Experiences on analysis of requirements quality

Experiences on Analysis of Requirements Quality

Petra Heck
Consultant and Researcher
Laboratory for Quality Software (NL)
info@laquso.com, petraheck@hotmail.com

Päivi Parviainen
Senior Research Scientist
VTT Technical Research Centre of Finland
Paivi.Parviainen@vtt.fi

Abstract

The quality of any product depends on the quality of the basis of making it, i.e., the quality of the requirements has strong effect on the quality of the end products. In practice, however, the quality of requirement specifications is poor, in fact a primary reason why so many projects continue to fail. Thus, the current approaches as applied in practice are clearly not enough to develop high quality requirements specifications. Also, the poor quality of the requirements is typically not recognized during requirements development. In this paper we present a method called LSPCM developed for certifying software product quality. We also describe experiences from using the method for analyzing requirements quality in three cases. The three different cases show that the checks in the LSPCM are useful for finding inconsistencies in requirements specifications, regardless of the application domain.

1. Introduction

The importance and effect of software intensive systems increase continuously as more and more applications depend on the reliability, availability, and integrity of software. These systems are also becoming more and more complex, causing that the development of quality systems has become a major scientific and engineering challenge. The quality of the systems is strongly affected by the quality of the requirements but proper requirements engineering is not an established practice within the software developing community [1]. This causes that most errors are introduced in the requirements phase and are caused by poorly written, ambiguous, unclear or missed requirements, as reported by several studies [1, 2, 3]. Failure to correctly specify the requirements can lead to major delays, cost overruns, commercial consequences including the loss of money or property, layoffs, and even the loss of lives.

Requirements engineering (RE) is generally accepted to be the most critical and complex process within the development of embedded systems, see, for example, [4, 5, 6]. One reason for this is that in requirements engineering the most diverse set of product demands from the most diverse set of stakeholders has to be considered, making the requirements engineering process both multidisciplinary and complex.

In this paper we focus on verification of the requirements, meaning checking the quality of the requirements descriptions. We describe a method developed for certifying software product quality, focusing on the requirements part of the method. We also describe experiences from using the method in three cases in different application domains.

1.1. Requirements Quality

According to the IEEE Guide for Developing System Requirements Specifications [7], a well-formed requirement is "a statement of system functionality (a capability) that can be validated, that must be possessed by a system to solve a customer problem or to achieve a customer objective, and that is qualified by measurable conditions and bounded by constraints". Capabilities are the fundamental requirements of the system, representing the features or functions needed by the stakeholders. Conditions and constraints are attributes that are stipulated by a capability.

When requirements are expressed in natural language, descriptions should be written by using simple and concise language in order for them to be understandable for all stakeholders. However, in practice the quality of requirement specifications is poor, the requirements are ambiguous, incomplete, unverifiable, inadequately prioritized, and mutually inconsistent [8]. In fact, this poor quality of the requirements (incomplete and changing requirements) is a primary reason why so many projects continue to fail [9]. Thus, the current approaches as applied in practice are clearly not enough to develop high quality requirements specifications. Furthermore, the poor quality of the requirements is typically not recognized during requirements development; the requirements may be reviewed, but many of the defects are not found, causing that the defects replicate in the following work. The later these errors are found, the

978-0-7695-3372-8/08 $25.00 © 2008 IEEE
DOI 10.1109/ICSEA.2008.32

more impact they have on work products quality and the more costly they are to fix, i.e., according to Boehm [10] finding and fixing a software problem after delivery is often 100 times more expensive than finding and fixing it during the requirement and design phase.

1.2. Related Work

Many checklists and questionnaires for ensuring the quality of requirements of varying degrees of completeness and usefulness are described in requirements engineering books [1, 11, 12, 13]. There are also several guidelines and standards published addressing the requirements quality [7, 14, 15, 16]. Thus, the theory of writing high quality requirements is well established. However, in practice, the requirements are developed by people who have had little or no training in requirements engineering [8], often lacking knowledge of the published theory. Also, evaluating requirements quality based on standards is not straightforward, as understanding the practical meaning of the terms used in the standards is not simple. There are also some tools that can help in the analysis of the quality of requirements written in natural language such as ARM [17], CICO [18] and QuARS [19]. These tools analysis is based on predefined quality criteria and elements of (English) language.

The LaQuSo Software Product Certification Model (LSPCM) was developed because we did not find any other practical method or framework for checking the quality of software artifacts, e.g., requirements specifications. We needed a method that would yield similar results regardless of who performed the quality check and which is also application domain independent. The framework is based on literature and own experiences, main new contribution being that the framework brings together information that was scattered over different sources.

2. LSPCM Overview

For the certification of software product quality the Laboratory for Quality Software (LaQuSo) has developed the Software Product Certification Model (LSPCM) [20]. Benefit from this certification is for example that the issues found during certification directly point to issues in the software product. Solving issues leads to a higher quality product. Another benefit of the certification is that in the certification the software product is investigated on a number of predefined criteria. These same criteria can later be used during the development of new products. In this way the organization learns from the certification: what is a good software product and how can we check our software products ourselves? The LSPCM has a similar structure as the well known models GQM (Goal Question Metric) and FCM (Feature Criteria Metric), as requirements are certified according to certification criteria (= Goal in GQM or = Feature in FCM), categories (= Question in GQM or = Criteria in FCM) and checks (= Metric in both GQM and FCM).

Software artifacts include in addition to the final software product (the source code or working system), also intermediate deliverables like user requirements and detailed design. Each major deliverable is a **Product Area** in the LSPCM. Each area can be further divided into subparts, called elements. These elements can be separate artifacts, a chapter within a document, or different parts of a larger model. For instance, the user manual will be a separate artifact delivered with the system, the non-functional requirements will be a separate section of the user requirements document, and the stakeholders can be described as part of the business process description.

Certification Criteria (CC) are criteria that apply to each Product Area (PA). For each of the CC's different Achievement Levels (AL) can be established. There are three CC, generic for all Product Areas:

- *Completeness (CC1):* All required elements in the Product Area (see section 2.2 for examples) should be present and as formalized as possible. The completeness of a PA can be basic or extra elements may have been added. These elements can be semi-formal (AL2) or formal (AL3), which refers to the fact that they are specified in a formal language. The more formal an element is, the easier it can be subject to formal verification (less transformation is needed).
- *Uniformity (CC2):* The style of the elements in the PA should be standardized. The uniformity of a PA can be only within the PA itself (AL1), with respect to a company standard (AL2), or with respect to an industry standard (AL3), meaning a general accepted description technique that is not specific for the company like the use of UML diagrams for design documents.
- *Conformance (CC3):* All elements should conform to the property that is subject of the certification. The conformance of the PA to a property can be established with different means that increase confidence: manually (AL1), with tool support (AL2), or by formal verification (AL3).

For each of these Certification Criteria different Achievement Levels can be established and the Certification Criteria can be translated into Specific Criteria (SC) per Product Area that indicate what completeness (CC1), uniformity (CC2), and conformance (CC3) mean for that Product Area. The Specific Criteria indicate what the required elements and checks are to achieve a certain level of the Certification Criteria. In this article we focus on the User Requirements Product Area of the LSPCM and the specific criteria for it.

2.1. Analysis Process

This section explains the general idea behind the LaQuSo Model, more details can be found in [20].

Although the current goal of the model is to hand out certificates to customers, its framework and checks can just as well be used by anyone that needs to check the quality of requirements. Next we will explain how LaQuSo hands out certificates.

If an organization wants confidence in a software artifact a LaQuSo certificate can be requested. Being part of universities, LaQuSo is an independent evaluator. Certification is a check that the artifact fulfils a defined set of properties and the certificate will always refer to the properties that were used. The certificate covers only the product quality, not the management and development processes. The quality certificate consists of a diagnosis report and verdict document. Certification may also be an iterative process where the customer delivers improved versions of an artifact until a certificate is achieved. Each certification case has its own goals and a specific certification plan (steps to take and techniques to apply) is made for each project. In this paper we describe three projects in which the goal was to grant a "Consistency of User Requirements" certificate. This certificate has the following characteristics:

Table 1. Certificate characteristics

Product Area	User Requirements
Properties	Consistency
Level	Manually verified
Description	Check on the internal consistency of the requirements
Input	Natural language requirements specification

Next, we'll describe the elements we consider to be part of a complete user requirements specification:

- *Functional requirements or Use-cases.* Functional requirements describe the functionality of the system from the perspective of the user in plain text or in the form of use-cases. A use-case is a named "piece of functionality" as seen from the perspective of an actor. For each use-case several use-case scenario's are given (sequences of actions, events or tasks), the permitted or desired ones as well as the forbidden ones.
- *Objects.* Many types of entities play a role in the environment processes and some of them have to be represented in the system. Only these are collected. The object description can be informal in the form of a glossary, or more advanced in the form of a data dictionary or object model.
- *Behavioral Properties.* General behavioral properties that should hold, such as properties expressing that certain conditions may never occur or that certain conditions should always hold. Usually these properties have a temporal aspect and therefore it is possible to express them in temporal logic, although a translation in natural language is essential for most stakeholders.
- *Non-functional requirements.* A set of different types of quality attributes that can be used to judge the operation of a system.

2.2. Requirement Checks

In this section we suggest a necessarily incomplete list for the User Requirements PA. The specific criteria (SC) are a direct translation of the three general certification criteria to the user requirements PA: describe the requirements as detailed and as formal as possible (CC1 -> SC1), comply with requirements engineering standards (CC2 -> SC2), and maintain correct and consistent relations between the elements in the requirements description (CC3 -> SC3a) and with the context analysis (CC3 -> SC3b). From this set of elements and the specific criteria we derive a number of specific checks (based on literature, standards and our own experience).

Table 2. The requirement checks

Category	Check
SC1.1 Required Elements	• Functional requirements describe the functionality of the system from the perspective of the user. • Non-functional requirements i.e., quality requirements are described (e.g., performance and security measures). • Glossary defines the entities that have to be represented in the system.
SC1.2 Semi-formal Elements	• Data dictionary or object model that contains data elements' definitions and representations including semantics for data elements. The semantic components focus on creating precise meaning of data elements. • Use cases (with scenarios) are defined and for each use case several use case scenarios are given (sequences of actions, events or tasks), the permitted or desired and the forbidden ones. • Behavioural properties, i.e., the constraints on the behaviour of the system are defined. The constraints express that e.g. certain conditions may never occur or certain conditions should always hold. Usually behavioural properties have a temporal aspect.
SC1.3 Formal Elements	• Relational diagram of data/object model is used to describe conceptual data models by providing graphical notations for entities and their relationships, and the constraints that bind them. Examples of diagramming techniques are UML class diagram and ER-diagram • Process model of use case scenarios describe the relations between the steps in the use case scenarios in a formal language like Petri Nets. • Behavioral properties specifications are expressed in a formal language. If they e.g. have a temporal aspect, temporal

	logic can be used.
SC2.1 Compliance w. Industry Standards	• ERD diagram for object/data model; • UML diagrams for use cases.
SC3a.1 Internal Correctness	• No two requirements or use cases contradict each other. • No requirement is ambiguous. • Functional requirements specify what, not how. • Each requirement is testable. • Each requirement is uniquely identified. • Each requirement is atomic. • The glossary definitions are non-cyclic. • Use case diagrams correspond to use case text.
SC3a.2 Automated Correctness Checks	• Requirements are stored in a requirements management tool which uniquely identifies them.
SC3a.3 Formally Verified Correctness	• Verify use case scenario models for e.g. correct workflow (no deadlocks or dead tasks) and mutual consistency. • Check data model diagram for normal form.
SC3b.1 External Consistency	• Each ambiguous or unclear term is contained in the glossary. • The use cases or functional requirements are a detailing of the environment description in the context analysis (no contradictions). Each step in a business process that involves the system has been included. Each task that the system should fulfil for its environment has been included. All actors of the context analysis have been included in the requirements. • Each object is mentioned in the requirements and all objects mentioned in the requirements are contained in the object model. • The requirements do not contradict the behavioural properties. • The use case or functional requirements do not conflict with the non-functional requirements.
SC3b.2 Automated Consistency Checks	• Requirements and glossary/objects are stored in a requirement mgmt tool showing the requirements, scenarios, actors, and objects relations.
SC3b.3 Formally Verified Consistency	• Verify use case scenario models for e.g. compliance with behavioral properties and non-functional requirements. • Verify that the requirements description complies with the environment description from the context analysis.

3. Case experiences

In this section we describe three cases where the method has been used to analyse requirements. For each of the cases, the findings are presented on a general level, including a reference to the check category that resulted in the finding.

3.1. Central Registration System

The system is a central point where new identification numbers are generated, distributed and registered. We were asked to judge the quality of the functional design, which consisted of functional requirements, 15 use case descriptions with UML activity diagrams, a process model of the business processes, a functional architecture (logical module structure), an object model, a glossary and a supplementary specification (all non-functional requirements such as legal, security, performance etc.).

The following types of inconsistencies were found:

- A number of spelling and structural errors were found. [SC3a.1]
- Some post-conditions of use cases were not consistent with the main scenario. [SC3a.1]
- Activity diagrams did not use the correct (UML) symbols: e.g. included states as activities. [SC2.1]
- The object model did not use ERD symbols correctly and did not contain much description for the attributes. [SC1.3 and SC2.1]
- The glossary contained only abbreviations. [SC1.1]
- The activity diagrams did not always match the use case text (especially not for the alternative flows). [SC3a.1]
- One of the actors was not used in a consistent manner (mix between human and nonhuman). [SC3a.1]
- One use case mentions two options in the summary and illustrates only one in the scenarios. [SC3a.1]
- Use cases described system features that were not mentioned in the other documents. [SC3b.1]
- There was an overview document that did not contain all use cases and their relations. [SC3b.1]
- Some components to support the use cases were missing in the functional architecture. [SC3b.1]
- Use cases for administration functions such as user management were missing. [SC3b.1]

All major inconsistencies were solved before the design was handed over to the developers of the system. This minimized the input needed from the designers during the development phase and reduces the risk for confusion and misinterpretation.

3.2. Counter Automation Solution

The company operates many offices with multiple counters were customers come for various transaction types. The system is a central registration system for all

counter transactions that take place. Client applications support the counter workers. We were asked to judge the quality of the functional design, which consisted of 200+ use case descriptions with flow charts, supplementary specifications (all requirements that were not specific to one single use case) and a glossary.

The following types of inconsistencies were found:

- There were around 200 use cases without a clear overview of the use cases and their relations. [SC1.2]
- Use case structures were not always applied correctly. E.g. the name of a use case should be noun + verb and a use case should have a clear trigger. [SC2.1]
- Paragraphs in use cases that summarize information from the rest of the documents did not always include all relevant items. [SC3b.1]
- Not all referenced documents were included in the baseline. [SC1.1]
- There were many open points ("to be decided") in the documents. [SC3a.1]
- The overview and reference documents in the supplementary specifications were not consistent with the separate use cases. [SC3b.1]
- Many times it was not clear if 'N/A' (not applicable) was correctly filled in. [SC3a.1]
- The content of a document or paragraph did not always match its purpose. [SC3a.1]
- Not all use case documents had the same layout. [SC3a.1]
- Flowcharts were not always consistent with the use case text. [SC3a.1]
- The glossary did not contain all terms and missed a clear list of translations English – Dutch. [SC1.1]
- Many system management functions were not specified. [SC3b.1]
- The only structure in the document set was sequentially numbered per release (1 till 5) [SC3a.1]

With so many inconsistencies the document set was not suitable for system maintenance. It was not possible or very time-consuming to assess the impact of future change requests. The document set was adjusted according to the findings as much as time and budget permitted. The rest of the findings were used as cautions in the use of the documents for implementation of change requests.

3.3. Embedded Systems Case

The target of the analysis was system requirements for a large embedded system. This was also the first time the method was used by some one not familiar with the method beforehand.

The provided material included over 200 requirements. As the amount of requirements was high, not all of them could be checked according to the check list. Instead a sub-set of requirements was selected for the analysis. Altogether the analysis covered about 50 requirements. Requirements were selected randomly individually and as all requirements belonging to a feature.

The LSPCM method had so far been used mainly for software-only information systems and certain documentation was expected (user requirements, software requirements, high-level design). In this case, the checks defined by the method needed to be tailored. In practice, a combination of the checks for user requirements and high level design was done, however, not all checks of high-level design were included (as they were design issues). Also, in addition to the aspects indicated by the method, two additional checks were added based on discussion with the company's representative, namely stakeholders and acceptance criteria.

The following types of inconsistencies were found:

- Requirements relations / references to each other were unclear. [SC3a.1]
- No non-functional requirements were defined. [SC1.1]
- Various different ways to describe the requirements, from one line descriptions to several pages of use case steps, sequence charts, etc. [SC2.1]
- Lot of unclear abbreviations were used. [SC1.1, SC3a.1, SC3b.1]
- Not defining "what" but also a lot of "how", i.e., lot of design level issues. [SC3a.1]
- Ambiguities in requirement definitions. [SC3a.1, SC3b.1]
- No stakeholders defined for the requirements. [SC3a.1]

4. Discussion and Lessons Learnt

The three different projects show that the checks in the LSPCM are useful for finding inconsistencies in requirements specifications, regardless of the application domain. The checks that mostly found issues were the manual level checks [SC1.1, SC2.1, SC3a.1 and SC3b.1], as there were little formal elements defined for the requirements in all three cases. That some projects need tailoring of the LSPCM is already foreseen in the model; one could argue that the LSPCM is more a general framework to define specific checks per project than a rigid model. Within this general framework one can predefine rigid certificates by combining different checks.

As expected the three projects all lead to new general checks to be added to the list that is already included in the LSPCM. The idea is that this list will continue to grow with new experiences, literature and standards appearing.

The projects also taught us that requirements validation remains expert work. It is hard to explain in written text how to do this or what to look for. The idea of the LSPCM is to give some guidance to evaluators, but more details and examples need to be included in

the future to make it usable for people not so experienced in requirements engineering. For example, the statement "each ambiguous term is included in the glossary" will lead to different scores by each assessor, but at least it gives them a hint to pay attention to the link between the requirements text and the glossary.

For some of the projects it was necessary to first make a translation of the input material to the elements as described in the model. Each requirements document in practice will have a different structure. This means the start of each project needs to consist of creating a picture of the structure of the input requirements specification: what are the elements in the structure and what are their interrelations?

Another remarkable finding is that the better the quality of the input requirements, the more inconsistencies we can find. With really poorly written requirements, it is best to start again at the beginning working with the original analyst to identify the different elements in the specification and reconstruct them in a new document.

5. Conclusions

We have presented a method called LSPCM and experiences of using the method for certifying software product quality focusing on the requirements part of it. We have described experiences from using the method in three cases and described the general findings relating to the requirement descriptions quality. The three projects show that the checks in the LSPCM are useful for finding inconsistencies in requirements specification, regardless of the application domain. However, it is clear that requirements verification and validation remains expert work. The idea of the LSPCM is to give some guidance to evaluators, but more details and examples need to be included in the future to make it usable by people not so experienced in requirements engineering. That will be the focus of our future work.

References

[1] Firesmith, D. G., "Quality Requirements Checklist", in Journal of Object Technology, vol. 4, no. 9 November - December 2005, pp. 31 - 38, http://www.jot.fm/issues/issue_2005_11/column4

[2] Martin, J, 1984, An Information Systems Manifesto, Prentice Hall

[3] Damian D. 2002. The study of requirements engineering in global software development: as challenging as important. In Proceedings of Global Software Development, Workshop #9, organized in the International Conference on Software Engineering (ICSE) 2002, Orlando,FL.

[4] Komi-Sirviö, S, & Tihinen, M., 2005, Lessons learned by participants of distributed software development, Knowledge and Process Management, vol. 12, no. 2, pp. 108-122

[5] Juristo, N., Moreno, A.M., and Silva, A.A. 2002. Is the European Industry Moving Toward Solving Requirements Engineering Problems? IEEE Software 19(6): 70-77.

[6] Siddigi, J. 1996. Requirement Engineering: The Emerging Wisdom. IEEE Software 13(2): 15-19.

[7] "IEEE Std 1233 1998 Edition, Guide for Developing System Requirements Specifications," The Institute of Electrical and Electronics Engineers, Inc. 1998.

[8] Donald G. Firesmith: "Specifying Good Requirements", in Journal of Object Technology, vol. 2, no. 4, July-August 2003, pp. 77-87. http://www.jot.fm/issues/issue_2003_07/column7

[9] The Standish Group International, Inc. The CHAOS Report, published on www.standishgroup.com 1996, 1998, 2000, 2002, 2004 and 2006.

[10] Boehm, 2001 B. Boehm, Software defects reduction top 10 list, IEEE Computer 34 (2001) (1), pp. 135–137.

[11] R. R. Young, The Requirements Engineering Handbook. Norwood, MA: Artech House, 2004.

[12] K. E. Wiegers, Software Requirements, 2nd edition. Redmond, Washington: Microsoft Press, 2003.

[13] Ian Sommerville and Pete Sawyer: Requirements Engineering: A Good Practices Guide, John Wiley & Sons, 1997.

[14] IEEE Std 830 1998 Edition, IEEE Recommended Practice for Software Requirements Specifications, The Institute of Electrical and Electronics Engineers, Inc. 1998.

[15] IEEE Std 1061-1998, IEEE Standard for a Software Quality Metrics Methodology [23]

[16] IEEE Std 1362-1998, IEEE Guide for Information Technology System Definition Concept of Operations (ConOps) Document [1]

[17] ARM, http://satc.gsfc.nasa.gov/tools/arm/, Wilson, W. M., Rosenberg, L.H. & Hyatt, L. E., Automated Quality Analysis Of Natural Language Requirement Specifications, NASA, SATC

[18] CICO, http://circe.di.unipi.it/Cico/ e.g., Gervasi, V., & Nuseibeh, B., Lightweight validation of natural language requirements, Software: Practice & Experience, 32(2):113-133, Feb. 2002.

[19] QuARS, http://quars.isti.cnr.it/, Lami, G., QuARS: A Tool for Analyzing Requirements, Technical Report, CMU/SEI-2005-TR-014

[20] Petra Heck, Marko van Eekelen. The LaQuSo Software Product Certification Model, CS-Report 08-03, Technical University Eindhoven, 2008.