JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2005**

Vol. 4, No. 9, November-December 2005

Quality Requirements Checklist

Donald Firesmith, Software Engineering Institute, U.S.A.

Abstract

On an individual requirement by requirement basis, quality requirements are typically much more important than functional requirements because they most strongly drive the architecture of software-intensive systems. Thus, it is how well the quality requirements are engineered and implemented that tends to determine the success or failure of mission critical systems. Yet, missing or poorly specified quality requirements can all too commonly be identified during effective evaluations of the requirements specifications and the resulting architectures. This column provides a short checklist for use during the engineering and evaluation of quality requirements to help the requirements team develop better quality requirements and to help evaluators of these requirements identify defects in the associated requirements specifications.

1 THE CRITICALITY OF QUALITY REQUIREMENTS

A great deal of formal and anecdotal evidence exists that the typical quality of actual requirement specifications today is embarrassingly poor. In practice, far too many requirements are ambiguous, incomplete, infeasible, unverifiable, inadequately prioritized, and mutually inconsistent [Firesmith 2003a]. In fact, this poor quality of individual requirements and the requirements specifications that document them is a primary reason why so many projects continue to fail [Standish 1994]. Because so many requirements defects remain in requirements specifications after they have been reviewed and approved, clearly the current approaches *as applied in practice* being used to develop and review requirements are seriously inadequate to discover and correct these defects.

2 A MAJOR WAY QUALITY REQUIREMENT DEFECTS ARE CURRENTLY IDENTIFIED

Unfortunately, the poor quality of the requirements is typically not recognized during requirements engineering and the evaluation of requirements specifications. Due to inadequate customer organization (e.g., the Government or commercial market) experience, training and tool support, the stakeholder (e.g., business, user) requirements typically contain large numbers of defects. These requirements may be internally

Cite this article as follows: Donald G. Firesmith: "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

reviewed, but most defects are not found. These stakeholder requirements are then passed on to the development organization (e.g., prime contractor or internal IT), which derives system-level technical requirements. For similar reasons, these technical requirements are typically of poor quality, and a great many defects are not identified when the requirements specifications are duly evaluated during peer-level and more formal milestone reviews. This process continues down the system logical architecture from system to subsystems to subsubsystems and from prime contractors to subcontractors and integrated product teams (IPTs) who are responsible for implementing the allocated requirements. Although many defects are identified and fixed during this process of derivation and evaluation, a vast number of requirements errors still slip through the requirements engineering process into the architecture, design, and implementation.

While this passing on of requirements defects is true of all types of requirements, it is especially true of quality requirements. Unfortunately, quality requirements are the primary drivers of the system and subsystem architectures. Only when it becomes time to technically evaluate these architectures does the dismal quality of the quality requirements become obvious. Because quality requirements are the primary drivers of the architecture, an effective architecture assessment must evaluate the architecture against its support (or lack of support) for these architecturally-significant drivers. Because the requirements specifications tend to be extremely week when it comes to properly specifying quality requirements, the assessors of the architecture often find that they cannot properly evaluate the architecture because they do not have adequate requirements against which to assess it. In fact, because assessors of the architecture are so used to dealing with requirements specifications that do not contain adequate architecturally-significant quality requirements, they have been forced out of self defense to take on some of the responsibilities of the requirements team. They have been forced to develop architectural assessment methods that begin with some kind of identification and derivation of these quality requirements.

For example, it is largely because requirements teams have consistently failed to adequately specify the quality requirements that Software Engineering Institute (SEI) architects have developed the Quality Attribute Workshop (QAW) to identify and document these critical architectural drivers [SEI 2005]. It is also a major reason why the SEI's ATAM method begins with steps designed to ensure that the architecturally-significant quality factors against which to assess the architecture exist [Clements 2002]. But while such approaches are sufficient to enable an assessment of the architecture to take place, they do not replace proper requirements engineering nor are proper quality requirements needed by other stakeholders (e.g., designers, implementers, and testers) specified.

As one small step towards ensuring the early existence of adequate quality requirements, this column presents a short checklist of questions to be answered during requirements engineering and the associated evaluations of the requirements.

3 CHECKLIST

During my many years of industry experience engineering and evaluating requirements specifications, I have found the following checklist of questions useful. I recommend that they be considered during the engineering and evaluation of the quality requirements:

Requirements Identification

- Have the quality requirements been elicited from an appropriate sample of all legitimate stakeholders?
  - Were the requirements elicited from all critically-important stakeholders?
  - Were all types of stakeholders used as sources of the requirements?
  - Were the stakeholders questioned a representative sample of all stakeholders?
  - Was the sampling of stakeholders large enough to be statistically valid?
- Have relevant laws, regulations, and standards been used as sources of quality requirements?

Requirements Analysis

Unlike functional requirements where a single analysis technique (e.g., use case modeling) can be performed to identify essentially all requirements, the same analysis is not appropriate for all quality requirements. For example, such hazard analysis techniques as fault tree analysis (FTA), event tree analysis (ETA), failure modes and effects analysis (FMEA) is appropriate for safety requirements [Leveson1995], but not appropriate for performance and reliability requirements.

- Has each quality requirement been analyzed using an appropriate analysis technique (e.g., hazard analysis for safety requirements, performance modeling for performance requirements)?
- Have the appropriate analysis techniques been used to the right level of detail?
- Has a sufficient number of techniques been used? For example, fault tree analysis and event tree analysis are not redundant but rather complementary.

Types of Quality Requirements

- **Quality Model.** Has an appropriate quality model been used as a basis for identifying the types of quality requirements?
- **Standard.** Was the quality model taken from an international standard, national standard, military standard, industry standard or was it an ad hoc quality model developed specifically for the endeavor?
- **Completeness.** Was the quality model sufficiently complete to capture all relevant types of quality requirements?

- **Quality Factors or Subfactors.** Were the quality requirements only based on quality factors (e.g., performance) or were quality subfactors (e.g., jitter, response time, schedulability, and throughput) used to identify subtypes of quality requirements.

Structure of Quality Requirements

As documented in figure 1, a quality model defines the meaning of quality for a system. The quality requirements of a system are related to the quality model that is used to define the meaning of quality for that system. Specifically, each quality requirement specifies that the system under development shall achieve a minimum amount of some quality factor or subfactor defined by the quality model. This common purpose and relationship to quality models imply that no matter what quality factors they address, all quality requirements should have roughly the same structure and contents. As illustrated in figure 2, each quality requirement should consist of the following three parts:

- **Quality Criterion.** The heart of each quality requirement is a quality criterion that mandates that the system shall exhibit a single aspect of quality of the system. The actual criterion is system-specific, but it describes a general type of quality as defined in the associated quality model.
- **Condition(s).** Each quality requirement should typically also include an appropriate set of one or more conditions that specify under which circumstances the quality criterion shall hold. Usually if the conditions are not explicitly listed, the quality criterion is interpreted to apply under all conditions. However, the quality criterion may not be appropriate during system start-up, system-shutdown, or during degraded modes of operation. Perhaps the criterion is only appropriated during certain normal modes of operation such as aircraft taxiing, takeoff, climbing, cruising, decent, landing. A great many quality requirements are incomplete because they lack appropriate conditions.
- **Threshold.** All quality requirements are scalar in that they require a specific minimum amount of some quality factor. Quality requirements therefore should include a required threshold on some appropriate scale of measurement. Without the threshold, the requirement does not specify how much quality is good enough and therefore tends to be an unfeasible goal rather than a true requirement.

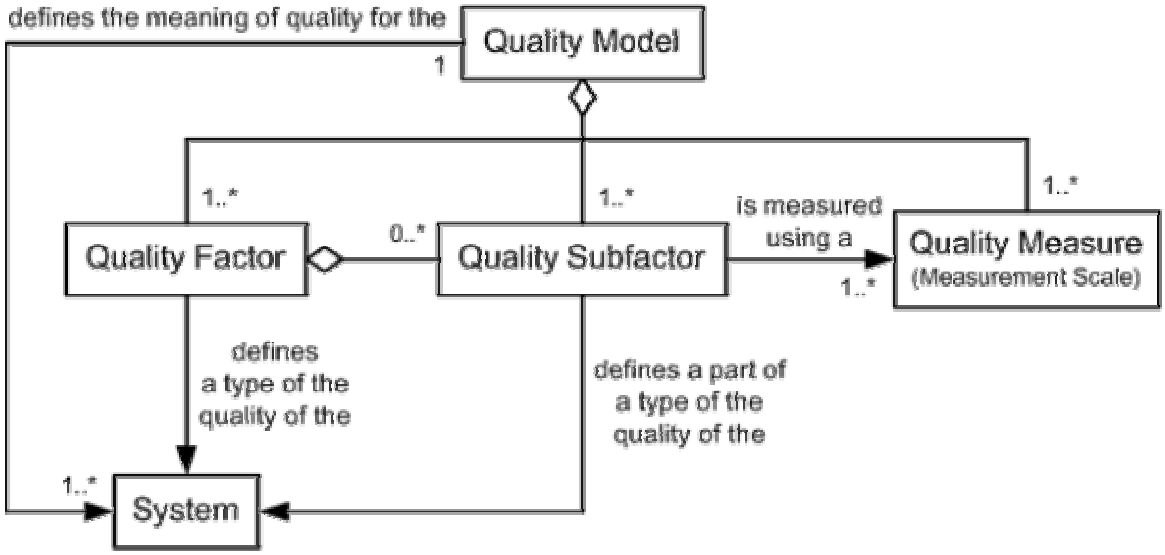


Figure 1: Quality Model

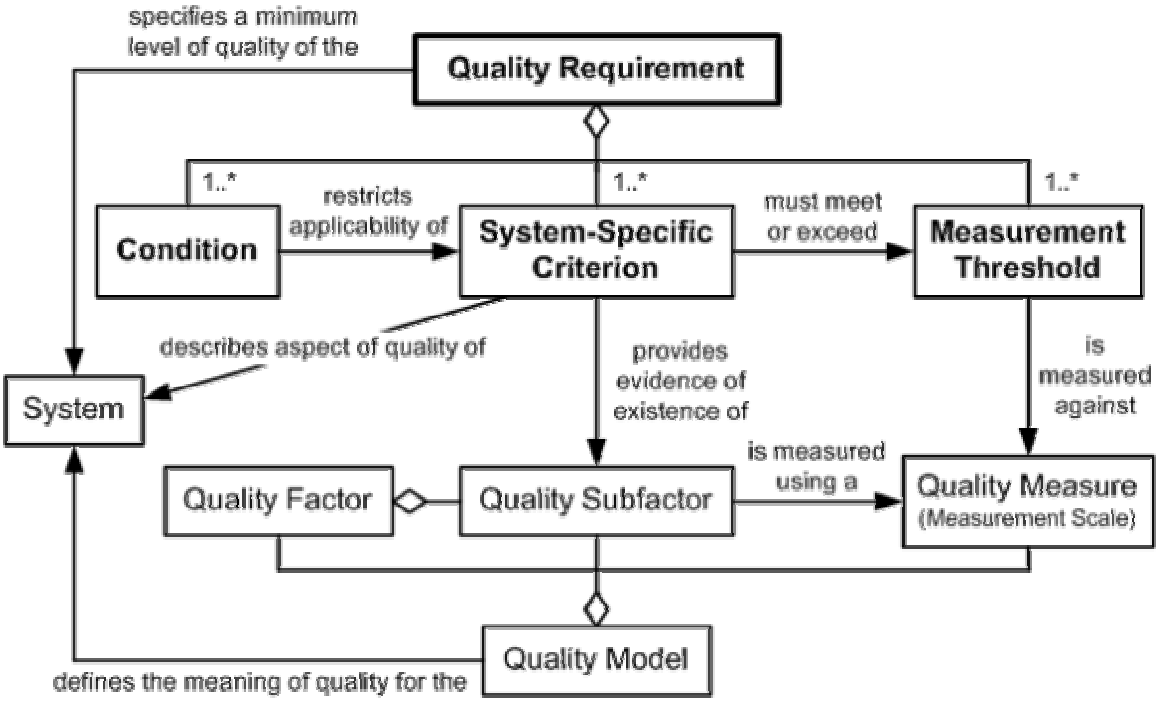


Figure2: Quality Requirements

Evaluate each quality requirement against the follow criteria to ensure it has a proper content and form:

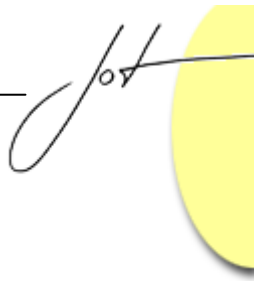
- **Quality Criterion.** Does each quality requirement include a well-defined, cohesive, system-specific quality criterion that adequately describes a single required aspect of the system in terms of an appropriate quality factor or one of its quality subfactors?

- **Goals as Requirements.** Is each quality requirement more than merely a statement that the system shall have the property of the quality factor or subfactor? For example, "The system shall be safe." Such statements are infeasible goals rather than verifiable requirements.
- **Scale with Threshold.** Is each quality requirement scalar in the sense that it has an associated mandatory threshold that specifies exactly how much of the quality factor or subfactor is necessary? Is the threshold well defined in terms of an associated scale of measurement? Is the scale of measurement appropriate for the quality criterion?
- **Conditions**. Does each quality requirement include one or more conditions that describe when the quality criterion must hold? If a quality criterion does not have any associated condition, must the quality criterion hold under all conditions (e.g., system startup, system shutdown, degraded mode)? Have all system modes and states been considered when determining the scope of applicability of the quality requirement?

Quality of Quality Requirements

Like other requirements, quality requirements should have certain characteristics:

- Is each quality requirement:
  - **Mandatory**:
    - Not just an unintended architecture, design, or implementation constraint?
    - Relevant?
  - **Feasible** given endeavor, technology, and physical limitations? Simple statements of goals such as "The system shall be secure" or "The system shall be reliable" are not good requirements because they are either infeasible because no system is totally secure or reliable or else ambiguous because they do not say how secure or reliable they need to be.
  - **Scalable** so that it is clear just how much quality is required?
  - **Unambiguous** so that all stakeholders and developers will interpret it the same way?
  - **Verifiable** via testing, demonstration, inspection, etc?
  - **Correct** in that it meets some real need of the stakeholders?
  - **Prioritized**, so that it can be allocated to an appropriate build or release?
  - **Traced** to its source?
- Does each quality requirement have an associated:
  - **Rationale**?
  - **Verification method**?



4 CONCLUSION

Because of the great importance of quality requirements as the primary drivers of the architecture of software-intensive systems, it is critical that the requirements team properly engineer them. Stakeholders of the system should not be complacent with the current state of the practice, whereby almost all requirements specifications do not contain many of the quality requirements that they should. Even those requirements that are specified are highly likely to contain significant defects. Unfortunately, many of these defects involving quality requirements are found only *after* the requirements have passed their technical evaluations. It is often during architecture evaluations that the poor quality of the quality requirements first becomes obvious when it is discovered that the architecture cannot be properly evaluated against the architecturally-significant requirements, which all too often have not been specified or else only been specified in an ambiguous, unverifiable manner. This causes the evaluations to be postponed until some of the quality requirements have been sufficiently identified to enable a determination of the architecture's support for them.

The problem does not require the development of totally new requirements engineering methods. Instead, requirements teams must begin to turn the state of the art into the state of the practice by actually using currently known methods and techniques. This paper provided a brief checklist of questions for members of the requirements evaluation team to ask the members of the requirements team. Checking the quality requirements against these evaluation criteria will go a long way towards finding existing defects and even eliminating defects from being introduced in the first place.

REFERENCES

[Clements 2002] Paul Clements, Rick Kazman, and Mark Klein, *Evaluating Software Architectures: Methods and Case Studies*, Addison Wesley, 2002.

[Firesmith 2003a] 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

[Firesmith 2003b] Donald Firesmith: "Using Quality Models to Engineer Quality Requirements", in *Journal of Object Technology*, vol. 2, no. 5, September-October 2003, pp. 67-75. http://www.jot.fm/issues/issue_2003_09/column6

[Leveson 1995] Nancy Leveson, *Safeware: System Safety and Computers*, Addison Wesley, 1995.

[SEI 2005] Software Engineering Instititue, Quality Attribute Workshop, 2005. http://www.sei.cmu.edu/architecture/products_services/qaw.html

[Standish 1994] The Standish Group International, Inc. *The CHAOS Report*, 1994. http://www1.standishgroup.com/sample_research/chaos_1994_1.php

ACKNOWLEDGEMENTS

Many thanks go to my colleague Mary Ann Lapham, who reviewed this column and provided helpful observations and recommendations.

Disclaimers

The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

The views and conclusions contained in this column are solely those of the author and should not be interpreted as representing official policies, either expressed or implied, of the Software Engineering Institute, Carnegie Mellon University, the U.S. Air Force, the U.S. Department of Defense, or the U.S. Government.

About the author

Donald Firesmith is a senior member of the technical staff at the Software Engineering Institute (SEI), where he helps the US Government acquire large, complex, software-intensive systems. Working in industrial software development since 1979, he has worked primarily with object technology since 1984 and has written 5 books on the subject. During the last four years, he has developed the world's largest (1,100+ webpage), free, and open source informational website of reusable process engineering components. Based on the OPEN Process Framework (OPF), it is located at http://www.opfro.org. Currently writing a book on the engineering of safety requirements, he can be reached at dgf@sei.cmu.edu.