and rework can account for up to 40–50 % of the development effort [10]. In addition, removing defects early in the development process is more cost effective than addressing the defects during testing or maintenance [7]. Correcting a defect late in the process gets more expensive as development effort has already been spent and more artifacts are affected. A requirements issue can become up to 100 times more expensive if it is detected in operation, compared to detecting it in the requirements phase [10].

Based on these data, the knowledge that requirements deficiencies are the prime source of project failures [21], and that over 40% of problems in the software development cycle result from low quality requirements [47], QA techniques for requirements are one of the most promising and cost effective techniques to ensure successful development and to prevent avoidable rework in later phases. Independently of whether high quality is required or not, QA in the requirements phase pays of. But it does, of course, become even more important if high quality is a key success factor.

The remainder of the chapter discusses quality of and quality assurance for requirements. Techniques to assure the specified quality aspects are discussed and a framework on how to integrate the different QA techniques is presented (Sect. 8.2). The subsequent sections deal with concrete QA approaches. A general introduction to constructive apaches is described in Sect. 8.3, together with examples of constructive approaches. The analytical approaches, such as inspections and early test case creation, are presented in Sect. 8.4. In Sect. 8.3 and Sect. 8.4 the issue of traceability and how it can be used to facilitate QA is elaborated. The final

sections, Sect. 8.7 and Sect. 8.8, summarize the important future work and conclude the chapter, respectively.

8.2 Requirements and Quality Assurance

Quality is hard to define as it is a complex concept, dependent on organizational viewpoints and context characteristics [32]. For example, do fewer defects per lines of code equal high quality? What if one of these defects causes the loss of life? Quality has a very different meaning in different situations. In a word processor, different quality criteria are important than in an electronic control unit of a car or an airplane.

With requirements, this becomes even more difficult, as the notion of quality often depends on the opinions of various stakeholders. For example, if you have not understood the stakeholders' needs correctly, you are bound to end up with a system that is not considered to be of good quality as it might not support the user in fulfilling certain tasks. This section introduces how quality and quality assurance can be defined for requirements and presents aspects of defining a quality strategy for early QA.

First of all, it is important to define what is meant by a defect in the requirements phase. In this chapter, the term *issue* is used as an umbrella term for all matters that should be resolved in the requirements context. The terms defects, errors, faults or problems are other words used with a similar meaning. However, in the case of requirements, it is sometimes unclear whether an issue really is a defect. For example, if two stakeholders disagree on one aspect of a requirement, this is an issue that should be resolved, but would usually not be referred to as a defect in the traditional sense. If it is not resolved, at least one stakeholder will reject the system in acceptance test. However, contradicting requirements are closer to the conventional interpretation of a defect. Therefore, the matters mentioned in the examples are summarized as requirements issues that need to be resolved through the QA activities on the requirements.

8.2.1 Quality of Requirements

The quality of requirements is dependent on various stakeholders and their perspective. Several different views need to be considered in order to define what quality means in a certain context [32]. The first view on quality is the transcendental view. Therein, quality is considered as something that we always strive for as an ideal but we will never be able to implement this ideal. The goal of this viewpoint is to express the complexity of the concept quality in general. Second, the user view evaluates the quality of a software product with respect to its fitness of purpose to fulfill certain user tasks. The third view, the manufacturing view, focuses on the product view during production and after delivery. It is focused on the adherence of standards and evaluates whether the product was build right the

first time. The fourth view is the product view. The focus for this view is on internal quality aspects of the product that can be measured. It is assumed that ensuring certain internal quality aspects has an impact on the external quality and the quality in use of the product. Finally, the value-based view relates quality to cost. It considers quality as something the customer is willing to pay for [32].

Mapping these views on the quality of requirements reveals relevant stakeholders and needed QA for the requirements. The requirements should, for example, describe what the user requires of the final system (user-view). Furthermore, they should be described in a way that allows the developers to produce the software effectively and efficiently (product-view). The requirements engineers have to follow certain standards when specifying the requirements to ensure the quality of the requirements right from the start (manufacturing view). Finally, the customers have to decide on the value of each requirement and whether the implementation cost is motivated (value-based view).

All these aspect have to be considered when discussing the quality of requirements. The inherently human based nature of requirements engineering and the necessity to consider not only technical but also social aspects when eliciting, negotiating and specifying requirements makes the definition of quality characteristics for requirements even harder. Standards are a starting point for defining the quality of requirements and requirements specifications [24, 25]. Further, there exist a number of processes, guidelines, and best practices on how to perform good requirements engineering [8, 11, 14, 41, 46, 50]. The advocates of these approaches argue that, for example, adhering to the process facilitates requirement engineering and minimizes later quality problems. In order to specify an initial set of quality criteria, the IEEE standard for requirements specification [24] is used as a starting point (see Table 8.1). The standard is extended to provide a more complete picture of relevant quality aspects of requirements (e.g. [16]), especially to address customer and user needs (value-based and user view on requirements quality). Moreover, we extended the definition of the quality attributes beyond the quality of a requirements specification. In accordance to the different views on quality in general, the definitions of the quality attributes were adapted (see also [13]). In consequence, the quality aspects consider technical and human related aspects, which both are relevant for the overall quality of the requirements.

The information in brackets behind the attribute name specifies whether the attribute is originally defined in the IEEE standard or whether the attribute is part of the extension (IEEE/new). The second information specifies which view on the requirements' quality is addressed with the attribute.

Table 8.1 Quality attributes for requirements (1 of 2)

Quality Attribute	Definition
Correctness (IEEE, user-view)	The requirements that are implemented have to reflect the expected (intended) behavior of the users and customers. That is, everything stated as a requirement is something that shall be met by the final system to fulfill a certain purpose (suitability).

Table 8.1 ***(cont.)*** Quality attributes for requirements (2 of 2)

Quality Attribute	Definition
Unambiguity (IEEE, product-view)	The requirements should only have one possible interpretation. Note that one requirement might be unambiguous to a certain group of stakeholder but has a different meaning in another. It is important to involve all stakeholders in the requirements engineering process to gain a common understanding (see Chaps. 2 and 3)
Completeness (IEEE, product-view)	All important elements that are relevant to fulfill the different user's tasks should be considered. This includes relevant functional and non-functional requirements and interfaces to other systems, the definition of responses to all potential inputs to the system, all references to figures and tables in the specification, and a definition of all relevant terms and measures.
Consistency (IEEE, product, manufacturing view)	The stated requirements should be consistent with all other requirements, and other important constraints such as hardware restrictions, budget restrictions, etc.
Ranked for Importance / Stability (IEEE, product, value-based, user view)	Each requirement specifies its importance and/or its stability. Stability expresses the likelihood that the requirement changes, while importance specifies how essential the requirement is for the success of the project (from a value-based and a user point of view). See also Chap. 5
Verifiability (IEEE, product view)	All requirements should be verifiable. That is, there exists a process for a machine or a human to check (in a cost effective way) whether the requirement is fulfilled or not.
Modifiable (IEEE, product view)	All requirements should be modifiable, that is the structure of the requirements and the requirements specification allow the integration of changes in an easy, consistent and complete way.
Traceable (IEEE, manufacturing view)	All requirements should be traceable, that is, it should be possible to reference the requirement in an easy way. Moreover, it is possible to identify the origin of a requirement (see also Chap. 4)
Comprehensibility (New, manufacturing, user, value-based view)	The requirements are specified and phrased in a way that is understood by all involved stakeholders.
Feasibility (New, value-based, product view)	All requirements can be implemented with the available technology, human resources and budget. Moreover, all requirements contribute to the monetary success of the system, that is, they are worth to include in the system.
Right Level of Detail (New, user, manufacturing, value-based view)	The information given in the requirements is suitable to gain the right understanding of the system and to start implementation. There are no unnecessary implementation or design details specified in the requirements.

The IEEE Standard was extended to give a more complete way of describing the quality of requirements:

- C*omprehensibility* is essential, as there are many different stakeholders involved in the requirements engineering process. It is important that the requirements can be easily understood by all of these stakeholders and that they all have a common understanding of the requirements.
- *Feasibility* is especially important to consider as a requirement and is only of value if it can be transformed into a design and an implementations with reasonable effort and cost.
- Finally, the requirements should be specified on an *adequate level of detail,* that is, concrete enough to allow that design and implementation can be started ,but that is on the other hand abstract enough to allow discussion between all involved stakeholders (which have in many cases technical and non-technical backgrounds).

Note that there are relationships among the attributes. For example, ambiguous requirements are also difficult to understand. Further, if the requirements are not traceable, the verifiability, modifiability and the comprehensibility can be affected. Even though the classification is not orthogonal, each attribute refers to a special aspect of requirements' quality that should be considered. A more detailed analysis is needed regarding how the different quality attributes impact each other and how this information can be used to balance QA activities on the requirements (see Sect. 8.5).

8.2.2 Requirements Quality Strategy

Developing software without any defects is impossible (see, for example [32], specifically the transcendental view of quality). It is, however, possible to achieve an optimal compromise between the desired quality and available resources, considering the specific context factors and quality need of a company or a project. Many factors influence the importance of different quality attributes in a specific context. For example, in certain domains, it is more important to be the first on the market than to have high quality products in the sense of few defects. There is a lot of software being tremendously successful, from a commercial point of view, which is anything but of high quality. On the other hand, the cost of a single defect can be fatal and incredibly expensive, for example the Ariane 5 disaster [33]. Thoroughness and budget for quality assurance need to be related to the cost of erroneous implementation, leading to financial or human costs.

During the requirements engineering phase, it is important to define a quality strategy that addresses those quality issues that can easily be verified and validated in the requirements phase. Other quality aspects that cannot be efficiently addressed during the requirements phase should be left for later phases.

A *quality strategy* defines how, when and where different QA approaches, in combination with other approaches in the software development process, are used to assure high quality. This includes the planning of resources (which approach is applied when and how much effort should be spent) and the definition of an optimized combination of the different QA approaches with the aim of achieving the

desired quality at the desired cost. The definition of such a strategy is not a trivial task. It requires detailed knowledge about the context of the company and the project, the required level of assurance of the different quality attributes (i.e. to which degree we can be sure the requirement are fulfilled) and which QA approaches are applicable. Figure 8.2 summarizes the elements impacting a quality strategy. At the top of the picture are context related elements, at the bottom technically oriented elements. There are five context elements relevant for QA strategies:

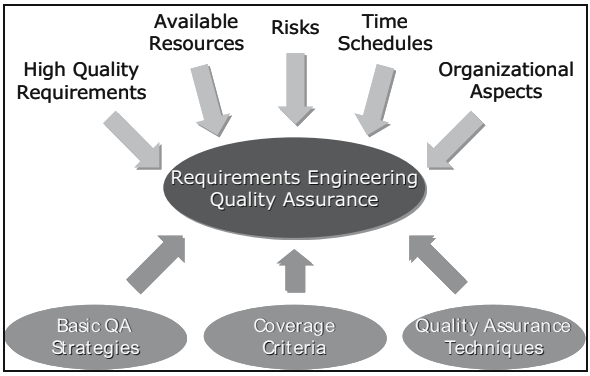

Fig. 8.2 Elements important to define a quality assurance strategy for requirements

1. *The quality of requirements* specifies the quality criteria for good requirements, as described in the previous section. These criteria can vary from company to company and from project to project. They impact the strategy in that they specify what should be achieved with the quality strategy. It is important to define optimal and minimal sets of quality characteristics of requirements [32].
2. The *available resources* describe the available effort, budget, hardware, and personnel to perform QA during the requirements activities. In addition, the availability of additional experts has to be considered, as for certain quality assurance approaches, certain stakeholders beyond the requirements engineering processes might be essential (e.g. lead architect during requirements reviews). The available resources have also a direct impact on the applicable QA approaches. For example, if only a small effort is available to perform requirements reviews it is not possible to fulfill a full Fagan inspection with many participants but only a peer review or desk-checking approach [49].
3. *Risks* related to certain requirements, especially risks of not realizing a requirement or implementing a requirement in the wrong way, are an additional factor influencing the quality strategy. Risk is defined as not being able to live up to the quality goals and is an important factor for deciding on which part of the requirements which QA approach should focus. For example, not meeting a requirement important to protect human lives bears a high risk and should therefore be checked extra carefully. Moreover, risks can be used to plan the

limited quality assurance resources. For example, with the help of risk analysis, it is possible to identify the most critical requirements in the sense of loss of lives or loss of money. The QA approaches should then be focused exactly on these aspects (see also Chap. 5 for related approaches).
4. The overall *time schedule* is related to the available resources and defines the time available for QA in general and within the requirements phase in particular. Time resources are especially important as they relate the requirements QA activities with other development activities.
5. Finally, the *organizational aspects*, such as development process, e.g., plan-driven or agile development, or product domain, (e.g., desktop software or airplane control system) influence the decision on which QA approaches to use. Moreover, it is important to take the various stakeholders into account. Dependent on the domain, different sets of stakeholders are varying importance (see also Chaps. 2 and 3). These aspects impact the quality strategy in that certain QA approaches might not be applicable due to the organizational constraints. For example, in an agile process, requirements reviews are almost impossible to perform as in the most agile processes requirements are not documented in a way that would allow an inspection (e.g. user stories in extreme-programming often are not longer than one sentence that specifies a general feature [5]).

The context elements are important to consider as they define in which way the QA approaches can be applied and which restrictions and constraints must be adhered to. Beside the context in which the quality strategy is embedded, it is also important to consider technical aspects of quality assurance:

1. *The basic strategies* represent those strategies in place in a company or a project that define how to perform QA in the requirements phase. In that sense they represent the current state of the practice in a certain context. Due to the lack of sophisticated quality strategies, ad-hoc approaches are most frequently applied. For example, the simplest but also the least systematic strategy is to state that everything in the requirements specification should be verified or that all quality issues should be tackled in later development phases. Experience based strategies give hints on what to address in the requirements based on the experience of earlier projects. Such basic strategies should be considered when creating a more sophisticated quality strategy. They provide valuable input on where to start from and what has paid of in the past.
2. The *coverage criteria* define which aspects of the requirements should be covered by the QA approach. One example of a coverage criterion is that all requirements are covered by at least one test case. An aspect related to coverage that should be considered is the depth of the QA approach [35]. Depth defines the level of detail to which the requirements are verified or validated or, in other words, the quality level to be achieved. The greater the depth, the more resources are required for QA and the more sophisticated QA approaches are required.
3. The most important element of a requirements quality strategy is the potential *quality assurance approaches* and methods that can be used to ensure the dif-

ferent quality characteristics of the requirements. As discussed, the context elements and the technical elements impact the applicability of QA approaches. The QA approaches are the technical core element of the quality strategy as they represent the means of achieving good requirements quality.

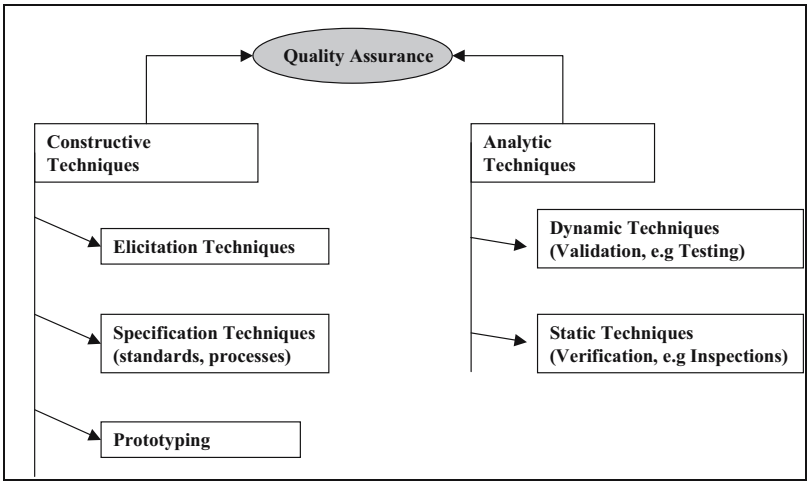

Fig. 8.3 Excerpt of quality assurance approaches

The framework presented in this section supports the definition of a good quality assurance strategy. It specifies which elements are important to consider when talking about quality assurance in the requirements engineering phase. It is important that all these elements are considered in the specific context of a specific company and have to be instantiated accordingly. To instantiate the framework into a concrete quality assurance strategy, it is essential to stress the continuous collection of data. Such a measurement approach should address the question which requirements issues are the most expensive ones and which quality assurance techniques work best in the specific context. The most essential element in the framework is the QA techniques (QA approaches) that can be applied. This is the element that should be considered first, i.e. before defining a detailed QA strategy, it is important to investigate potential approaches to verify the quality attributes of the requirements.

8.2.3 Quality Assurance Approaches for Requirements

In this report the quality assurance approaches are divided into one of two classes: constructive and analytical approaches. Figure 8.3 provides some examples of QA approaches of the different classes.

Constructive approaches ensure that mistakes are minimized during the creation of a work product (e.g. the requirements specification). That is, they prevent

issues from being introduced. Examples of constructive approaches in the requirements phase are style guidelines on how to specify requirements, templates for the requirements specification, elicitation approaches and prototyping.

Analytical approaches are performed on the completed artifact or a self contained part of it with the aim to detect issues. Analytical quality assurance approaches can be further divided into *static quality assurance approaches, dynamic quality assurance approaches* (including formal methods) [36]. The difference between the two classes is that dynamic approaches require an executable version of the system. Testing approaches are examples of dynamic quality assurance. Static quality assurance approaches can be performed without executing code. Inspections and formal verifications are an examples of static approaches. There is in most cases no executable code available during the requirements engineering phase. Hence, usually only static approaches are applicable.

It is important to distinguish between QA in the requirements analysis phase and in the requirements validation phase [46]. QA in the analysis phase means that requirements issues are prevented from being introduced (i.e. during elicitation) with the help of constructive approaches omissions and ambiguous requirements are addressed. The validation process of requirements is based on a requirements document and tries to resolve issues within this document. Here, the analytical approaches are applied.

8.3 Constructive Approaches

Constructive approaches ensure quality during the creation of the requirements. In that sense, constructive approaches are preventive, as they aim to minimize mistakes from being made. These approaches are called constructive as they are applied while developing the requirements. Different ways constructing requirements and eliciting them from the various stakeholders are discussed in Chap. 2 and Chap. 3 of the book. How these approaches contribute to higher quality of the requirements in this section.

Requirements engineering is largely a human-based activity. Even if formal methods are used, at some point you will be interacting with customers and other stakeholders. As we humans are fallible, we are bound to make mistakes. Therefore, even if constructive methods are applied according to all the rules, there will still be a need to check the results, that is, apply analytical approaches. In this section, the impact of constructive approaches is presented. In Sect. 8.4, the analytical approaches are presented.

8.3.1 Elicitation Techniques

The elicitation step is important to the overall quality of the requirements and the acceptance of the final system [35, 46]. During the elicitation step, requirements are captured from various sources, such as the customer, the users, earlier projects,

market studies etc. In this process, various stakeholders such as the customers, the technical staff (developers), and end users work together to derive an appropriate set of requirements. Requirements engineers can apply different techniques to support the various stakeholders in discovering the requirements, e.g. interviews, questionnaires, workshops and focus groups (see Chap. 2 for more details).

By means of elicitation techniques, the following quality attributes can be ensured:

- Comprehensibility: by developing a common terminology and ensuring that the different stakeholders speak the same language, comprehensibility is improved.
- Completeness: if the elicitation is performed correctly, all the (relevant) stakeholders, and their individual stakes, should be identified. Here, elicitation activities contribute to higher quality in that they support the requirements engineers in the identification processes.
- Verifiability and feasibility: again, by involving the relevant stakeholders, quality can be assured. By involving the testers the attribute verifiability is improved, and by involving the developers feasibility is improved.
- Correctness: the elicitation process should be driven by the business concerns [46]. Suitability, as part of correctness, is supported by this, as it is then more likely that the developed software will bring a real financial benefit in the context of use.

8.3.2 Specification Techniques

The main objective of the specification step is to document the requirement in such a way that they can be used as a basis for development (see Chap. 3). Usually, the output of the specification activity is a requirements document that captures the relevant aspect of the system to be built (i.e. functional, non-functional aspects, restrictions, etc.). In the section it is outlined how certain specification techniques, best practices and standards can help to ensure the quality of the requirements.

Standards, such as IEEE 830-1998 and IEEE 1233-1998 [24, 25], describe which elements a “good” requirements specification should have and which quality attributes the requirements should fulfill. Templates also provide elements that should be specified when documenting the requirements. Examples include templates on how to specify use cases or how to structure the requirements document.

With respect to the quality characteristics defined in Sect. 8.2.1, standards and templates contribute to better requirements in the following way:

- Completeness: in the case that the requirement engineers adhere to the recommendations in the standards and apply the pre-defined templates it can be ensured that all relevant aspects of a requirements document are considered, i.e. completeness of the document.
- Understandability and modifiability: the structure provided by templates and standards ensures that requirements document look similar over different projects in a company. Standardization of requirements documents prevents ambi-

guities within the documents and improves the understandability as well as the modifiability, as elements that need to be changed can be found more easily.

In addition to standards and templates, there is a huge collection of best practices showing how different steps in the requirements engineering process should be performed in order to gain high quality output of each of these steps only to mention some of them: [8, 11, 14, 17, 25, 35, 41, 46, 50].

Specifying functional requirements using, for example, use cases and related scenarios ensures also the comprehensibility of the requirements right from the start, as use cases and scenarios are easy to understand for technical and non-technical stakeholders. This also supports the attribute right level of detail. In addition, use cases seem to be valuable source for the definition of acceptance and system test cases (see Sect. 8.4.2). Therefore, specifying the requirements in a structured, scenario-oriented way improves their verifiability.

Basically, it would be possible to address almost all of the quality attributes in a constructive way if certain processes and standards are rigorously followed and applied. However, practice shows, that such rigorous approaches are not always reasonable or feasible (e.g. due to time restrictions, budget restrictions, regulations, etc.).

8.3.3 Prototyping

Another constructive approach that can be used to support elicitation is prototyping. A prototype is an executable version of the system under development, though restricted in one way or another. For example, a user interface prototype implements parts of the user interface, the structure and navigation, but will not have all the functionality, while a performance prototype focuses on memory and CPU load and might have no user interface at all.

The goal of a prototype is for the stakeholders to be able to try the system and make improvement suggestions [46]. By doing this, they get a better feeling of whether the system represents what they required, and thus it helps to identify missing requirements and detect misconceptions. The most important value of a prototype is that it crosses the gap between the description and implementation [17]. Further, a quite common issue with the requirements is that the customer does often not exactly know what they want.

In general, developing a prototype requires a careful study of the requirements [46]. A prototype typically target the following quality attributes:

- Inconsistencies and incompleteness: the process of developing a prototype will, in it self, reveals inconsistencies and incompleteness of the requirements and thus improves their quality.
- Correctness: correctness is improved by letting the different stakeholders work with and evaluate a concrete object rather than the abstract requirements.
- Feasibility: by trying out different solutions, already in the requirements phase, feasibility is improved. A lot of time and money can be saved if dead-ends are detected at an early stage.

To underline the benefits of prototyping in the context of QA, an experiment showed that prototyping can significantly reduce requirements and design errors, especially for the user interfaces [9].

8.4 Analytical Approaches

The analytical quality assurance approaches assess the requirements specification to check whether the requirements specified in there fulfill the quality criteria specified. The main challenge of the analytical approaches is that there are no reference documents against which the requirements can be checked, i.e. there is no documented source of truth against to compare. This emphasizes that QA of requirements has to involve all relevant stakeholders of the requirements. In the following, two analytical approaches requirements inspections and test case creation (as a part of acceptance testing) are presented in more detail.

8.4.1 Requirements Inspections

Inspections are a valuable means to ensure the quality of a software product right after its creation. There are many experimental and industrial results that show the value of inspection in general and requirements inspection in particular [2, 3, 4, 9, 17, 19, 20, 34, 37, 40, 43, 44, 48, 49]. Inspections in general aim at minimizing the issues of a certain product being propagated to later phases, as the issues are addressed in the same phase in which they are introduced. Considering the costs of an requirement issue (see Sect. 8.1), requirements inspections are one of the most cost effective QA approaches, as they prevent issues from being propagated from the requirements to other artifacts and cause follow-up defects and avoidable rework [7, 17, 37, 44, 49].

A second important benefit of early QA is that many organizations report an improved knowledge transfer achieved when performing early QA activities such as inspections and test case creation. For example, with the help of the reading scenarios and the checklist questions it is possible to transfer knowledge about defect patterns, best practices and known pitfalls from experts to less experienced people.

An inspection is characterized by a process, the roles involved in the process, reading techniques used, and the information on how the results of the inspection are documented. These elements can be seen as the four dimensions of an inspection [34].

The Inspection Process

A basic inspection process contains four main steps: planning (managing the organizational issues of an inspection), detection (inspectors search for issues in the document under inspection), collection or meeting step (moderated meeting merging the results of the inspectors into approved defect list) and correction (where the author has to resolve all the identified issues). These steps are common for al-

most all instantiations of the inspection process. However, several inspection processes mention additional steps such as the overview meeting or the follow-up meeting [18, 49]

Each phase of the process can be implemented in different ways depending on the level of detail with which the requirements should be inspected. For example, in the case that the requirements should be checked only from an abstract viewpoint, the individual preparation phase of the process could be skipped and the requirements would be discussed during a meeting with certain experts. According to the IEEE Standard 1028-1997 [26], such a process would be similar to a walkthrough of the requirements document. The company applying the inspection approach has to decide to which level of detail the requirements should be inspected [49]. This mainly depends on the requirements quality strategy as discussed in Sect. 8.2.2 (see discussion on how different elements of the framework impact the QA approaches). The above-mentioned process steps are the four most essential steps that should be performed in case the requirements are to be inspected in a more detailed way.

Reading Techniques

The most important, but also the most difficult step, in a requirements inspection is the detection step. In this step, the inspectors identify requirements issues. A reading technique supports the inspectors in performing this step. A reading technique represents a series of steps or procedures that guide an inspector in acquiring a deeper understanding of the requirements under inspection and detecting issues in them [34].

There are different kinds of reading techniques that can be used during a requirements inspection: ad-hoc reading (reading without further guidance based on ones experience), checklist-based reading (using a list of questions to point to potential issues in the requirements) and scenario-based reading (using a step-wise description to guide the inspector during the defect detection step). Again, depending on the desired level of depth and coverage, one of these techniques might be more suitable for verifying the requirements than another. A more detailed summary of different reading techniques can be found in [34].

Checklist based reading (CBR), as the name indicates, is based on checklists containing questions that should be answered during the defect detection. These questions focus on certain quality aspects that are relevant for the requirements under inspection. The checklist approach tells an inspector what to check. However, an often cited weakness of CBR is that it provides little support for how to perform the analysis [34, 48]. The reviewers get no guidance or hints on how to answer the questions in the checklist.

A checklist for use cases, for example, is presented in [2]. Many other checklists for requirements can be found on the Internet. However, it is important to note that there exist no standard checklist that can be applied in all contexts. A checklist has to be company- and sometimes even project-specific. Thus, the checklist has to be tailored to the context and characteristics of the company and the project. It is important to consider the elements of the requirements quality framework as it provides input for defining valuable checklist questions (e.g. input

on quality goals, existing checklists (basic techniques), quality characteristics of importance, organizational restrictions, etc.). In addition, one should consider known defects or problems and, of course, expert knowledge, as further sources for checklists questions.

Checklists have three basic weaknesses [34]. First, the checklist questions are often extremely general. That is, concrete guidance on how to use the checklist is missing. Further, the checklist questions are often not up to date. To overcome these drawbacks, alternative approaches were developed. One class of alternative approaches is called scenario-based approaches. For requirements, the following scenario-based approaches are applicable: Perspective-based reading (PBR) [4, 34, 43, 45], traceability-based reading [45], defect-based reading [40] and usage-based reading [48].

The basic idea of the scenario-based reading techniques is that inspectors are guided by a scenario that tells them what to look for during the inspection and how to perform the inspection. Furthermore, the scenario guides the inspector to actively work with the requirements, resulting in a deeper understanding of the requirements and their interrelationships [34, 43]. Having such a deep understanding of the requirements is a prerequisite for finding more subtle and logical defects, which are often critical to the final system. Finally, the scenarios focus the attention of the inspectors on the essential quality aspects and on the essential parts of the requirements under inspection that need the most thorough investigation [34]. This input should be taken for example from prioritization techniques (see Chap. 5).

The special aspect of PBR is that the requirements are inspected from the viewpoint of different stakeholders, see Fig. 8.4. Different stakeholders have different interests in the requirements. The assumption behind PBR is that the requirements are of good quality if all stakeholders who use the requirements for their specific tasks, agree on the requirements quality (find no serious issues in them).

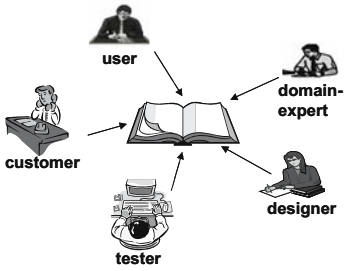

Fig. 8.4 Some perspectives to inspect the requirements

In each company context, the involved perspectives are different. Therefore, the first essential step when applying the PBR approach is identification of the potential perspectives and the quality concerns these perspectives are interested in.

During an inspection traceability links (see Chap. 5) can help to guide the inspectors through the requirements. For example, the quality attribute of consis-

tency (see Sect. 8.2.1) is directly related to the ability to trace one requirement to another. The problems with inconsistency are well documented and are often one reason for quality problems and project delays [35]. With well defined links between the requirements it is possible for the inspector to follow these links and check that the requirements work together in a consistent and correct way. In that sense, the defect detection step gets more efficient as the inspectors do not have to think of potential relationships between requirements but can follow the links between them. Beside the consistency issue, it is also possible for the inspector to judge whether certain functions are completely realized with the different requirements described in the specification by following the traceability links and judging whether the sum of the requirements results in the desired support for the user. Finally, the traceability links indicate requirements that are highly related to each other and therefore help the inspectors to judge the maintainability and understandability of the requirements.

But also without the support of traceability, inspections can address many of the quality attributes specified in Sect. 8.2.1 (assuming that the inspection is performed thoroughly): correctness, completeness, unambiguity, comprehensibility, feasibility, modifiability, verifiability. This can be achieved with the right set of questions in the reading scenarios and checklists.

8.4.2 Requirements-Based Testing

Testing is usually performed at the end of the development process when executable system parts are available. Test cases are usually defined and run on the system to validate whether the system fulfills its specification. For example, the test cases derived from the requirements are used during the acceptance and system test phase. Testing is often perceived as the pure execution of the test cases at the end of the development cycle. This perception has led to the myth that testing can start only at the end of the software development process [22]. However, testing is more than running the test cases and looking for failures in the final software. At least the two steps test planning and test case creation can and should be integrated in the development process much earlier than they are usually integrated.

It is recommended that test planning and test case creation should be performed as soon as the requirements, or a self-contained sub-set, are defined [22, 51]. The idea of early test case creation is similar to the idea of perspective-based inspections. Through the early construction of the test cases, the test engineers gain a better understanding of the requirements and are able to identify weaknesses and potential issues within the requirements. Moreover, test engineers bring in a completely new perspective on the requirements which also contributes to identify requirements issues during the early test case creation. For example, if the test engineers have difficulties in deriving the acceptance test case from requirements it might be necessary to refine the requirements, to add missing information or to remove/restate the requirement as it is not possible to test them.

The principle of early test case creation helps to improve the quality of the requirements by identifying correctness, completeness, ambiguity, consistency and

verifiability issues during the specification of the test cases. If this is done at the very end of the project, these issues are propagated from the requirements to all later phases and the test engineers might base their test cases on the wrong requirements, as the requirements are taken for granted (a fixed source of truth which is different at the beginning of the process).

An overview on requirements based testing approaches can be found in [15]. Special approaches that work on use cases are described in [6, 12, 30, 42]. General approaches that can be applied on the requirements specification to define detailed test cases are, for example, defined in [31, 39].

Again, it is possible to use traceability links to facilitate this activity (see Chap. 5). They provide a better understanding of which aspects in the requirements have to be tested together (e.g. in a test scenario) and which requirements are already covered by the defined test cases. Depending on the granularity of the traceability links, it is then possible to judge which requirements as a whole are covered by one ore more test cases, which test cases test more than one requirement, or whether there are test cases that cover only a single requirements. This information helps to identify points that need further consideration and special attention. Furthermore, traceability can help to select those parts that need regression testing by identifying which requirements are affected by a certain change [1, 33].

8.4.3 Automated Approaches and Formal Methods

Due to the abstract and informal nature of most requirements documents it is difficult to apply any automated tools to ensure their quality. For simple issues, such as grammar or spelling defects, there are tools available. Removing such issues from the requirements typically improve their comprehensibility.

For one quality attribute, unambiguity, more tool support is available. The idea of tools that address ambiguity flaws is the identification of certain patterns and keywords in the requirements that point to potential risk areas (i.e. areas where more than one interpretation of the requirements is possible). These tools identify, based on a glossary, phrases that are marked as weak or subjective, for example, “if possible”, “may”, “could”, “optionally”, etc. The tools parse the requirements document based on the pre-defined glossary and provide a list of all occurrences of the weak-phrases in the document [19, 52]. Even though the tools automatically detect certain quality issues in the requirements, the applicability of these tools in industrial practice has to be further investigated.

Further automation is possible when the requirements are defined in a formal way. The use of formal languages copes with requirements issues by avoiding the imprecise nature of natural language. Requirements are specified in a semantically well-defined way, typically mathematically based. Several benefits can be gained by using formal methods. The communication between the stakeholders is more precise, and thus, misunderstandings and ambiguities can be reduced. It is possible to check the completeness and the consistency of the requirements document, and automated proof of safety properties is possible. Finally, the requirements engi-

neer can perform simulations of the future system, when the language is supported by a tool. Examples for such languages are SCR [23], SDL [27], and VDM [29].

However, formal methods also have drawbacks. They are difficult to learn and difficult to understand for a person without the necessary background. Specifically, the customer is often not interested in learning the formal language, and a compromise needs to be found. The first version of the requirements might be formulated in natural language, in the language of the customer. The requirements must then be translated into the formal version.

8.5 Open Research Questions

Based on the current state of the practice, some open questions with respect to quality assurance in the requirements engineering phase are identified. First, some open issues with respect to testing are discussed and afterwards, inspections are further elaborated on.

The survey on existing approaches for early test case creation and the involvement of testing during requirements engineering reveals that there are many promising approaches and that the need of early tester involvement is clearly recognized. However, the survey also shows that there is a lack of empirical evidence that the proposed approaches do, in fact, save money, improve the quality of the requirements, and to improve the overall system quality by means of better acceptance and system test cases that are more related to the requirements. Future research should focus on gathering this data as these results are important to transfer the approaches into industry (convince practitioners of the benefits).

Related to this aspect is the fact that test case creation for system and acceptance testing is performed without the involvement of the final system user. Almost all of the research papers explicitly mention that the user should be involved during test case creation but do not state how this should be done. Here, research is necessary to define ways to involve the system end users in this process in a most efficient and beneficial way.

Many test case creation approaches provide only little guidance on how to derive the test cases from the requirements or intermediate models of the requirements (e.g. sequence or state charts that represent the requirements). Often, there are only high level descriptions on how to come up with good test cases. Therefore, more research activities should focus on the relationships between requirements and other artifacts such as certain types of models and, of course, test cases. Guidelines that provide a stepwise approach on how to derive intermediate models should be defined to further facilitate the test case creation activity and, of course, any further development steps (e.g., analysis and high level design would also benefit from such guidelines).

Finally, it is a common fact that within requirements specifications, more and more various notations are used (e.g., pure text, tables, use case diagrams, sequence diagrams, etc.). How to deal with this variety of notations during test case

creation is an unresolved question. Each notation provides relevant input and has to be considered during test case creation.

Concerning requirements inspections, most of the above-mentioned open issues with respect to requirements based testing also apply to inspections. Approaches for inspecting heterogeneous requirements documents need to be developed, and a process needs to be defined on how to most efficiently integrate the various stakeholders of the requirements in the inspection process. Here, it is especially important to define decision support for inspections that gives guidance on when to include which stakeholders (e.g., when to include which perspectives) and when it is necessary to perform which process steps. More research is therefore needed that investigates the factors influencing a good inspection process and to develop guidelines to customize the inspections in an optimal way to the quality needs that should be addressed during requirements engineering.

One part of this decision support should be a guideline on which inspection technique (checklist, scenario-based reading, including usage based reading, perspective-based reading, defect-based reading, etc.) should be used to verify the requirements. Past research focused only on the question of which of the techniques outperforms the other technique with respect to efficiency and effectiveness of the inspection process. The more relevant question seems to be how the different reading techniques should be combined to gain a more efficient inspection of the requirements. Thus, one should address the question of which of the reading techniques is more suited in detecting certain types of requirements issues. A second part of such a customization approach should be a guideline that provides hints on which questions or reading scenarios should be used during the inspection in order to address certain quality issues in a most efficient way. Therefore, it is up to future research to investigate what kind of inspection questions (for requirements) have an impact on which qualities the customer is interested in.

Finally, the question of tool support for inspections should be further addressed. With respect to requirements inspections, it should be investigated which quality issues could be automatically checked by a tool (e.g., application to certain structural restrictions) and how the inspection of other quality aspects could be facilitated, e.g. by providing support in checking certain checklist questions.

The most important open question that should be addressed in future research activities is how the different quality assurance approaches (constructive and analytic ones) of the requirements engineering phase can be combined into a comprehensive quality assurance strategy. There are some initial results that address this question [13], but these results need to be further investigated. In [9], it is stated that different quality assurance approaches help to address different quality issues. Unfortunately, neither the state of art nor the state of the practice can explicitly state which approaches are most efficient to address which quality issues, i.e. it is important to evaluate which of the approaches is more effective and consumes less effort in addressing certain requirements quality issues. In other words, future research has to investigate which qualities of the final system and the requirements are most efficiently assured by means of which approach (constructive, testing, inspections). Here, especially, more research is needed on the impact of applying constructive approaches such as certain elicitation techniques or specification

techniques on the quality of the requirements. A second important step is the definition of external or system quality characteristics that should be addressed (e.g. safety, security, reusability, maintainability, etc.) and how these qualities manifest in the requirements. If this connection can be drawn, it is possible to customize the different QA approaches in that way that they focus on those system qualities that are most relevant for the customer.

These are all cornerstones that need to be investigated for the definition of a requirements quality strategy. And we should force our efforts as a well-defined requirements quality strategy would help to minimize the costs for quality assurance and, in parallel, increase the effectiveness and efficiency of the different approaches.

8.6 Conclusion

Quality is an elusive but important subject for requirements, especially since the quality of the requirements will more or less affect all other artifacts in the development. This chapter presents ideas on a framework for quality assurance (QA) in the requirements phase. The framework describes a set of attributes that are used to define quality. In addition, the framework describes what has to be considered when defining a QA strategy, to achieve the defined quality characteristics of requirements and requirements documents.

Further, an overview of state of art constructive and analytical QA approaches is presented. The theoretical contribution consists of an overview of state of the art QA approaches for requirements, as well as a more detailed description of a selected set. The QA approaches addressed are inspections, test case creation, and the impact of elicitation specification, and prototyping on quality. Moreover, some initial ideas are sketched on when to apply a specific type of QA approach by means of a mapping the QA approach to requirements quality characteristics.

Looking at the state of the art, it is clear that there are certain gaps in our understanding of how high quality requirements can be achieved and how the costs of the QA activities on the requirements affect the cost of the rest of the development. It is, however, a fair amount of research performed on individual QA approaches, but the combination and wider effects need more investigation.

Acknowledgements

This work has been partly supported by the ForPICS project, funded by the Provincia Autonoma di Trento, Italy. The authors also would like to thank the anonymous reviewers, as well as colleagues, specifically Sonnhild Namingha, for helpful comments on draft version of this chapter.

References

1. Ahlowalia N (2002) Testing from use cases using path analysis technique. In: Proceedings of the International Conference on Software Testing, Analysis & Review, Anaheim, CA, USA
2. Anda B, Sjøberg D I K (2002) Towards an inspection technique for UC models. In: Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE), Italy, pp 127–134
3. Aurum A, Petersson H, Wohlin C (2002) State-of-the-Art: Software Inspections Turning 25 Years. Journal on Software Testing, Verification and Reliability 12(3): 133–154
4. Basili V R, Green S, Laitenberger O, Lanubile F, Shull F, Sorumgard S, Zelkowitz M (1996) The empirical investigation of perspective-based reading. Empirical Software Engineering 1(2): 133–164
5. Beck K (1999) Extreme programming explained. Boston: Addison-Wesley
6. Binder RV (1999) Testing object-oriented systems: Patterns, models and tools. Boston: Addison-Wesley Object Technologies Series
7. Briand L, Freimut B, Vollei F, (2000) Assessing the cost-effectiveness of inspections by combining project data and expert opinion. In: Proceedings of the 11th International Symposium on Software Reliability Engineering, pp.124–135
8. Bittner K, Spence I (2003) Use case modeling. Boston: Addison-Wesley
9. Boehm BW, Gray TE (1984) Prototyping versus specifying: A multi-project experiment. IEEE Transaction on Software Engineering 10(3):290–302
10. Boehm BW, Basili VR (2001) Software defect reduction top 10 list. IEEE Computer 34(1):135–137
11. Cockburn, A (2001) Writing effective use cases. Boston: Addison-Wesley
12. Collard R (1999) Test design: Developing test cases from use cases. Software Testing and Quality Engineering July/August 1(4): 31–36
13. Denger C, Paech B (2004) An integrated quality assurance approach for use case based requirements. In: Proceedings of the German conference of Modellierung, pp.59–74
14. Denger C, Paech B, Benz S (2003) Guidelines -- Creating use cases for embedded systems. IESE-Report, 078.03/E, Kaiserslautern, Germany
15. Denger C, Medina M (2003) Test cases derived from user requirements specifications: Literature survey. IESE Report No. 033.03/E, Kaiserslautern, Germany
16. Denger C, Kerkow D, Knethen Av, Paech B (2003) A comprehensive approach for creating high-quality requirements and specifications in automotive projects. In: Proceedings of the International Conference Software and Systems Engineering and their Applications, 2-6 December, Paris, France
17. Endres A, Rombach H D (2003) A handbook of software and systems engineering. Empirical Observations, Laws and Theories. New York: Addison-Wesley
18. Fagan ME, (1976) Design and code inspections to reduce errors in program development. IBM Systems Journal 15(3):182–211
19. Fantechi A, Gnesi S, Lami G, Maccari A (2002) Application of linguistic techniques for use case analysis. In: Proceedings of the International Conference on Requirements Engineering, pp 157-164, Essen, Germany
20. Gilb T, Graham D (1993) Software inspection. Boston, Addison-Wesley
21. Glass RL (1998) Software runaways. Lessons learned from massive software project failures. Upper Saddle River, NJ: Prentice Hall

22. Graham D (2002) Requirements and testing: Seven missing-link myths. IEEE Software 19(9):15-17
23. Heitmeyer CL, Jeffords RD, Labaw BG, (1996) Automated consistency checking of requirements specifications. ACM Transactions on Software Engineering and Methodology 5(3): 231–261
24. IEEE Recommended practice for software requirements specification. IEEE Standard 830-1998, 1998
25. IEEE guide for developing system requirements specification. IEEE Standard 1233-1998, 1998
26. IEEE standard for software reviews. IEEE Standard 1028-1997, 1997
27. ITU-T (1993) Recommendation Z.100. Specification and description language (SDL) ITU-International Communication Unit, Geneva
28. Jalote P (1989) Testing of completeness of specifications. IEEE Transactions on Software Engineering 15(5): 526–531
29. Jones CB, (1990) Systematic software development using VDM. Upper Saddle River, NJ, Prentice Hall
30. Kamsties E, Pohl K, Reis S, Reuys A (2004) Szenario-basiertes systemtesten von software-produktfamilien mit ScenTED. In: Proceedings of Modellierung, Marburg, Germany, pp.169–186
31. Keese P, Meyerhoff D (2003) Tutorial on requirements-based testing (SQS). Held in conjunction of the International Conference on Software Testing, Cologne, Germany
32. Kitchenham B, Pfleeger S (1996) Software quality: the elusive target. IEEE Software, 13(1): 12–21
33. Le Lann G (1996) The Ariane 5 Flight 501 Failure - A case study in system engineering for computing systems. Research report RR-3079, INRIA
34. Laitenberger O (2000) Cost-effective detection of software defects through perspective-based inspections. PhD Thesis in Experimental Software Engineering; Fraunhofer IRB Verlag
35. Leffingwell D, Widrig D (2000) Managing software requirements – A unified approach. Boston: Addison-Wesley
36. Liggesmeyer P (1990), Modultest und modulverifikation – State of the art. Mannheim, Wien Zürich, BI-Wissesverlag
37. Briand LC, Freimut B, Vollei F (2000) Assessing the cost-effectiveness of inspections by combining project data and expert opinion. In: Proceedings of the 11th International Symposium on Software Reliability Engineering, pp 124–135
38. Musa J (1993) Operational profiles in software-reliability engineering. IEEE Software 10:(2): 14–32
39. Ostraned T J, Balcer M J (1988) The category-partition method for specifying and generating functional tests. Communications of the ACM 31(6):676–686
40. Porter A, Votta LG (1998) Comparing detection methods for software requirements specification: A replication using professional subjects. Empirical Software Engineering 3(4): 355–379
41. Robertson S, Robertson JH (1999) Mastering the requirements process. Boston: Addison-Wesley
42. Rupp C, Queins S (2003) Vom use-case zum Test-Case. OBJEKTspektrum, Vol.4
43. Shull F, Rus I, Basili V (2000) How perspective-based reading can improve requirements inspections. IEEE Computer 33(7):73–79

44. Shull F, Basili V, Boehm B, Brown AW, Costa P, Lindvall M, Port D, Rus I, Tesoriero R, Zelkowitz M (2002) What we have learned about fighting defects. In: Proceedings of 8th International Metrics Software Metrics Symposium: p 249ff., Ottawa, Canada
45. Shull F, Travassos G H, Carver J (1999) Evolving a set of techniques for OO inspections. Technical Report CS-TR-4070, UMIACS-TR-99-63; University of Maryland
46. Sommerville I, Sawyer P (1997) Requirements engineering. A good practice guide. Chichester: John Wiley & Sons
47. http://www.standishgroup.com/chaos_chronicles/index.php Accessed on 3rd December 2004
48. Thelin T, Runeson P, Wohlin C (2003) An experimental comparison of usage-based reading and checklist-based reading. IEEE Transactions on Software Engineering, 29(8): 687–704
49. Wiegers K E (2002) Peer reviews in software. A practical guide. Boston: Addison-Wesley
50. Wiegers K E (1999) Writing quality requirements. Software Development Magazine, 7(5): 44–48
51. Wiegers K E (2000) Karl Wiegers describes 10 requirements traps to avoid. Software Testing & Quality Engineering Journal, January/February, 2(1)
52. Wilson WM, Rosenberg LH, Hyatt LE (1996) Automated quality analysis of natural language requirements specifications. NASA Software Assurance Technology Center, USA

Author Biography

Christian Denger studied computer science at the University of Kaiserslautern, Germany, with a minor in economics. He received his master in computer science in 2002. Since then, he has been working as a scientist at the Fraunhofer Institute for Experimental Software Engineering in Kaiserslautern, Germany. His research interests are software inspections in the context of defect cost reduction approaches in early development phases and the combination of quality assurance techniques in the context of embedded systems. Currently, he is involved in several German and international projects as a team member and project leader and is pursuing a PhD degree at the University of Kaiserslautern.

Thomas Olsson works as a scientist at the Fraunhofer Institute for Experimental Software Engineering in Kaiserslautern, Germany. He received a Licentiate of Engineering in Software Engineering in 2002 and a Master of Science in Computer Science and Engineering in 1999, both from Lund University, Sweden. His research interests lie in heterogeneous information and documentation models, especially in the context of requirements. Currently, he is leading one European and one German project, and is at the same time pursuing a PhD degree at Lund University.

Part 2
The Next Practice in Requirements Engineering

This part provides descriptions of some specific ways of addressing the challenges in requirements engineering as well as presenting various areas where requirements engineering plays a key role in the success of a software project. There are seven chapters in this Part. Chapter 9 addresses the possibility of using goal modeling in requirements engineering and, in particular, how to reason with goals. Chapter 10 recognizes that software requirements are often represented in natural language, which results in some challenges when it comes to the management of large repositories of requirements. Natural language also raises the challenge of overcoming ambiguity in the wording of requirements. Chap. 11 presents an introduction and some empirical results in relation to ambiguity. Part I has established that decision-making is an important aspect of engineering and managing requirements. Thus Chap. 12 is devoted to decision-support. Requirements engineering is all too often focused on bespoke software development. In many cases, software is developed for markets. A market-driven approach to requirements engineering is presented in Chap. 13. Software development methods evolve overtime. One such family of methods is agile methods. The handling of requirements within agile development is presented in Chap. 14. Finally, requirements engineering in a web-based context is presented in Chap. 15.

Thus, in summary, this part contains chapters on the following topics:

- Chapter 9: Goal modeling
- Chapter 10: Use of natural language
- Chapter 11: Ambiguity in requirements
- Chapter 12: Decision support
- Chapter 13: Market-orientation
- Chapter 14: Agile methods
- Chapter 15: Web-based development

These seven chapters highlight some of the main issues related to engineering and managing software requirements. The chapters are written by researchers from around the world that have conducted extensive and reputable research in the above areas.

The seven chapters are by Collette Rolland and Camille Salinesi from University of Paris, France; Johan Natt och Dag from Lund University, Sweden and Vincenzo Gervasi from University of Pisa, Italy; Erik Kamsties from University of Essen, Germany; An Ngo-The and Günther Ruhe from University of Calgary, Canada; Björn Regnell from Lund University, Sweden and Sjaak Brinkkemper from Utrecht University, The Netherlands; Alberto Sillitti and Giancarlo Succi from the Free University of Bozen, Italy; Jacob L. Cybulski from Deakin University, Australia and Pradip K. Sarkar from Central Queensland University, Australia.