Quality Definitions of Qualities

Spencer Smith
McMaster University, Canada
smiths@mcmaster.ca

Jacques Carette
McMaster University, Canada
carette@mcmaster.ca

Olu Owojaiye
McMaster University, Canada
owojaiyo@mcmaster.ca

Peter Michalski
McMaster University, Canada
michap@mcmaster.ca

Ao Dong
McMaster University, Canada

Abstract

...

2012 ACM Subject Classification Author: Please fill in 1 or more \ccsdesc macro

Keywords and phrases Author: Please fill in \keywords macro

Contents

1 Introduction

Purpose and scope of the document. [Needs to be filled in. Should reference the overall 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.

The pattern in the presentation for each of the qualities is the same. First we summarize all of the definitions that we could find in the literature. To keep the clutter of quotation marks down, we have adopted the convention that each definition is given verbatim from the cited source, but without showing quotation marks. After the summary of the existing definitions, we propose the definition that we would like to work with going forward. This definition can either be our preference from the existing definitions, or a new definition, which is often found by combining existing definitions. Following the proposed definition is an explanation for the reasoning that led to this choice.

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 components that together make up "quality". Unfortunately, these are called *qualities*. These are associated to the software product, the software artifacts (documentation, test cases, etc) and to the software development process itself, and combinations thereof.

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 [owner —OO]

▶ **Definition 1.** A measure of the ease of installation. [Where does this definition come from? Please add a citation. Is it from McCall et al. [1977]? I thought so originally, but after looking into it, I don't think so anymore. —SS]

▶ **Definition 2.** Installability is related to the amount of human effort required to install a software in a designated environment [Berander et al., 2005].

▶ **Definition 3.** "The capability of a software product to be installed in a specified environment" [Berander et al., 2005].

▶ **Definition 4.** "The efficacy of the installation, uninstallation or reinstallation process of a software product in a specified environment" [ISO-IEC, 2010]

Proposed Definition

Definition 2. [This definition does not allow for measuring the machine time. —SS]

Reasoning

[The rationale is missing. Please add. —SS]

2.2 Correctness [owner —OO]

▶ **Definition 5.** The degree to which a software's specification is satisfied [Berander et al., 2005].

▶ **Definition 6.** A software is correct if it functions according to its specified functional and non-functional specification [Ghezzi et al., 2003]. [This is not how Ghezzi et al. [2003] defines correctness. They only require satisfaction of the functional requirements. —SS]

▶ **Definition 7.** According to Wilson [2009], "the degree to which a system is free from defects in its specification, design, and implementation, is its correctness."

▶ **Definition 8.** "The ability of software products to perform their exact tasks, as defined by their specification" [Meyer, 1988].

▶ **Definition 9.** "The extent to which a program satisfies its specifications and fulfills the user's mission objectives" [McCall et al., 1977]. (As summarized in van Vliet [2000].)

Proposed Definition

Definition 6

Reasoning

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.

The quality of a software's operation is dependent on the degree of correctness [Berander et al., 2005]. Correctness and reliability are said to have dependencies, such that if a system exhibits a high degree of correctness then it tends to be reliable [Ghezzi et al., 2003].

2.3 Verifiability [owner —OO]

▶ **Definition 10.** A software is verifiable if its properties can be verified [Ghezzi et al., 2003].

▶ **Definition 11.** Attributes of software that relate to the effort needed for validating the modified software [Berander et al., 2005].

▶ **Definition 12.** The degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met [610-1990, 1991b].

▶ **Definition 13.** The degree to which a requirement is stated in terms that permit establishment of test criteria and performance [610-1990, 1991b].

▶ **Definition 14.** Ease of performing testing on a software product or system [610-1990, 1991b].

Proposed Definition

Combo of definitions 13 and 14 ?

Reasoning

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. [Verifiability is sometimes referred to as testability, so I culled some testability definitions here —OO] [culled means removed - is that your intention? —SS]

[In case we need it, here is a definition of testability. —SS]

▶ **Definition 15.** "The effort required to test a program to ensure that it performs its intended function" [McCall et al., 1977]. (As summarized in van Vliet [2000].)

2.4 Validatability [owner —OO]

▶ **Definition 16.** Validatability of a software is the degree of ease in validating (checking) that software meets user needs.

▶ **Definition 17.** Validatability means "solving the right equations" [Roache, 1998, p. 23]. [question about this definition —OO]

▶ **Definition 18.** The degree of ease of evaluating software products or system to determine whether it satisfies specified business requirements. http://softwaretestingfundamentals.com/verification-vs-validation/ [Here I tweaked the meaning of validation to suit validatability —OO] [Only a few resource on validatability —OO]

Proposed Definition

Definition 18.

Reasoning

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 [owner —OO]

▶ **Definition 19.** The probability that the software will operate as expected over a specified time interval [Ghezzi et al., 2003].

▶ **Definition 20.** A set of attributes that relate to the capability of software to maintain its level of performance under stated conditions for a stated period of time [Berander et al., 2005].

▶ **Definition 21.** "The capability of the software product to maintain a specified level of performance when used under specified conditions" [for Standardization/International Electrotechnical Commission et al., 2001].

▶ **Definition 22.** Code possesses the characteristic reliability to the extent that it can be expected to perform its intended functions satisfactorily [Boehm, 2007].

▶ **Definition 23.** A concern encompassing correctness and robustness [Meyer, 1988].

▶ **Definition 24.** "The extent to which a program can be expected to perform its intended function with required precision" [McCall et al., 1977]. (As summarized in van Vliet [2000].)

Proposed Definition

[Needs to be completed. —SS]

Reasoning

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.

Reliability models can be used to predict reliability of a software product. For example measuring Mean Time to Fail (MTTF) can be a good measure of reliability [Berander et al., 2005].

2.6 Robustness [owner —PM]

▶ **Definition 25.** The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions [610-1990, 1991b].

▶ **Definition 26.** The quality can be further informally refined as the ability of a software to keep an acceptable behaviour, expressed in terms of robustness requirements, in spite of exceptional or unforeseen execution conditions (such as the unavailability of system resources, communication failures, invalid or stressful inputs, etc.) [Fernandez et al., 2005].

▶ **Definition 27.** Code possesses the characteristic of robustness to the extent that it can continue to perform despite some violation of the assumptions in its specification [Boehm, 2007].

▶ **Definition 28.** A program is robust if it behaves "reasonably", even in circumstances that were not anticipated in the requirements specification - for example, when it encounters incorrect input data or some hardware malfunction [Ghezzi et al., 1991].

Proposed Definition

Definition 28 rephrased: Software possesses the characteristic of robustness if it behaves "reasonably" in circumstances that were not anticipated in the requirements specification or are in violation of it.

Reasoning

This definition indicates that robustness is related to the quality of correctness (both within and outside of it).

[Still want to refine what we mean by reasonable —PM]

2.7 Performance [owner —PM]

▶ **Definition 29.** The degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage [610-1990, 1991b].

▶ **Definition 30.** How well or how rapidly the system must perform specific functions. Performance requirements encompass speed (database response times, for instance), throughput (transactions per second), capacity (con-current usage loads), and timing (hard real-time demands) [Wiegers, 2003].

▶ **Definition 31.** In software engineering we often equate performance with efficiency. A software system is efficient if it uses computing resources economically [Ghezzi et al., 1991].

Proposed Definition

Combined definition 29 and 30: The degree to which a system or component accomplishes its designated functions within given constraints, such as speed (database response times, for instance), throughput (transactions per second), capacity (con-current usage loads), and timing (hard real-time demands).

Reasoning

This definition includes all important categories of performance.

[I think this is done. —PM]

2.8 Usability [owner —JC]

▶ **Definition 32.** The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use.

ISO defines usability as

Nielsen and (separately) Schneidermann have defined usability as part of usefulness and is composed of:

- Learnability: How easy is it for users to accomplish basic tasks the first time they encounter the design?
- Efficiency: Once users have learned the design, how quickly can they perform tasks?
- Memorability: When users return to the design after a period of not using it, how easily can they re-establish proficiency?
- Errors: How many errors do users make, how severe are these errors, and how easily can they recover from the errors?
- Satisfaction: How pleasant is it to use the design?

In that context, it makes sense to separate *usefulness* into *usability* (purely an interface concern) and *utility* (in the economics sense of the word).

There are two ISO standards covering this, namely ISO/TR 16982:202 and ISO 9241.

The Interaction Design Foundation `https://www.interaction-design.org/literature/topics/usability` further lists the following desirable outcomes:

1. It should be easy for the user to become familiar with and competent in using the user interface during the first contact with the website. For example, if a travel agent's website is a well-designed one, the user should be able to move through the sequence of actions to book a ticket quickly.
2. It should be easy for users to achieve their objective through using the website. If a user has the goal of booking a flight, a good design will guide him/her through the easiest process to purchase that ticket.
3. It should be easy to recall the user interface and how to use it on subsequent visits. So, a good design on the travel agent's site means the user should learn from the first time and book a second ticket just as easily.

One core reference, for definitions and metrics, is Bevan [1995].

▶ **Definition 33.** "The effort required to learn, operate, prepare input, and interpret output of a program" [McCall et al., 1977]. (As summarized in van Vliet [2000].)

Proposed Definition

[Still needs to be completed —SS]

Reasoning

[Needs to be completed —SS]

2.9 Maintainability [owner —PM]

▶ **Definition 34.** The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment [610-1990, 1991b].

▶ **Definition 35.** ISO/IEC 25010 refers to maintainability as the degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers [25010:2011, 2011].

▶ **Definition 36.** Effort required to locate and fix an error in a program [Pressman, 2005].

▶ **Definition 37.** A set of attributes that bear on the effort needed to make specified modifications (which may include corrections, improvements, or adaptations of software to environmental changes and changes in the requirements and functional specifications)[Pfleeger, 2006].

▶ **Definition 38.** We will view maintainability as two separate qualities: repairability and evolvability. Software is repairable if it allows the fixing of defects; it is evolvable if it allows changes that enable it to satisfy new requirements [Ghezzi et al., 1991].

▶ **Definition 39.** Code possesses the characteristic of maintainability to the extent that it facilitates updating to satisfy new requirements or to correct deficiencies [Boehm, 2007].

▶ **Definition 40.** "The effort required to locate and fix an error in an operational program" [McCall et al., 1977]. (As summarized in van Vliet [2000].) [It looks like Pressman [2005] was using McCall et al. [1977] for their definition. —SS]

Proposed Definition

Combined Definition 34 and Definition 39: The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or satisfy new requirements.

Reasoning

This definition includes all potential reasons to modify the software.

[I think this is done. —PM]

2.10 Reusability [owner —PM]

▶ **Definition 41.** The degree to which a software module or other work product can be used in more than one software system [610-1990, 1991b].

▶ **Definition 42.** Extent to which a program [or parts of a program] can be reused in other applications - related to the packaging and scope of the functions that the program performs [Pressman, 2005].

▶ **Definition 43.** The extent to which a software component can be used with or without adaptation in a problem solution other than the one for which it was originally developed [Kalagiakos, 2003].

▶ **Definition 44.** Reusability is the likelihood a segment of source code that can be used again to add new functionalities with slight or no modification [Sandhu et al., 2010].

▶ **Definition 45.** "The extent to which a program (or parts thereof) can be reused in other applications" [McCall et al., 1977]. (As summarized in van Vliet [2000].)

Proposed Definition

Definition 43: The extent to which a software component can be used with or without adaptation in a problem solution other than the one for which it was originally developed.

Reasoning

This definition highlights the possible but not necessary adaptation of the software component(s) being transferred.

[I think this is done. —PM]

2.11 Portability [owner —PM]

▶ **Definition 46.** The ease with which a system or component can be transferred from one hardware or software environment to another [610-1990, 1991b].

▶ **Definition 47.** An application is portable across a class of environments to the degree that the effort required to transport and adapt it to a new environment in the class is less than the effort of redevelopment [Mooney, 1990].

▶ **Definition 48.** Effort required to transfer the program from one hardware and/or software system environment to another [Pressman, 2005].

▶ **Definition 49.** A set of attributes that bear on the ability of software to be transferred from one environment to another (including the organizational, hardware, of software environment)[Pfleeger, 2006].

▶ **Definition 50.** Code possesses the characteristic of portability to the extent that it can be operated easily and well on computer configurations other than its current one. This implies that special function features, not easily available at other facilities, are not used, that standard library functions and subroutines are selected for universal applicability, and so on [Boehm, 2007].

▶ **Definition 51.** Portability refers to the ability to run a system on different hardware platforms [Ghezzi et al., 1991].

▶ **Definition 52.** "The effort required to transfer a program from one hardware and/or software environment to another" [McCall et al., 1977]. (As summarized in van Vliet [2000].)

Proposed Definition

Definition 48 rephrased: Effort required to transfer a program between system environments.

Reasoning

This is measurable and succinct.

[I think this is done. —PM]

2.12 Understandability [owner —JC]

Understandability is artifact-dependent. What it means for a user-interface (graphical or otherwise) to be understandable is wildly different than what it means for the code, and even the user documentation.

The literature here is thin and scattered. More work will need to be done to find something useful.

Interestingly, the business literature seems to have taken more care to define this. Here we encounter

Understandability is the concept that X should be presented so that a reader can easily comprehend it.

At least this brings in the idea that the *reader* is actively involved, and indirectly that the reader's knowledge may be relevant, as well as the "clarity of exposition" of X.

Section 11.2 of Adams et al. [2015] does have a full definition.

Proposed Definition

[Still needs to be completed —SS]

Reasoning

[Needs to be completed —SS]

2.13 Interoperability [owner —AD]

▶ **Definition 53.** "The effort required to couple one system with another" [McCall et al., 1977]. (As summarized in van Vliet [2000].)

▶ **Definition 54.** Interoperability is the ability of two or more systems or components to exchange information and to use the information that has been exchanged [610-1990, 1991a].

▶ **Definition 55.** The degree to which two or more systems, products or components can exchange information and use the information that has been exchanged [25010:2011, 2011].

▶ **Definition 56.** The capability to communicate, execute programs, and transfer data among various functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units [24765:2010, 2010].

▶ **Definition 57.** Interoperability is a characteristic of a product or system, whose interfaces are completely understood, to work with other products or systems, present or future, in either implementation or access, without any restrictions [AFUL, 2019].

▶ **Definition 58.** Interoperability is the ability of different information systems, devices and applications ('systems') to access, exchange, integrate and cooperatively use data in a coordinated manner, within and across organizational, regional and national boundaries, to provide timely and seamless portability of information and optimize the health of individuals and populations globally. Health data exchange architectures, application interfaces and standards enable data to be accessed and shared appropriately and securely across the complete spectrum of care, within all applicable settings and with relevant stakeholders, including by the individual [HIMSS, 2019].

Four Levels of Interoperability:

- Foundational (Level 1) – establishes the inter-connectivity requirements needed for one system or application to securely communicate data to and receive data from another
- Structural (Level 2) – defines the format, syntax, and organization of data exchange including at the data field level for interpretation
- Semantic (Level 3) – provides for common underlying models and codification of the data including the use of data elements with standardized definitions from publicly available value sets and coding vocabularies, providing shared understanding and meaning to the user
- Organizational (Level 4) – includes governance, policy, social, legal and organizational considerations to facilitate the secure, seamless and timely communication and use of data both within and between organizations, entities and individuals. These components enable shared consent, trust and integrated end-user processes and workflows

Proposed Definition

Definition 58 rephrased: Interoperability is the ability of different information systems, devices and applications ('systems') to access, exchange, integrate and cooperatively use data in a coordinated manner, within and across organizational, regional and national boundaries, to provide timely and seamless portability of information.

[I moved the proposed definition to the end, to match the other sections. Hopefully I did not change the meaning. —SS]

Reasoning

[Needs to be completed —SS]

2.14 Visibility/Transparency [owner —AD]

▶ **Definition 59.** A software development process is visible if all of its steps and its current status are documented clearly. Another term used to characterize this property is transparency [Ghezzi et al., 1991].

Proposed Definition

Definition 59.

Reasoning

[Needs to be completed —SS]

2.15 Reproducibility [owner —SS]

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 participating 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].

Reproducibility is defined as:

▶ **Definition 60.** A result is said to be reproducible if another researcher can take the original code and input data, execute it, and re-obtain the same result (Peng, Dominici, and Zeger, 2006), as cited in Benureau and Rougier [2017].

The related concept of replicable is defined as:

▶ **Definition 61.** Documentation achieves replicability if the description it provides of the algorithms is sufficiently precise and complete for an independent researcher to re-obtain the results it presents. [Benureau and Rougier, 2017]

It would be worthwhile to look for some additional definitions.

Proposed Definition

[Needs to be completed —SS]

Reasoning

[Needs to be completed —SS]

2.16 Productivity [owner —AD]

▶ **Definition 62.** The best definition of the productivity of a process is

$$\text{Productivity} = \frac{\text{Outputs produced by the process}}{\text{Inputs consumed by the process}}$$

Defining inputs. For the software process, providing a meaningful definition of inputs is a nontrivial but generally workable problem. Inputs to the software process generally comprise labor, computers, supplies, and other support facilities and equipment. Defining outputs. The big problem in defining software productivity is defining outputs. Here we find a defining delivered source instructions (DSI) or lines of code as the output of the software process is totally inadequate, and they argue that there are a number of deficiencies in using DSI. However, most organizations doing practical productivity measurement still use DSI as their primary metric [Boehm, 1987].

▶ **Definition 63.** Productivity is the amount of output (what is produced) per unit of input used.If we can measure the size of the software product and the effort required to develop the product, we have:

$$\text{productivity} = \text{size}/\text{effort} \tag{1}$$

Equation (1) assumes that size is the output of the software production process and effort is the input to the process. This can be contrasted with the viewpoint of software cost models where we use size as an independent variable (i.e., an input) to predict effort which is treated as an output. Equation (1) is simple to operationalize if we have a single dominant size measure, for example, product size measured in lines of code [Kitchenham and Mendes, 2004].

Proposed Definition

The first sentence of Definition 63: Productivity is the amount of output (what is produced) per unit of input used.

Reasoning

[Needs to be completed —SS]

2.17 Sustainability [owner —SS]

One of the original definitions of sustainability (for systems, not software specific), and still often quoted, is:

▶ **Definition 64.** The ability to meet the needs of the present without compromising the ability of future generations to meet their own needs [Brundtland, 1987].

This is the definition used by International Institute for Sustainable Development [2019].

To make it more useful, this definition is often split into three dimensions: social, economic and environmental. [cite UN paper [9] in Penzenstadler and Femmer [2013] —SS] To this list Penzenstadler and Henning (2013) have added technical sustainability [Penzenstadler and Femmer, 2013]. Where technical sustainability for software is defined as:

▶ **Definition 65.** Technical sustainability has the central objective of long-time usage of systems and their adequate evolution with changing surrounding conditions and respective requirements [Penzenstadler and Femmer, 2013].

The fourth dimension of technical sustainability is also added by [Wolfram et al., 2017]. Technical sustainability is the focus on the thesis by Hygerth [2016].

▶ **Definition 66.** Sustainable development is a mindset (principles) and an accompanying set of practices that enable a team to achieve and maintain an optimal development pace indefinitely [Tate, 2005].

Parnas discusses as software aging [Parnas, 1994].

SCS specific definitions:

▶ **Definition 67.** The concept of sustainability is based on three pillars: the ecological, the economical and the social. This means that for a software to be sustainable, we must take all of its effects – direct and indirect – on the environment, the economy and the society into account. In addition, the entire life cycle of a software has to be considered: from planning and conception to programming, distribution, installation, usage and disposal [Heine, 2017].

▶ **Definition 68.** The capacity of the software to endure. In other words, sustainability means that the software will continue to be available in the future, on new platforms, meeting new needs [Katz, 2016].

Definition from Neil Chue Hong:

▶ **Definition 69.** Sustainable software is software which is: – Easy to evolve and maintain – Fulfils its intent over time – Survives uncertainty – Supports relevant concerns (Political, Economic, Social, Technical, Legal, Environmental) [Katz, 2016].

Paper critical of a lack of a definition [Venters et al., 2014].

Sounds like definition of maintainability.

Find paper that combines nonfunctional qualities into sustainability.

Sustainability depends on the software artifacts AND the software team AND the development process.

Proposed Definition

[Needs to be completed —SS]

Reasoning

[Needs to be completed —SS]

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]. Correctness was in the above list, so it is not repeated here. Smith and Koothoor [2016] present further details on the qualities of documentation for SCS.

3.1 Completeness [owner —AD]

▶ **Definition 70.** A specification is complete to the extent that all of its parts are present and each part is fully developed. A software specification must exhibit several properties to assure its completeness [Boehm, 1984]:

- No TBDs. TBDs are places in the specification where decisions have been postponed by writing "To be Determined" or "TBD."
- No nonexistent references. These are references in the specification to functions, inputs, or outputs (including databases) not defined in the specification.
- No missing specification items. These are items that should be present as part of the standard format of the specification, but are not present.

- No missing functions. These are functions that should be part of the software product but are not called for in the specification.
- No missing products. These are products that should be part of the delivered software but are not called for in the specification.

Proposed Definition

The first sentence of Definition 70: A specification is complete to the extent that all of its parts are present and each part is fully developed.

Reasoning

[Needs to be completed —SS]

3.2 Consistency [owner —AD]

▶ **Definition 71.** A specification is consistent to the extent that its provisions do not conflict with each other or with governing specifications and objectives. Specifications require consistency in several ways [Boehm, 1984].

- Internal consistency. Items within the specification do not conflict with each other.
- External consistency. Items in the specification do not conflict with external specifications or entities.
- Traceability. Items in the specification have clear antecedents in earlier specifications or statements of system objectives.

▶ **Definition 72.** Consistency requires that no two or more requirements in a specification contradict each other. It is also often regarded as the case where words and terms have the same meaning throughout the requirements specifications (consistent use of terminology). These two views of consistency imply that mutually exclusive statements and clashes in terminology should be avoided [Zowghi and Gervasi, 2003].

▶ **Definition 73.** Consistency: 1. the degree of uniformity, standardization, and freedom from contradiction among the documents or parts of a system or component 2. software attributes that provide uniform design and implementation techniques and notations [24765:2010, 2010].

Proposed Definition

Definition 73.

Reasoning

[Needs to be completed —SS]

3.3 Modifiability [owner —JC]

Here we do seem to have a simple, if somewhat uninformative, definition:

▶ **Definition 74.** Modifiability is the degree of ease at which changes can be made to a system, and the flexibility with which the system adapts to such changes.

IEEE Standard 610 seems to speak about this. (which is superseded?)

Reasoning

[Needs to be completed —SS]

3.4 Traceability [owner —JC]

Here the Wikipedia page https://en.wikipedia.org/wiki/Traceability is actually rather informative, especially as it also lists how this concept is used in other domains. A generic definition that is still quite useful is

▶ **Definition 75.** The capability (and implementation) of keeping track of a given set or type of information to a given degree, or the ability to chronologically interrelate uniquely identifiable entities in a way that is verifiable.

By specializing the above to software artifacts, "interrelate" to "why is this here" (for forward tracing from requirements), this does indeed give what is meant in SE.

Various standards (DO178C, ISO 26262, and IEC61508) explicitly mention it.

24765-2017 - ISO/IEC/IEEE International Standard - Systems and software engineering–Vocabulary has a full definition, namely

1. the degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another;
2. the identification and documentation of derivation paths (upward) and allocation or flowdown paths (downward) of work products in the work product hierarchy;
3. the degree to which each element in a software development product establishes its reason for existing; and discernible association among two or more logical entities, such as requirements, system elements, verifications, or tasks.

Proposed Definition

[Needs to be completed. —SS]

Reasoning

[Needs to be completed —SS]

3.5 Unambiguity [owner —SS]

A specification is unambiguous when it has a unique interpretation. If there is a possibility that two readers will have two different interpretations, than the specification is ambiguous. [When I get the Ghezzi text back from Olu, I'll check to see if they have anything to add to this definition. —SS]

A Software Requirements Specification (SRS) is unambiguous if, and only if, every requirement stated therein has only one interpretation [IEEE, 1998].

Proposed Definition

[Needs to be completed. —SS]

Reasoning

[Needs to be completed —SS]

3.6 Verifiability [owner —SS]

- Verification - Are we building the product right? Are we implementing the requirements correctly (internal)
- Validation - Are we building the right product? Are we getting the right requirements (external)
- According to Capability Maturity Model (CMM)

- Software Verification: The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. [IEEE-STD-610]
- Software Validation: The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements. [IEEE-STD-610]

"An SRS is verifiable if, and only if, every requirement stated therein is verifiable. A requirement is verifiable if, and only if, there exists some finite cost-effective process with which a person or machine can check that the software product meets the requirement. In general any ambiguous requirement is not verifiable." [IEEE, 1998]

Verifiability is related to testability, which is defined by McCall et al. as "The effort required to test a program to ensure that it performs its intended function" [van Vliet, 2000].

[When I get the Ghezzi text back from Olu, I'll check to see if they have anything to add to this definition. —SS]

Proposed Definition

[Needs to be completed —SS]

Reasoning

[Needs to be completed —SS]

3.7 Abstract [owner —SS]

▶ **Definition 76.** 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 [Ghezzi et al., 2003].

▶ **Definition 77.** "An abstraction for a software artifact is a succinct description that suppresses the details that are unimportant to a software developer and emphasizes the information that is important." [Krueger, 1992]

▶ **Definition 78.** "Abstraction means that we concentrate on the essential features and ignore, abstract from, details that are not relevant at the level we are currently working." [van Vliet, 2000, p. 296]

▶ **Definition 79.** "Abstraction in mathematics is the process of extracting the underlying essence of a mathematical concept, removing any dependence on real world objects with which it might originally have been connected, and generalizing it so that it has wider applications or matching among other abstract descriptions of equivalent phenomena." Wikipedia Definition

Abstraction is related to reusability (and other qualities).

[When I get the Ghezzi text back from Olu, I'll check to see if they have anything to add to this definition. —SS]

Proposed Definition

[Needs to be completed —SS]

Reasoning

[Needs to be completed —SS]

References

ISO/IEC/IEEE Standards Committee 24765:2010. Systems and software engineering - vocabulary. Standard, International Organization for Standardization, Dec 2010.

ISO/IEC Standards Committee 25010:2011. Systems and software engineering - systems and software quality requirements and evaluation (square) - system and software quality models. Standard, International Organization for Standardization, Mar 2011.

IEEE Standards Committee 610-1990. Ieee standard computer dictionary: A compilation of ieee standard computer glossaries. *IEEE Std 610*, pages 1–217, Jan 1991a. doi: 10.1109/IEEESTD.1991.106963.

IEEE Standards Committee 610-1990. Ieee standard glossary of software engineering terminology. Standard, IEEE, 1991b.

Kevin MacG Adams et al. *Nonfunctional requirements in systems analysis and design*, volume 28. Springer, 2015.

AFUL. Definition of interoperability. http://interoperability-definition.info/en/, 2019. [Online; accessed 1-November-2019].

F. Benureau and N. Rougier. Re-run, Repeat, Reproduce, Reuse, Replicate: Transforming Code into Scientific Contributions. *ArXiv e-prints*, August 2017.

Patrik Berander, Lars-Ola Damm, Jeanette Eriksson, Tony Gorschek, Kennet Henningsson, Per Jönsson, Simon Kågström, Drazen Milicic, Frans Mårtensson, Kari Rönkkö, et al. Software quality attributes and trade-offs. *Blekinge Institute of Technology*, 2005.

Nigel Bevan. Measuring usability as quality of use. *Software Quality Journal*, 4(2):115–130, 1995.

B. W. Boehm. Verifying and validating software requirements and design specifications. *IEEE Software*, 1(1):75–88, Jan 1984. doi: 10.1109/MS.1984.233702.

Barry W. Boehm. Improving software productivity. *Computer*, pages 43–47, 1987.

Barry W Boehm. *Software engineering: Barry W. Boehm's lifetime contributions to software development, management, and research*, volume 69. John Wiley & Sons, 2007.

G. H. Brundtland. *Our Common Future*. Oxford University Press, 1987. URL https://EconPapers.repec.org/RePEc:oxp:obooks:9780192820808.

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.

Jean-Claude Fernandez, Laurent Mounier, and Cyril Pachon. A model-based approach for robustness testing. In *IFIP International Conference on Testing of Communicating Systems*, pages 333–348. Springer, 2005.

International Organization for Standardization/International Electrotechnical Commission et al. Iso/iec 9126–software engineering–product quality, 2001.

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of software engineering*. Prentice Hall PTR, 1991.

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

Robert Heine. What is sustainable software? https://www.energypedia-consult.com/en/blog/robert-heine/what-sustainable-software, July 2017.

HIMSS. What is interoperability? https://www.himss.org/library/interoperability-standards/what-is-interoperability, 2019. [Online; accessed 1-November-2019].

Henrik Hygerth. Sustainable software engineering : An investigation into the technical sustainability dimension. Master's thesis, KTH, Sustainability and Industrial Dynamics, 2016.

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

IISD International Institute for Sustainable Development. Sustainable development. https://www.iisd.org/topic/sustainable-development, October 2019.

EN ISO-IEC. 17043: 2010. *ISO: Geneva, Switzerland*, 2010.

Panagiotis Kalagiakos. The non-technical factors of reusability. In *Proceedings of the 29th Conference on EUROMICRO*, page 124. IEEE Computer Society, 2003.

Daniel Katz. Defining software sustainability. https://danielskatzblog.wordpress.com/2016/09/13/defining-software-sustainability/, September 2016.

B. Kitchenham and E. Mendes. Software productivity measurement using multiple size measures. *IEEE Transactions on Software Engineering*, 30(12):1023–1035, Dec 2004. doi: 10.1109/TSE.2004.104.

Charles W. Krueger. Software reuse. *ACM Comput. Surv.*, 24(2):131–183, June 1992. ISSN 0360-0300. doi: 10.1145/130844.130856. URL http://doi.acm.org/10.1145/130844.130856.

J. McCall, P. Richards, and G. Walters. *Factors in Software Quality*. NTIS AD-A049-014, 015, 055, November 1977.

Bertrand Meyer. *Object-oriented software construction*, volume 2. Prentice hall New York, 1988.

James D. Mooney. Strategies for supporting application portability. *Computer*, 23(11):59–70, 1990.

D. L. Parnas. Software aging. In *Proceedings of 16th International Conference on Software Engineering*, pages 279–287, May 1994. doi: 10.1109/ICSE.1994.296790.

Birgit Penzenstadler and Henning Femmer. A generic model for sustainability with process- and product-specific instances. In *Proceedings of the 2013 Workshop on Green in/by Software Engineering*, GIBSE '13, pages 3–8, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1866-2. doi: 10.1145/2451605.2451609. URL http://doi.acm.org/10.1145/2451605.2451609.

Atlee Pfleeger. *Software Engineering Theory and Practice - Third Edition*. Pearson Education, 2006.

Roger S Pressman. *Software engineering: a practitioner's approach*. Palgrave Macmillan, 2005.

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

Parvinder S Sandhu, Priyanka Kakkar, Shilpa Sharma, et al. A survey on software reusability. In *2010 International Conference on Mechanical and Electrical Technology*, pages 769–773. IEEE, 2010.

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.

Kevin Tate. *Sustainable Software Development: An Agile Perspective*. Addison-Wesley Professional, 2005. ISBN 0321286081.

Hans van Vliet. *Software Engineering (2nd ed.): Principles and Practice*. John Wiley & Sons, Inc., New York, NY, USA, 2000. ISBN 0-471-97508-7.

CC Venters, C Jay, LMS Lau, MK Griffiths, V Holmes, RR Ward, J Austin, CE Dibsdale, and J Xu. Software sustainability: The modern tower of babel, 2014. URL http://

eprints.whiterose.ac.uk/84941/. (c) 2014, The Author(s). This is an author produced version of a paper published in CEUR Workshop Proceedings. Uploaded in accordance with the publisher's self-archiving policy.

Wiegers. *Software Requirements, 2e*. Microsoft Press, 2003.

Matthew Wilson. Quality matters: Correctness, robustness and reliability. *Overload*, 17:93, 2009.

Nina Wolfram, Patricia Lago, and Francesco Osborne. Sustainability in software engineering. In *2017 Sustainable Internet and ICT for Sustainability, SustainIT 2017, Funchal, Portugal, December 6-7, 2017*, pages 55–61, 2017. doi: 10.23919/SustainIT.2017.8379798. URL https://doi.org/10.23919/SustainIT.2017.8379798.

Didar Zowghi and Vincenzo Gervasi. On the interplay between consistency, completeness, and correctness in requirements evolution. *Information and Software Technology*, 45(14):993 – 1009, 2003. ISSN 0950-5849. doi: https://doi.org/10.1016/S0950-5849(03)00100-9. URL http://www.sciencedirect.com/science/article/pii/S0950584903001009. Eighth International Workshop on Requirements Engineering: Foundation for Software Quality.