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
donga9@mcmaster.ca

Abstract

We have found some of the existing definitions of software qualities inadequate as a basis for conducting and presenting our software engineering research. Therefore, we have endeavored to create a list of definitions that are consistent (same words/phrases are always used for the same concepts), measurable (all qualities are conceptually measurable, even if the data needed for measurement is often unavailable), abstract (quality definitions state what the quality achieves, but not how to achieve it) and traceable (explicit links are given to all previous definitions and the ideas within those definitions). Our methodology starts with an exhaustive literature survey to find existing definitions. We then determine the recommended definition, which is either one of the existing definitions, or a new definition, found by combining existing definitions. Each recommended definition is accompanied by a rationale that explicitly justifies the definition. By presenting the rationales, we hope our software engineering colleagues will scrutinize our work and provide recommendations for future modifications. The qualities that are defined include installability, correctness, reliability, robustness, performance, usability, verifiability, validatability, testability, maintainability, reusability, portability, understandability, interoperability, productivity, visibility/transparency, reproducibility, sustainability, completeness, consistency, modifiability, traceability, unambiguity and abstract. Each quality is classified into one or more of the following categories: direct, indirect, internal, external, product or process.

2012 ACM Subject Classification Software and its engineering

Keywords and phrases software qualities, installability, correctness, verfiability, validatability, testability, reliability, robustness, performance, usability, maintainability, reusability, portability, understandability, interoperability, visibility/transparency, reproducibility, productivity, sustainability, completness, consistency, modifiability, traceability, unambiguity, verifiability, abstract

Contents

1 Introduction

In the course of conducting and presenting our software engineering related research we have often desired clear definitions for relevant software qualities, like maintainability, usability, productivity, sustainability, etc. Unfortunately, the definitions of qualities that we have found in the literature are not always satisfying. This inadequacy of existing definitions became particularly clear when we recently undertook an exercise measuring the "state of the practice" in different software domains, including the domains of seismology software [Smith et al., 2018b], Geographic Information Systems [Smith et al., 2018a] and mesh generators [Smith et al., 2016]. When determining methods to quickly assess quality across large samples of existing software, we found communication amongst ourselves and with others challenging. We craved unambiguous and standardized definitions to enable and inspire methods for measuring software qualities. To help ground our future research, and possibly help others in the same situation, we have collected an exhaustive list of definitions. From this list we have produced our set of recommended software quality definitions.

In the current work, software is defined as either programs, which run (execute) on a computer, or libraries, which provide services to be used by programs. Both programs and libraries are created using computer code, written in a programming language.

Quality is defined as a measure of the excellence or worth of an entity. As is standard practice, we do not think of quality as a single measure, but rather as a set of measures. That is, quality is a collection of different qualities, often called "ilities." In the overview of managing software quality provided by van Vliet [2000, p. 110] the different measures are called quality attributes, quality factors, and quality criteria. In ISO/IEC [2001] they are called quality characteristics, sub-characteristics and attributes. To keep our presentation simple, and to not imply a classification hierarchy that we don't intend, we will just use the term qualities to refer to the different measures of quality.

The qualities of most interest to project managers and users are summarized in the next subsection (Section 1.1). Since these qualities cannot generally be measured directly, they are characterized as indirect qualities. The relevant indirect qualities are listed in this section and each quality is grouped into one or more of the following categories: external, internal, product or process. The next subsection lists the relevant direct qualities that assist in achieving the indirect qualities. The final subsection presents the methodology that was employed to summarize, analyze and define each quality.

1.1 Indirect Qualities

Many taxonomies have been proposed for software qualities, as in McCall et al. [1977], Boehm et al. [1976], ISO/IEC [2001]. Rather than aiming for complete coverage of the history of qualities, the wide variety of potential qualities, or the relationships between them, we will instead take a practical approach; we will focus on the qualities that we have found most relevant in our research. This means we will include the usual software engineering suspects, like usability and maintainability. Since we also do research on scientific computing software, we will also include scientifically relevant qualities, like validatability and reproducibility.

Given our respect for Ghezzi et al. [2003, p. 15–33], we used their list of qualities as a starting point. Our list, shown in Table 1, includes all of their qualities, except for timeliness, which was excluded because the already included quality of productivity sufficiently overlaps with timeliness. To the list from Ghezzi et al. [2003] we have added the practical quality of installability, since our aforementioned studies on the state of the practice in different software domains (for instance Smith et al. [2018a]) have shown us that none of the other

qualities matter if the software cannot be properly installed. Given our interest in scientific computing, we also added the qualities of validatability and reproducibility. Since we found few definitions in the literature on the quality of verifiability, we also added the closely related quality of testability. The quality of sustainability was added since this term has emerged as a popular goal for software projects.

The order of the qualities in Table 1 follow the order used in Ghezzi et al. [2003], which puts related qualities (like correctness and reliability) together. Moreover, the order is roughly the order in which qualities are considered in practice. We followed these same guidelines when inserting our additional qualities into the list.

	External	Internal	Product	Process
Installability	✓		✓	
Correctness	✓	✓	✓	✓
Reliability	✓		✓	✓
Robustness	✓		✓	✓
Performance	✓		✓	
Usability	✓		✓	✓
Verifiability		✓	✓	✓
Validatability	✓		✓	✓
Testability		✓	✓	
Maintainability		✓	✓	✓
Reusability		✓	✓	✓
Portability		✓	✓	
Understandability		✓	✓	
Interoperability	✓		✓	
Productivity				✓
Visibility				✓
Reproducibility	✓		✓	
Sustainability		✓	✓	

Table 1 Software Qualities

Table 1 makes a distinction between external and internal qualities. External qualities are the qualities that are relevant to the user of the software while internal qualities are those that are relevant to developers [Ghezzi et al., 2003, p. 16]. Although the users are not typically interested in internal qualities, it is the internal qualities that influence external quality. External qualities are assessed using the produced software and any user relevant documentation. Internal qualities reference such things as the code, documentation, test cases, and the software development process. Correctness is shown as both an internal and external quality because it is possible to judge the correctness of software and to judge the correctness of the various internal artifacts produced while developing the software. The external quality of reliability is influenced by the internal quality of verifiability. Since validatability is related to getting the right requirements, it is shown as an external quality. Although one could judge the usability of internal documentation, the quality of usability is considered as external, since the quality of understandability designated to cover the internal equivalent of usability.

Some qualities focus on the product, others on the process, and yet others on both, as

shown in Table 1. The term product refers to both the external and internal products. The external products are the user-visible products, like the software itself and any user documentation. The internal products are the products (sometimes called work products or artifacts) relevant to the developers, like requirements documentation, design documentation, test cases, makefiles, etc. The process refers to the sequence of activities that produce the software. The process generates both the internal and external products. The process is based on principles, methods and techniques while being supported by tools. All product qualities (internal and external) are effected by the process, so the checkmarks in Table 1 are used for a different purpose. The checkmarks show which qualities are applied to the process itself. For instance, we can consider the correctness and maintainability of a process, but discussing the portability of a process makes little sense. Rather than apply the quality of performance to a process, we instead introduce the process specific quality of productivity. Although we can imagine discussing the understandability of a process, to avoid confusion and overlap, we reserve understandability for the internal quality applied to products (like code). For the related process quality, we designate the term usability.

The qualities shown in Table 1 are qualities that can only be measured indirectly [van Vliet, 2000, p. 109]. For instance, a direct measure of maintainability doesn't exist. Since it cannot be measured directly, we base our estimation of maintainability on direct measures that are likely to influence maintainability, like consistency, traceability, etc. Qualities that can be measured directly are discussed in the next section.

1.2 Direct Qualities

To achieve the qualities listed in Section 1.1, conventional wisdom holds that the internal products should aim for the qualities listed in this section. All but the final quality listed (abstraction), are adapted from the IEEE recommended practice for producing good software requirements [IEEE, 1998]. Abstraction is an important addition because it is the software development principle that allows us to deal with complexity [Ghezzi et al., 2003, p. 40]. IEEE [1998] also includes correctness and verifiability, but we have excluded these from our list, since they belong in the list of indirect qualities in the previous section.

- Completeness
- Consistency
- Modifiability
- Traceability
- Unambiguity
- Abstract

All of the direct qualities listed above are internal qualities. Furthermore, they apply to both the (internal) product and the process.

1.3 Methodology

For each quality we collect as many distinct instances of the definition that we can find. Some of the definitions occur in multiple places (for instance the definitions from McCall et al. [1977] appear in many software engineering textbooks). Since these instances do not add any new information, they are not included; we do not make an effort to identify how frequently the definitions reappear in different sources. Once all the definitions are collected, we determine the definition that we would like to recommend. The recommended definition

can either be our preference from the existing definitions, or a new definition, which is often found by combining existing definitions.

To ensure that our recommended definitions are quality definitions, they are based on the direct quality measures (Section 1.2). To the list of direct qualities, we also add the quality criteria of measurability. How the direct qualities are applied to develop quality definitions of qualities is described below.

Completeness To judge the completeness of our definitions, we verify that all of the relevant points from the collected definitions have been included, or explicitly rejected in the definition's rationale. We also verify that all facets of the definition are covered when a quality applies to multiple categories of internal, external, product and/or process. Finally, we check that all of the qualities listed in this section have been considered.

Consistency The final list of definitions should be consistent in the terminology used. For instance, we do not use synonyms; we use the same word or phrase to mean the same concept throughout the final list of definitions. Specific decisions on terminology are as follows:

- To represent the idea of cost, effort, time, resources, and efficiency, we say *effort*.
- To represent "the degree to which" or the "the extent to which", we will say *the extent to which*.
- [Fill in additional consistency decisions, as we go through definitions. —SS]

Modifiability The modifiability of the definitions is aided by the fact that the definitions are short. Changes are not difficult to make. The modifiability is further aided by the related qualities of traceability and abstraction.

Traceability Each definition includes a section for the rationale. The rationale explicitly links the recommended definition to the previously published definitions. The rationale also explains why any aspects of the previous definitions were ignored. The rationale also explicitly traces the selected definition to the qualities in this section.

Unambiguity With natural language it is difficult to remove all ambiguity, but every effort is made to keep the definitions simple and to use standard and consistent terminology. The rationale given for each recommended definition also serves the purpose of making the definitions unambiguous.

Abstract Quality definitions should say what the quality achieves, but not how to achieve it. We aim for abstract definitions that do not assume any specific methodology or tools. For those qualities that fit into multiple categories of internal, external, product and/or process, we aim to have one definition that will apply to all. When that is not possible, we will clarify how the definition is modified between categories.

Measurability Although many qualities can only be indirectly measured, the definition of the quality should imply a measure that is conceptually possible, even if the data needed to complete the measurement is often unavailable. The scale type of the units of measure are as follows, in order of preference: ratio, interval, ordinal and nominal. These scale types are defined in van Vliet [2000, p. 107]. The ratio and interval scale types are preferred to the ordinal and nominal types because they can convey more information. Specific consequences of aiming for measurability are as follows:

- We will avoid the common phrase in definitions of "the capability of" because this implies a binary measure. By this definition a product (or process) is either capable or not; it is a binary measure. Instead of measuring capability, we will prefer to use the phrase "the effort required".
- [Add to this list as measurability specific decisions are made. —SS]

Our aim is to capture the essence of the quality in an unambiguous way, even if it cannot be measured. This is analogous to the definition of true error in scientific computing. The true error, which is the difference between the calculated and the true solution, can rarely be calculated since the true solution is generally unknown. Even though true error can only be estimated, the concept of true error is integral to analyzing and understanding the behaviour of numerical algorithms.

After completing each definition we verify that the definition works for all of the quality categories (internal, external, product and process) that apply. If not, the definition is modified. The rationale section is written to explicitly addresses each of these points. Each definition is also checked to verify that it is complete, consistent, traceable, unambiguous, abstract and measurable. The rational section is written to explain how these qualities, for each quality definition, are achieved.

2 Indirect Qualities of Software, Artifacts and Processes

The presentation below for each of the qualities is the same. First, we summarize, in the order of their publication, 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. In cases where the definition has to be rephrased, quotation marks are used to show those portions that are taken verbatim for the original source. Second, after the summary of the existing definitions, we propose the definition that we would like to work with going forward. Third, following the proposed definition is an explanation for the reasoning that led to this choice. The definitions apply to internal, external, product and process following the designations in Table 1. Where necessary, this is clarified in the definition and its rationale.

2.1 Installability (external, product)

▶ **Definition 1.** The capability of the software product to be installed in a specified environment [ISO/IEC, 2001].

▶ **Definition 2.** The degree of effectiveness and efficiency with which a product or system can be successfully installed and/or uninstalled in a specified environment [ISO/IEC, 2011].

▶ **Definition 3.** Installability refers to the cost or effort required for the installation, given an installation is possible to begin with [Lenhard et al., 2013].

Installability

The effort required for the installation and/or uninstallation of software in a specified environment.

Reasoning

The recommended definition comes from combining Definitions 2 and 3. The rationales for the proposed definition are as follows:

- For consistency we use the term **effort** in place of efficiency.
- The "effectiveness" from Definition 2 is dropped because the success or failure of an installation is typically judged as a binary result. The installation is generally either completed, or not.

- Although we are okay with a binary result for success or failure of an installation, the measure of effort required to complete an installation is not a binary measure. For this reason, we do not use the term "the capability of" (from Definition 1), so that we can be consistent with our other recommended definitions where we avoid that term for measurability reasons.
- From Definition 3 the phrase "given an installation is possible to begin with" is removed because this notion is covered by the measure of effort. The effort will be infinite if an installation is impossible.

2.2 Correctness (external, internal, product, process)

▶ **Definition 4.** The extent to which a program satisfies its specifications and fulfills the user's mission objectives [McCall et al., 1977].

▶ **Definition 5.** The ability of software products to perform their exact tasks, as defined by their specification [Meyer, 1988].

▶ **Definition 6.** The degree to which a system or component is free from faults in its specification, design and implementation [IEEE, 1991a].

▶ **Definition 7.** A program is functionally correct if it behaves according to its stated functional specifications [Ghezzi et al., 2003].

▶ **Definition 8.** The degree to which a system is free from faults in its specification, design, and implementation [McConnell, 2004].

Correctness

An entity is correct if it matches its specification. The entity can be either software, an internal product (work product or artifact) or process. The specification can either be explicitly stated or implicit.

Reasoning

The selection definition is a rephrasing of Definition 7. All of the other definitions (Definition 4, Definition 5, Definition 6 and Definition 8) also mention specification. The rationale for the proposed definition follows:

- Definition 4 mentions "mission objectives". We feel that the mission objectives are incorporated into the specification, so this element is not need to be mentioned separately in the recommended definition.
- Definition 7 refers to the functional specification, but we believe that correct software also follows the nonfunctional specification. For instance, for safety critical software the performance requirements need to be satisfied, or the software will not be trustworthy and therefore not correct.
- Definition 6 and Definition 8 are almost the same, except Definition 6 mentions "system or component" rather than just "system". To keep our recommended definition simple, we do not introduce the distinction of components; we just focus on the entities of the software, internal products and processes.

- Several of the previous definitions imply measurement of different levels of correctness by phrases such as "The extent to which" (Definition 4) and "The degree to which" (Definitions 6 and 8). However, the proposed definition has a binary range of possible measurement – either correct or incorrect. This doesn't follow our stated goal (in Section 1.3) of having the quality of measurement cover the largest possible scale type. Like the recommended definition, Definition 6 also implies a binary measure with the phrase "the ability of". Making correctness a binary quality makes the definition unambiguous and it frees up room for the quality of reliability to cover the more nuanced aspects of how an entity is "correct".
- Like the recommended definition, Definition 6 implies that correctness can be judged for products. Definition 6 mentions the "specification, design and implementation".
- So that our definition of correctness can be abstract and thus also apply to internal products, the definition mentions internal products, and states that the specification can be implicit. To be correct an internal product needs to follow any specified template or documentation conventions. Moreover, although they may not be stated explicitly, the requirements for a document will be implicitly specified in the minds of the products authors/creators.
- So that our definition of correctness can be abstract and thus also apply to processes, the definition mentions processes and states the specification can be implicit. If the rules of the process are explicitly specified, then developers can check whether the rules have been followed. Even when the rules are not stated in writing, the developers will have implicit (possibly changing and inconsistent) rules in their minds.
- The definition for correctness allows the specification to be implicit, even for the external product of the software itself. Not every project writes down the requirements, but the requirements still exist inside the heads of the developers. This is why testing can be done for software, even when the developers do not specify their requirements.

2.3 Reliability (external, product, process)

▶ **Definition 9.** Code possesses the characteristic reliability to the extent that it can be expected to perform its intended functions satisfactorily [Boehm et al., 1976].

▶ **Definition 10.** Extent to which a program can be expected to perform its intended function with required precision [McCall et al., 1977].

▶ **Definition 11.** It is the probability of failure-free operation of a computer program in a specified environment for a specified time [Musa et al., 1987].

▶ **Definition 12.** The ability of a system or component to perform its required function under stated conditions for a specified period of time [IEEE, 1991a].

▶ **Definition 13.** The capability of the software product to maintain a specified level of performance when used under specified conditions ISO/IEC [2001].

▶ **Definition 14.** Informally, software is reliable if the user can depend on it [Ghezzi et al., 2003].

▶ **Definition 15.** Reliability is concerned with measuring the probability of occurrence of failure - that is, the probability of the observable effects of errors [Ghezzi et al., 2003].

▶ **Definition 16.** The ability of a system to perform its required functions under stated conditions whenever required—having a long mean time between failures [McConnell, 2004].

▶ **Definition 17.** Informally, the reliability of a system is the probability, over a given period of time, that the system will correctly deliver services as expected by the user [Sommerville, 2011].

▶ **Definition 18.** Reliability expresses the ability of the software to maintain a specified level of fault tolerance, when used under specified condition [Singh, 2013].

Reliability

Probability of failure-free operation of a computer program in a specified environment for a specified time, i.e. the average time interval between two failures also known as the mean time to failure (MTTF).

Reasoning

Definition 11 and 15 rephrased:. Reliability is defined such that the measurement metrics are visible - the specified environment and time. The different reasons for the proposed definition are as follows:

- reason 1
- reason 2
- ...

2.4 Robustness (external, product, process)

▶ **Definition 19.** The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions [IEEE, 1991b].

▶ **Definition 20.** 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]. [Use Ghezzi 2003 instead? —SS]

▶ **Definition 21.** 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 22.** 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].

Robustness

Definition 22 and Definition 20 rephrased: Software possesses the characteristic of robustness if it behaves "reasonably" in two situations: i) when it encounters circumstances not anticipated in the requirements specification; and ii) when the assumptions in its requirements specification are violated.

Reasoning

This definition indicates that robustness is related to the quality of correctness (both within and outside of it). [Still may want to refine what we mean by reasonable —PM] The different reasons for the proposed definition are as follows:

- reason 1
- reason 2
- ...

2.5 Performance (external, product)

▶ **Definition 23.** The degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage [IEEE, 1991b].

▶ **Definition 24.** In software engineering we often equate performance with efficiency. A software system is efficient if it uses computing resources economically [Ghezzi et al., 1991]. [Change to 2003 reference? —SS]

▶ **Definition 25.** 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 (concurrent usage loads), and timing (hard real-time demands) [Wiegers, 2003].

Performance

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 (concurrent usage loads), and timing (hard real-time demands). [What is the difference between speed and timing? —SS]

Reasoning

The definition is found by combining Definition 23 and Definition 25. This definition offers a comprehensive list of constraints that are commonly associated with software performance, such as speed, throughput, capacity, and timing. [I will check if Wiegers discusses the difference between speed and timing. —PM] [Wiegers did not discuss this difference. —PM] [Speed - change over time; timing - related to specification: when something should happen in relation to something else - state transitions —PM] [If we need both speed and timing, we should define them. We aren't including accuracy from the IEEE definition. Is that on purpose? —SS] [We have decided not to include accuracy. SS will find difference b/w speed and timing from a colleague. —PM] The different reasons for the proposed definition are as follows:

- reason 1
- reason 2
- ...

2.6 Usability (external, product, process)

▶ **Definition 26.** "The effort required to learn, operate, prepare input, and interpret output of a program" [McCall et al., 1977].

▶ **Definition 27.** "A software is usable - or user friendly - if its human users find it easy to use" [Ghezzi et al., 1991]. [switch to 2003 ref? —SS]

▶ **Definition 28.** 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. **?** and **?**

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

▶ **Definition 29.**
- 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). Nielsen [2012]

▶ **Definition 30.** Desirable outcomes of usability:

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.

From the Interaction Design Foundation https://www.interaction-design.org/literature/topics/usability.

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

▶ **Definition 31.** The capability of the software to be understood, learned, used and liked by the user, when used under specified conditions ISO/IEC [2001].

Usability

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. **?** and **?**

Reasoning

There has been a lot of thought put behind the ISO definition, from experts. The standard has been revisited, but the definition has not changed, which means that it has stood the test of time. The different reasons for the proposed definition are as follows:

- reason 1
- reason 2
- ...

2.7 Verifiability (internal, product, process)

▶ **Definition 32.** Verification is the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase [IEEE, 1991b].

▶ **Definition 33.** Verification involves solving the equations right [Roache, 1998, p. 23].

▶ **Definition 34.** A software is verifiable if its properties can be verified easily [Ghezzi et al., 2003]. [Is this a direct quote? —SS] [yes —OO]

▶ **Definition 35.** By verification, we mean all activities that are undertaken to ascertain that the software meets its objectives [Ghezzi et al., 2003].

▶ **Definition 36.** Verifiability refers to how well software components or the integrated product can be evaluated to demonstrate whether the system functions as expected [Wiegers, 2003].

▶ **Definition 37.** Verifiability "means that you should be able to write a set of tests that can demonstrate that the delivered system meets each specified requirement" [Sommerville, 2011].

▶ **Definition 38.** The evaluation of whether or not a product, service, or result complies with a regulation, requirement, specification, or imposed condition [PMI, 2017] [verification definition —OO]

- 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] [Need a proper citation —SS]
  - 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] [Need a proper citation —SS]

"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]. [Need to verify McCall reference, then delete VanVliet. —SS] [McCall's definition verbatim: Effort required to test a program to insure it performs its intended function. —PM]

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

▶ **Definition 39.** A software system is verifiable if its properties can be verified easily [Ghezzi et al., 1991].

Verifiability

Definition 37 rephrased: Verifiability is the extent to which a set of tests can be written and executed, to demonstrate that the delivered system meets the specification.

Reasoning

Definition is concise and measurable. Verifiability involves solving the equations right [Roache, 1998, p. 23] and the definition states that system needs to be verifiable based on the specifications. When applied to process means "did we apply the process correctly", as opposed to validatability, we refers to "did we apply the correct process". The different reasons for the proposed definition are as follows:

- reason 1
- reason 2
- ...

2.8 Validatability (external, product, process)

▶ **Definition 40.** The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements [IEEE, 1991b].

▶ **Definition 41.** Validation means solving the right equations [Roache, 1998, p. 23].

▶ **Definition 42.** Software validation is achieved through a series of tests that demonstrate conformity with requirements [Pressman, 2005].

▶ **Definition 43.** Validation is concerned with establishing that the product fulfills its intended use [Van Vliet et al., 2008].

▶ **Definition 44.** Software validation is the process of checking that the system conforms to its specification and that it meets the real needs of the users of the system [Sommerville, 2011].

▶ **Definition 45.** Software validation, where the software is checked to ensure that it is what the customer requires [Sommerville, 2011].

▶ **Definition 46.** The assurance that a product, service, or result meets the needs of the customer and other identified stakeholders [PMI, 2017].

Validatability

Definition 40 rephrased: The effort required in evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements

Reasoning

Definition describes that system should be measurable through the development life cycle, It is concise and can be measured. The different reasons for the proposed definition are as follows:

- reason 1
- reason 2
- ...

2.9 Testability (internal, product)

▶ **Definition 47.** Code possesses the characteristic testability to the extent that it facilitates the establishment of verification criteria and supports evaluation of its performance [Boehm et al., 1976]

▶ **Definition 48.** Effort required to test a program to ensure it performs its intended function [McCall et al., 1977].

▶ **Definition 49.** 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 [IEEE, 1991b].

▶ **Definition 50.** The degree to which you can unit-test and system-test a system; the degree to which you can verify that the system meets its requirements [McConnell, 2004].

Testability

Definition 49: 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.

Reasoning

Definition is concise and measurable. The different reasons for the proposed definition are as follows:

- reason 1
- reason 2
- ...

2.10 Maintainability (internal, product, process)

▶ **Definition 51.** Effort required to locate and fix an error in an operational program [McCall et al., 1977].

[It looks like Pressman [2005] was using McCall et al. [1977] for their definition. —SS] [PM - can you please look into whether Pressman [2005] cite McCall et al. [1977]? Do they

give a reason for dropping the word operational? —SS] [Yes, Pressman [2005] does cite the relevant section of the textbook with McCall et al. [1977]. While they do not give a reason for specifically dropping the word operational, they do indicate that their given definition "is a very limited definition". Perhaps their intent was breadth. —PM] [Peter, please verify the McCall reference definition, and then just cite that. Olu has found the McCall reference, so you shouldn't have to search for it. We can remove both the VanVliet and Pressman reference in preference of the McCall definition. —SS]

[I have verified the McCall definition and removed the Pressman and van Vliet references above. I will leave these comments until all similar definitions in this document have been edited. —PM]

▶ **Definition 52.** 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 [IEEE, 1991b].

▶ **Definition 53.** 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]. [update to 2003 ref? —SS]

▶ **Definition 54.** The capability of the software to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications. ISO/IEC [2001]

▶ **Definition 55.** 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 56.** Code possesses the characteristic of maintainability to the extent that it facilitates updating to satisfy new requirements or to correct deficiencies [Boehm, 2007].

▶ **Definition 57.** 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 [ISO/IEC, 2011].

Maintainability

Combined altered Definition 52 and Definition 56: The effort with which a software system or component can be modified to:

1. correct faults
2. improve performance or other attributes
3. satisfy new requirements

Reasoning

This definition offers a comprehensive list of potential reasons for modifying software, such as correcting faults, improving performance or other attributes, or satisfying new requirements.

[What are the other attributes? What is an attribute? Is it the same as a quality? We are calling performance a quality in this document. —SS]

[The term quality attribute has come up often in the literature - qualities appear to be a subset of attributes. Quality attributes seem to be associated with non-functional

requirements. The term resource attribute was encountered in a journal and implied an association with a functional requirement. The two might differ along those lines of NFR and FR. —PM]

[I think this is done —PM]

The different reasons for the proposed definition are as follows:

- reason 1
- reason 2
- ...

2.11 Reusability (internal, product, process)

▶ **Definition 58.** Extent to which a program can be used in other applications - related to the packaging and scope of the functions that programs perform [McCall et al., 1977].

▶ **Definition 59.** The degree to which a software module or other work product can be used in more than one software system [IEEE, 1991b].

▶ **Definition 60.** A product is reusable if we can "use the product - perhaps with minor changes - to build another product" [Ghezzi et al., 1991]. [Update to 2003? —SS]

▶ **Definition 61.** 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 62.** 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].

Reusability

Definition 61: 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.

2.12 Portability (internal, product)

▶ **Definition 63.** Effort required to transfer a program from one hardware configuration and/or software system environment to another [McCall et al., 1977].

▶ **Definition 64.** The ease with which a system or component can be transferred from one hardware or software environment to another [IEEE, 1991b].

▶ **Definition 65.** 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 66.** Portability refers to the ability to run a system on different hardware platforms [Ghezzi et al., 1991]. [update to 2003? —SS]

▶ **Definition 67.** The capability of software to be transferred from one environment to another ISO/IEC [2001].

▶ **Definition 68.** 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 69.** 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].

Portability

Definition 63 rephrased: Effort required to transfer a program between system environments (including hardware and software).

Reasoning

This is measurable and succinct. The different reasons for the proposed definition are as follows:

- reason 1
- reason 2
- ...

2.13 Understandability (internal, product)

▶ **Definition 70.** The effort "to uncover the logic of the application" [Ghezzi et al., 1991].

▶ **Definition 71.** The capability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use ISO/IEC [2001].

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.

Understandability

[Still needs to be completed —SS]

Reasoning

[Needs to be completed —SS]

2.14 Interoperability (external, product)

▶ **Definition 72.** Effort required to couple one system with another [McCall et al., 1977]. [Ao, if this is from McCall, we should just cite the primary source and remove the Van Vliet reference. —SS] [Yes this is almost exactly from McCall, except he did not have the word "The" at the beginning. —PM] [Someone with access to the book added it. I found the book in the pub folder and fixed it. —AD]

▶ **Definition 73.** Interoperability is the ability of two or more systems or components to exchange information and to use the information that has been exchanged [IEEE, 1991a].

▶ **Definition 74.** The ability of a system to coexist and cooperate with other systems. [Ghezzi et al., 1991]. [2003 ref? —SS]

▶ **Definition 75.** The capability of the software to interact with one or more specified systems ISO/IEC [2001].

▶ **Definition 76.** 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 [ISO/IEC/IEEE, 2010].

▶ **Definition 77.** The degree to which two or more systems, products or components can exchange information and use the information that has been exchanged [ISO/IEC, 2011].

▶ **Definition 78.** 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 79.** 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

Interoperability

Definition 77.

Reasoning

This definition is concise and also detailed enough to show the concept not only on system, but also on products and components. It also covered the concept with more than 2 systems. The different reasons for the proposed definition are as follows:

- reason 1
- reason 2
- ...

2.15 Productivity (process)

▶ **Definition 80.** The best definition of the productivity of a process is

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

Thus, we can improve the productivity of the software process by increasing its outputs, decreasing its inputs, or both. However, this means that we need to provide meaningful definitions of the inputs and outputs of the software process. [Where did this definition come from? Is this all from Boehm? This is quite a long quote. We might actually want to paraphrase. —SS] [all from Boehm. I feel the best way is deleting everything after the equation —AD]

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. However, one has to be careful which of various classes of items are to be counted as inputs. For example:

- Phases (just software development, or should we include system engineering, soft-ware requirements analysis, installation, or post development support?)
- Activities (to include documentation, project management, facilities management, conversion, training, database administration?)
- Personnel (to include secretaries, computer operators, business managers, contract administrators, line management?)
- Resources (to include facilities, equipment, communications, current versus future dollar payments?)

An organization can usually reach an agreement on which of the above are meaningful as inputs in their organizational context. Frequently, one can use present-value dollars as a uniform scale for various classes of resources.

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]. [Is this a direct quote from Boehm [1987]? The sentences seem incomplete? —SS] [I added some deleted parts, now it is a direct quote. After the ending of the quote, the following 2 pages discuss the flaws of DCI and a list of alternatives to DCI, so I ended the quote here. —AD]

▶ **Definition 81.** A quality of the software production process, referring to its efficiency and performance [Ghezzi et al., 1991]. [2003 ref? —SS]

▶ **Definition 82.** The number of lines of new code developed per person-day (an imperfect measure of productivity but one that could be measured consistently) [MacCormack et al., 2003].

▶ **Definition 83.** Productivity is the amount of output (what is produced) per unit of input used. In general, productivity is difficult to measure because outputs and inputs are typically quite diverse and are often themselves difficult to measure. In the context of software, productivity measurement is usually based on a simple ratio of product size to project effort. Thus,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].

Productivity

The revision of the combination of Definition 83 and Definition 82: Productivity is the amount of output per unit of input used, which can be measured by the summation of all output (such as the number of lines of new code, the number of pages of new documents and the number of new test cases) produced per person-day. [Use Long Term Productivity paper —SS]

Reasoning

It is concise and measurable. [What is the output? What is the input? I think the definition needs to give more information on these. In particular, the above definitions focus on code as the output, but documentation, test cases etc should also be part of the output. If we are going to measure this, we need a better idea of what we are measuring for outputs and inputs. —SS] [I added another def and made it more measurable. —AD] The different reasons for the proposed definition are as follows:

- reason 1
- reason 2
- ...

2.16 Visibility/Transparency (process)

▶ **Definition 84.** 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]. [change to 2003 ref? —SS]

▶ **Definition 85.** Business process visibility, also called process visibility, is the ability to accurately and completely view the processes, transactions and other activities operating within an enterprise [Rouse and Stuart, 2013].

▶ **Definition 86.** Visibility provides transparency into the development process. It is the ability to see progress at any point and determine the distance to completion of a goal. Visibility provides status of not only the progress of the project, but the product itself [GSA, 2019].

▶ **Definition 87.** Process transparency refers to the ability to look inside. The "look inside" provides an in-depth and clear visibility into the business processes and how these operate [PRIME, 2019].

▶ **Definition 88.** The degree to which something is seen by the public [Cambridge Dictionary, 2019c].

The degree to which something is seen or known about [Cambridge Dictionary, 2019c].

Visibility/Transparency

Definition 84 rephrased: The extent to which all of the steps of a software development process and the current status of it are conveyed clearly. [I wonder if rather than "documented clearly", we should be more abstract and talk about how these things can be easily determined. Documentation is probably how the information will be conveyed, but it doesn't have to be how it is done. —SS] [Adjusted —AD]

Reasoning

Definition 84 points out that documentation is the way to improve visibility. It is rephrased because the original one might refer to binary status - "visible" or "invisible". The different reasons for the proposed definition are as follows:

- reason 1
- reason 2
- ...

2.17 Reproducibility (external, product)

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 89.** 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 90.** 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.

Reproducibility

[Needs to be completed —SS] The importance of reproducibility is seen by retractions of influential papers, like the review of hydroxychloroquine trials

Reasoning

[Reproducibility literature review is given in Feinberg et al. [2020]. Includes examples where replicability was found to be lacking. Has a good litereature review in Section 2. —SS]

[Needs to be completed —SS]

2.18 Sustainability (internal, product)

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

▶ **Definition 91.** 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 92.** 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 93.** 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 94.** 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].

Software Sustainability Institute proposal:

▶ **Definition 95.** Capacity of the software to endure

Katz [2016] builds on this definition.

▶ **Definition 96.** 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].

Katz Presentation
Neil's blog
Definition from Neil Chue Hong:

▶ **Definition 97.** 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].

▶ **Definition 98.** Sustainability encompasses cost efficient maintainability and evolvability [Sehestedt et al., 2014].

Sehestedt et al. [2014] goes on to say that sustainability can be observed by evaluating the four criteria of the architectural model: completeness, consistency, correctness and clarity.

Definitions for sustainability are often built by combining other definitions. Willenbring.pdf lists sustainability factors: extensible, interoperable, maintainable, portable, reusable, scalable and usable. [Sounds like they are listing almost all software qualities. It seems that sustainability is at least in part achieved by having high quality software. —SS]

From Rosado de Souza, et al. there are two categories of software sustainability:

Intrinsic Pertaining to characteristics of the software
Extrinsic Pertaining to the software development environment

Find paper that combines nonfunctional qualities into sustainability.
Sounds like definition of maintainability.
Paper critical of a lack of a definition [Venters et al., 2014].
Sustainability depends on the software artifacts AND the software team AND the development process.

Sustainability

[Needs to be completed —SS]

Reasoning

[Needs to be completed —SS]. The different reasons for the proposed definition are as follows:

- reason 1
- reason 2
- ...

3 Direct Qualities

To achieve the qualities listed in Section 2, the internal products and process should achieve the qualities listed in this section.

3.1 Completeness

▶ **Definition 99.** Those attributes of the software that provide full implementation of the functions required. [McCall et al., 1977]. [Ao, if this is from McCall, we should just cite the primary source and remove the Van Vliet reference. —SS] [McCall's definition verbatim: Those attributes of the software that provide full implementation of the functions required. —PM] [Originally added by someone else so I didn't have access to the book. But now it's available and I double checked, Peter got it accurately. —AD]

▶ **Definition 100.** 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.

▶ **Definition 101.** An SRS is complete if, and only if, it includes the following elements:

- All significant requirements, whether relating to functionality, performance, design constraints, attributes, or external interfaces. In particular any external requirements imposed by a system specification should be acknowledged and treated.
- Definition of the responses of the software to all realizable classes of input data in all realizable classes of situations. Note that it is important to specify the responses to both valid and invalid input values.
- Full labels and references to all figures, tables, and diagrams in the SRS and definition of all terms and units of measure [IEEE, 1998].

▶ **Definition 102.** The quality of being whole or perfect and having nothing missing. [Cambridge Dictionary, 2019a].

[Were there any other definitions of completeness? You could add the definition of completeness from IEEE [1998, p. 5–6]. This definition is for requirements, but maybe there is something we can generalize from the definition? We could also look for definitions outside of software development. —SS] [added IEEE and dictionary —AD] [`https://annals-csis.org/proceedings/2016/pliks/468.pdf` has a method of measuring completeness and consistency, but I haven't summarized any def from it yet —AD]

Completeness

The first sentence of Definition 100: A specification is complete to the extent that all of its parts are present and each part is fully developed.

Reasoning

It is concise and measurable. The different reasons for the proposed definition are as follows:

- reason 1
- reason 2
- ...

3.2 Consistency

▶ **Definition 103.** Those attributes of the software that provide uniform design and implementation techniques and notation [McCall et al., 1977]. [Ao, if this is from McCall, we should just cite the primary source and remove the Van Vliet reference. —SS] [McCall's definition verbatim: Those attributes of the software that provide uniform design and implementation techniques and notation. —PM][Originally added by someone else so I didn't have access to the book. But now it's available and I double checked, Peter got it accurately. —AD]

▶ **Definition 104.** 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 105.** Consistency refers to internal consistency. If an SRS does not agree with some higher-level document, such as a system requirements specification, then it is not correct [IEEE, 1998].

▶ **Definition 106.** An SRS is internally consistent if, and only if, no subset of individual requirements described in it conflict. The three types of likely conflicts in an SRS are as follows [IEEE, 1998]:

a) The specified characteristics of real-world objects may conflict. For example,
  1) The format of an output report may be described in one requirement as tabular but in another as textual.
  2) One requirement may state that all lights shall be green while another may state that all lights shall be blue.

b) There may be logical or temporal conflict between two specified actions. For example,
  1) One requirement may specify that the program will add two inputs and another may specify that the program will multiply them.
  2) One requirement may state that "A" must always follow "B," while another may require that "A and B" occur simultaneously.

c) Two or more requirements may describe the same real-world object but use different terms for that object. For example, a program's request for a user input may be called a "prompt" in one requirement and a "cue" in another. The use of standard terminology and definitions promotes consistency.

▶ **Definition 107.** 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 108.** 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 [ISO/IEC/IEEE, 2010].

▶ **Definition 109.** The state or condition of always happening or behaving in the same way [Cambridge Dictionary, 2019b].

[The definition from IEEE [1998] might again be useful. We could also look for definitions outside of software development. Even a dictionary definition could be helpful. —SS] [added IEEE and dictionary —AD]

Consistency

The first sentence of Definition 104: A specification is consistent to the extent that its provisions do not conflict with each other or with governing specifications and objectives.

Reasoning

It is concise and measurable. The different reasons for the proposed definition are as follows:

- reason 1
- reason 2
- ...

3.3 Modifiability

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

▶ **Definition 110.** 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?)

Modifiability

The first sentence of Definition 104: A specification is consistent to the extent that its provisions do not conflict with each other or with governing specifications and objectives.

Reasoning

[Needs to be completed —SS] The different reasons for the proposed definition are as follows:

- reason 1
- reason 2
- ...

3.4 Traceability

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 111.** 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 flow-down 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.

▶ **Definition 112.** The ability to link software components to requirements [McCall et al., 1977]. (As summarized in van Vliet [2000].)

[Once we have verified McCall, we can remove the Van Vliet reference. —SS] [McCall's definition verbatim: Those attributes of the software that provide a thread from the requirements to the implementation with respect to the specific development and operational environment. —PM]

Traceability

[Needs to be completed. —SS]

Reasoning

[Needs to be completed —SS] The different reasons for the proposed definition are as follows:

- reason 1
- reason 2
- ...

3.5 Unambiguity

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].

Unambiguity

[Needs to be completed. —SS]

Reasoning

[Needs to be completed —SS]

3.6 Abstract

▶ **Definition 113.** 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 114.** "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 115.** "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 116.** "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]

Abstract

[Needs to be completed —SS]

Reasoning

[Needs to be completed —SS]

References

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.

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.

Barry W Boehm, John R Brown, and Mlity Lipow. Quantitative evaluation of software quality. In *Proceedings of the 2nd international conference on Software engineering*, pages 592–605. IEEE Computer Society Press, 1976.

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

Cambridge Dictionary. Meaning of completeness in english. https://dictionary.cambridge.org/us/dictionary/english/completeness, 2019a. [Online; accessed 3-December-2019].

Cambridge Dictionary. Meaning of consistency in english. https://dictionary.cambridge.org/us/dictionary/english/consistency, 2019b. [Online; accessed 3-December-2019].

Cambridge Dictionary. Meaning of visibility in english. https://dictionary.cambridge.org/us/dictionary/english/visibility, 2019c. [Online; accessed 3-December-2019].

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.

Melanie Feinberg, Will Sutherland, Sarah Beth Nelson, Mohammad Hossein Jarrahi, and Arcot Rajasekar. The new reality of reproducibility: The role of data work in scientific research. *Proc. ACM Hum.-Comput. Interact.*, 4(CSCW1), May 2020. doi: 10.1145/3392840. URL https://doi.org/10.1145/3392840.

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.

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.

GSA. Visibility & status in an agile environment. https://tech.gsa.gov/guides/visibility_and_status/, 2019. [Online; accessed 1-December-2019].

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. 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. Ieee standard glossary of software engineering terminology. Standard, IEEE, 1991b.

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.

ISO/IEC. *ISO/IEC 9126. Software engineering – Product quality*. ISO/IEC, 2001.

ISO/IEC. Standard 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. Standard, International Organization for Standardization, Mar 2011.

ISO/IEC/IEEE. Systems and software engineering - vocabulary. Standard, International Organization for Standardization, Dec 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.

Jörg Lenhard, Simon Harrer, and Guido Wirtz. Measuring the installability of service orchestrations using the square method. In *2013 IEEE 6th International Conference on Service-Oriented Computing and Applications*, pages 118–125. IEEE, 2013.

A. MacCormack, C. F. Kemerer, M. Cusumano, and B. Crandall. Trade-offs between productivity and quality in selecting software development practices. *IEEE Software*, 20(5):78–85, Sep. 2003. ISSN 1937-4194. doi: 10.1109/MS.2003.1231158. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.61.1633.

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

Steve McConnell. *Code complete*. Pearson Education, 2004.

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.

JD Musa, Anthony Iannino, and Kazuhira Okumoto. Software reliability: prediction and application, 1987.

Jakob Nielsen. Usability 101: Introduction to usability, Jan 2012. URL https://www.nngroup.com/articles/usability-101-introduction-to-usability/.

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.

PMI. *A guide to the project management body of knowledge (PMBOK guide)*. Project Management Inst, 2017.

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

PRIME. Process transparency – invaluable visibility. https://www.primebpm.com/process-transparency/, 2019. [Online; accessed 1-December-2019].

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

Margaret Rouse and Anne Stuart. Business process visibility. https://searchcio.techtarget.com/definition/business-process-visibility, 2013. [Online; accessed 1-December-2019].

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.

Stephan Sehestedt, Chih-Hong Cheng, and Eric Bouwers. Towards quantitative metrics for architecture models. In *Proceedings of the WICSA 2014 Companion Volume*, WICSA '14 Companion, pages 5:1–5:4, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2523-3. doi: 10.1145/2578128.2578226. URL http://doi.acm.org/10.1145/2578128.2578226.

Inderpal Singh. Different software quality model. *International Journal on Recent and Innovation Trands in Computing and Communication*, 1(5):438–442, 2013.

W Spencer Smith, D Adam Lazzarato, and Jacques Carette. State of the practice for mesh generation and mesh processing software. *Advances in Engineering Software*, 100:53–71, 2016.

W. Spencer Smith, Adam Lazzarato, and Jacques Carette. State of the practice for GIS software. https://arxiv.org/abs/1802.03422, February 2018a.

W. Spencer Smith, Zheng Zeng, and Jacques Carette. Seismology software: State of the practice. *Journal of Seismology*, 22(3):755–788, May 2018b.

Ian Sommerville. *Software Engineering 9*. Pearson Education, 2011.

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.

Hans Van Vliet, Hans Van Vliet, and JC Van Vliet. *Software engineering: principles and practice*, volume 13. Citeseer, 2008.

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.

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.