Quality Definitions of Qualities

Spencer Smith, Jacques Carette, Olu Owojaiye,
Peter Michalski and Ao Dong

September 26, 2019

Abstract

...

Contents

1 Introduction

Purpose and scope of the document. [Needs to be filled in. Should reference the overal research proposal, and the "state of the practice" exercise in particular. —SS]

The presentation is divided into two main sections: i) qualities that apply to software products, software artifacts and software development processes, and ii) qualities that are considered important for good specifications. The specification could be a specification of requirements, design or a test plan.

2 Qualities of Software Products, Artifacts and Processes

To assess the current state of software development, and to understand how future changes impact software development, we need a clear definition of what we mean by quality. The concept of quality is decomposed into a set of separate qualities. This set of qualities can be applied to the software product, the software artifacts (documentation, test cases, etc) and to the software development process itself.

Our analysis is centred around a set of software qualities. Quality is not considered as a single measure, but a collection of different qualities, often called "ilities." These qualities highlight the desirable nonfunctional properties for software artifacts, which include both documentation and code. Some qualities, such as visibility and productivity, apply to the process used for developing the software. The following list of qualities is based on Ghezzi et al. (2003). To the list from Ghezzi et al. (2003), we have added three qualities important for SC: installability, reproducibility and sustainability.

2.1 Installability

A measure of the ease of installation.

2.2 Correctness

Software is correct if it matches its specification.

2.3 Verifiability

Verifiability involves "solving the equations right" (Roache, 1998, p. 23); it benefits from rational documentation that systematically shows, with explicit traceability, how the governing equations are transformed into code.

2.4 Validatability

Validatability means "solving the right equations" (Roache, 1998, p. 23). Validatability is improved by a rational process via clear documentation of the theory and assumptions, along with an explicit statement of the systematic steps required for experimental validation.

2.5 Reliability

Reliability is a critical quality for scientific software, since the results of computations are meaningless, if they are not dependable. Reliability is closely tied to verifiability, since the key quality to verify is reliability, while the act of verification itself improves reliability.

2.6 Robustness

2.7 Performance

Performance considerations can make certification challenging, since QA becomes more difficult for more complex code. However, as Roache (Roache, 1998, p. 355) points out, using simpler algorithms and reducing the number of options in general purpose code, is not always a practical option.

2.8 Usability

Usability can be a problem. Different users, solving the same physical problem, using the same software, can come up with different answers, due to differences in parameter selection (Roache, 1998, p. 370). To reduce misuse, a rational process must state expected user characteristics, modelling assumptions, definitions and the range of applicability of the code.

2.9 Maintainability

Maintainability is necessary in scientific software, since change, through iteration, experimentation and exploration, is inevitable. Models of physical phenomena and numerical techniques necessarily evolve over time Carver et al. (2007); Segal and Morris (2008). Proper documentation, designed with change in mind, can greatly assist with change management. QA activities need to take the need for creativity into account, while not smothering it (Roache, 1998, p. 352).

2.10 Reusability

Reusability provides support for the quality of reliability, since reliability is improved by reusing trusted components Dubois (2005). (Care must still be taken with reusing trusted components, since blind reuse in a new context can lead to errors, as dramatically shown in the Ariane 5 disaster (Oliveira and Stewart, 2006, p. 37–38).) The odds of reuse are improved when it is considered right from the start.

2.11 Portability

2.12 Understandability

Understandability is necessary, since reviewers can only certify something they understand. Scientific software developers have the view "that the science a developer embeds in the code must be apparent to another scientist, even ten years later" Kelly (2013). Understandability applies to the documentation and code, while usability refers to the executable software. Documentation that follows a rational process is the easiest to follow.

2.13 Interoperability

2.14 Visibility/Transparency

2.15 Reproducibility

Reproducibility is a required component of the scientific method Davison (2012). Although QA has, "a bad name among creative scientists and engineers" (Roache, 1998, p. 352), the community need to recognize that partic-

ipating in QA management also improves reproducibility. Reproducibility, like QA, benefits from a consistent and repeatable computing environment, version control and separating code from configuration/parameters Davison (2012).

2.16 Productivity

2.17 Sustainability

3 Desirable Qualities of Good Specifications

To achieve the qualities listed in Section 2, the documentation should achieve the qualities listed in this section. All but the final quality listed (abstraction), are adapted from the IEEE recommended practise for producing good software requirements IEEE (1998). Abstraction means only revealing relevant details, which in a requirements document means stating what is to be achieved, but remaining silent on how it is to be achieved. Abstraction is an important software development principle for dealing with complexity (Ghezzi et al., 2003, p. 40). Smith and Koothoor (2016) present further details on the qualities of documentation for SCS.

3.1 Completeness

Documentation is said to be complete when all the requirements of the software are detailed. That is, each goal, functionality, attribute, design constraint, value, data, model, symbol, term (with its unit of measurement if applicable), abbreviation, acronym, assumption and performance requirement of the software is defined. The software's response to all classes of inputs, both valid and invalid and for both desired and undesired events, also needs to be specified.

3.2 Consistency

Documentation is said to be consistent when no subset of individual statements are in conflict with each other. That is, a specification of an item made at one place in the document should not contradict the specification of the same item at another location.

3.3 Modifiability

The documentation should be developed in such a way that it is easily modifiable so that likely future changes do not destroy the structure of the document. Also it should be easy to reflect the change, wherever needed in the document to maintain consistency, traceability and completeness. For documentation to be modifiable, its format must be structured in a way that repetition is avoided and cross-referencing is employed.

3.4 Traceability

Documentation should be traceable, as this facilitates maintenance and review. If a change is made to the design or code of the software, then all the documentation relating to those segments have to be modified. This property is also important for recertification.

3.5 Unambiguity

Documentation is said to be unambiguous only when every requirement's specification has a unique interpretation. The documentation should be unambiguous to all audiences, including developers, users and reviewers.

3.6 Correctness

There is no direct tool or method for measuring correctness. One way of building confidence in correctness is by reviewing to ensure that each requirement stated is one that the stakeholders and experts desire. By maintaining traceability, consistency and unambiguity, we can reduce the occurrence of errors and make the goal of reviewing for correctness easier.

3.7 Verifiability

Every requirement in the documentation must be the one fulfilled by the implemented software. Therefore all the requirements should be clear, unambiguous and testable, so that a person or a machine can verify whether the software product meets the requirements.

3.8 Abstract

Documented requirements are said to be abstract if they state what the software must do and the properties it must possess, but do not speak about how these are to be achieved. For example, a requirement can specify that an Ordinary Differential Equation (ODE) must be solved, but it should not mention that Euler's method should be used to solve the ODE. How to accomplish the requirement is a design decision, which is documented during the design phase.

References

Jeffrey C. Carver, Richard P. Kendall, Susan E. Squires, and Douglass E. Post. Software development environments for scientific and engineering software: A series of case studies. In *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, pages 550–559, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2828-7. doi: http://dx.doi.org/10.1109/ICSE.2007.77.

A. P. Davison. Automated capture of experiment context for easier reproducibility in computational research. *Computing in Science & Engineering*, 14(4):48–56, July-Aug 2012.

P. F. Dubois. Maintaining correctness in scientific programs. *Computing in Science & Engineering*, 7(3):80–85, May-June 2005. doi: 10.1109/MCSE.2005.54.

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

IEEE. Recommended practice for software requirements specifications. *IEEE Std 830-1998*, pages 1–40, October 1998. doi: 10.1109/IEEESTD.1998.88286.

Diane Kelly. Industrial scientific software: A set of interviews on software development. In *Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '13, pages 299–310, Riverton, NJ, USA, 2013. IBM Corp. URL http://dl.acm.org/citation.cfm?id=2555523.2555555.

Suely Oliveira and David E. Stewart. *Writing Scientific Software: A Guide to Good Style*. Cambridge University Press, New York, NY, USA, 2006. ISBN 0521858968.

Patrick J. Roache. *Verification and Validation in Computational Science and Engineering*. Hermosa Publishers, Albuquerque, New Mexico, 1998.

Judith Segal and Chris Morris. Developing scientific software. *IEEE Software*, 25(4):18–20, July/August 2008.

W. Spencer Smith and Nirmitha Koothoor. A document-driven method for certifying scientific computing software for use in nuclear safety analysis. *Nuclear Engineering and Technology*, 48(2):404–418, April 2016. ISSN 1738-5733. doi: http://dx.doi.org/10.1016/j.net.2015.11.008. URL http://www.sciencedirect.com/science/article/pii/S1738573315002582.