INTRODUCTION TO ISO/IEC 9126-1:2001

BY PANKAJ KAMTHAN

1. INTRODUCTION

Quality is never an accident; it is always the result of high intention, sincere effort, intelligent direction, and skillful execution; it represents the wise choice of many alternatives.
— William A. Foster

Definition [Software Product Quality] [ISO/IEC/IEEE, 2010]. [The] capability of a software product to satisfy **stated and implied** needs when used under specified conditions.

The ISO/IEC 9126-1 [ISO/IEC, 2001] is an International Standard from the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) that provides a quality model for a general software system. It is a **predecessor** of the ISO/IEC 25010 [ISO/IEC, 2011].

This document provides an **exploration, examination, and extension** of the ISO/IEC 9126-1 Standard. In doing so, examples are included for the purpose of annotation, clarification, and illustration.

CONVENTION/NOTATION

In this document, the terms **'attribute'** and **'characteristic'**, and the terms **'ISO/IEC 9126-1'** and **'ISO/IEC 9126-1 Standard'** are considered synonymous, and are used interchangeably.

2. UNDERSTANDING ISO/IEC 9126-1

The ISO/IEC 9126-1 is based on the separation of **different, but related, views of quality**:

- It separates quality into **process quality** and **product quality**.

- It separates product quality into **internal quality, external quality, and quality in use**.

    1. **Internal Quality.** This describes quality characteristics from the perspective of software engineer. It is relevant before delivery of product. It can be checked by verification.

    2. **External Quality.** This describes quality characteristics from the perspective of the customer. It is relevant after delivery of product. It can be checked by validation.

    3. **Quality in Use.** This describes quality characteristics from the perspective of a user. It is relevant after use of product. It can be checked by feedback.

REMARKS

The assumption underlying different, related, views in ISO/IEC 9126-1 is the following: having high quality processes helps creating artifacts with high internal quality; creating artifacts with high internal quality supports achieving a final product with high external quality; and a product of high external quality leads to a high degree of user experience with the product, that is, high quality in use.

2.1. THE RELATIONSHIP BETWEEN PROCESS QUALITY AND PRODUCT QUALITY

No problem can be solved from the same level of consciousness that created it.
— Albert Einstein

The ISO/IEC 9126-1 is based on the understanding that **process quality** contributes to improving **product quality**, and **product quality** contributes to improving **quality in use**, and **conversely**.

Therefore, improving a process is a means to improve product quality, and improving a product is a means to improve quality in use. Furthermore, evaluating quality in use (say, via user feedback) can lead to improving a product, and evaluating a product can lead to improving a process.

This **dynamic symbiosis** is illustrated in Figure 1.

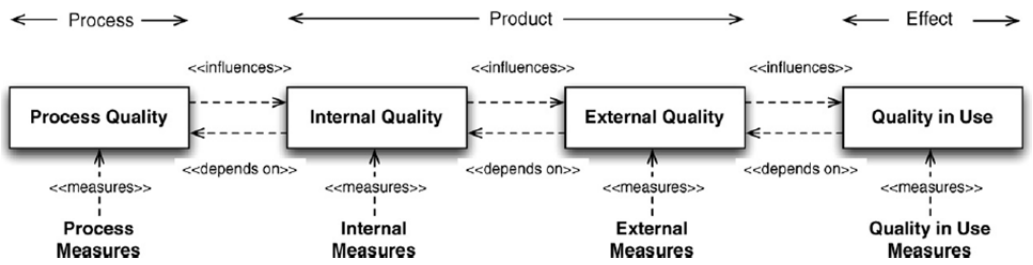


Figure 1. The bidirectional relationships between process quality and product quality. (Source: [Hansen, Jonasson, Neukirchen, 2011].)

3. ORGANIZATION OF QUALITY ATTRIBUTES

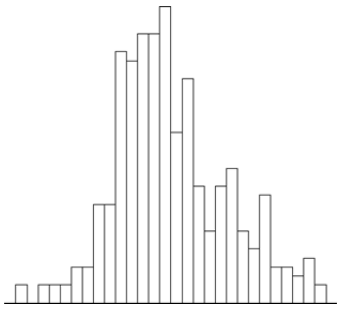
The ISO/IEC 9126-1 product quality model consists of a number of **hierarchically structured quality characteristics** and **sub-characteristics**, as shown in Table 1.

Quality					
Functionality	**Reliability**	**Usability**	**Efficiency**	**Maintainability**	**Portability**
Suitability Accuracy Interoperability Security Functionality Compliance	Maturity Fault-Tolerance Recoverability Reliability Compliance	Understandability Learnability Operability Attractiveness Usability Compliance	Time-Behavior Resource-Utilization Efficiency Compliance	Analyzability Changeability Stability Testability Maintainability Compliance	Adaptability Installability Co-Existence Replaceability Portability Compliance

Table 1. The ISO/IEC 9126-1 product quality model.

REMARKS

- It is evident that **not** all quality attributes are important or equally important for **all** software systems. (For a given software system, it could be imagined that the list of quality attributes reside on a pitch histogram.) For example, **portability** may not be important for a software system aimed for a specific platform only, and **maintainability** may not be important for a software system aimed for a one time use or for use for a short time period. Furthermore, **efficiency** may be desirable, but the cost of improving efficiency may not be feasible, or **efficiency** may not be desirable beyond human capability to process information.



(Source: Google Images.)

- The following quality attributes are considered especially important for **software architecture**: **Interoperability, Changeability (Modifiability), Reliability (Availability), Security, Testability, and Usability** [Bass, Clements, Kazman, 2003, Chapter 4; Gorton, 2011, Chapter 3; Critchley, 2015; Sangwan, 2015, Chapter 4]. Indeed, the maturation of software architecture as a discipline is related intimately to dealing with relevant quality attributes [Shaw, Garlan, 1996].

4. NATURE OF QUALITY ATTRIBUTES

4.1. ON "ABILITY"

It can be noted that many quality attributes end with **"ility"**, or more specifically with "ability". The following has been pointed out in [Alley, 1996, Appendix B]:

> *-ability words:* A word ending in "ability" is a signal that you can tighten the sentence by having the word "can" precede the verb buried in the "ability" word. Although examples such as "capability" are not offensive, pretentious constructions such as "operationability" and "manufacturability" are. Rewrite the sentence using "can operate" and "can manufacture."

4.2. SOFTWARE SYSTEM AS A SOCIO-TECHNICAL SYSTEM: QUALITY ATTRIBUTES ARE EMERGENT PROPERTIES, NOT FEATURES

In theory, a software system can be viewed as a technical system, isolated from its environment. This, for example, is the case with **S-Type Systems** [Madhavji, Fernández-Ramil, Perry, 2006, Chapter 1].

However, in practice, a software system should be viewed a **socio-technical system**, not isolated from its environment consisting of, among other things, people, processes, and other systems. This, for example, is the case with **E-Type Systems** [Madhavji, Fernández-Ramil, Perry, 2006, Chapter 1]. The socio-technical systems are characterized **holistically** by **emergent properties** [Leveson, 2011; Sommerville, 2011, Chapter 10; Penzenstadler, Raturi, Richardson, Tomlinson, 2014].

There are number of definitions of emergent property[1]. An emergent property 'emerges' only when a software system is **placed in and interacts with its environment**. (For example, security is an emergent property. A software system **does not become unsecure in and of itself**, but it can become unsecure once it is in its environment.)

An emergent property is a **property of the system as a whole** rather than a property of any of its parts. It is therefore an **irreducible property**. For example, a software system may not be reliable, even if all parts of that software system are reliable.

There are certain properties which are **not emergent**. For example, the size of a software system is **not** an emergent property [Fernandes, Machado, 2016, Section 3.3].

An emergent property is a **consequence of the relationships between parts of a system**. It can therefore only be assessed and measured once the parts have been integrated into a system.

In practice, it is usually **impossible** to ascertain **all possible relationships between parts of a socio-technical system**. This is one of the reasons why it is usually **difficult to specify software systems completely** or to **determine the root causes of software-related failures with absolute certainty**.

An emergent property could be **functional or non-functional** [Sommerville, 2011, Chapter 1].

A **functional property** emerges (as a **structure of the system**) after the parts of the system have been integrated into a whole. A **non-functional property** emerges (as a behavior of the system) after the system is placed in its **operational environment**.

[1] An **emergent property** is a characteristic that is exhibited only by the system in use, as opposed to an inherent characteristic already exhibited in the system's static existence [Penzenstadler, Raturi, Richardson, Tomlinson, 2014].

EXAMPLE

Let there be a bicycle, an abstraction of which is shown in Figure 2.

Figure 2. An icon for a bicycle. (Source: Google Images.)

For example, **"being a vehicle"** is a **functional property** of a bicycle, once it has been assembled from its parts.

For example, **"being able to ride"** is a **non-functional property** of a bicycle.

It is important to note that the mere fact that a bicycle has all its parts assembled correctly is **insufficient** to be able to ride a bicycle from point A to point B successfully. (The rider may not be concentrating and, as a result, may lose balance and fall off the bicycle; the road may not be smooth and, as a result, the chain may get dislodged; the wind may be too strong for the rider to be able to maintain balance; and so on.)

EXAMPLE

In general, the **features** of a software system are **functional properties**. In general, the **quality attributes** of a software system are **non-functional properties**.

For example, consider a word processor with a graphical user interface. It is usually possible to use a pull-down menu to check the types of fonts that are supported by the word processor; it is, however, **not** possible to check whether the word processor is reliable or usable.

4.3. QUALITY ATTRIBUTES ARE FORETHOUGHT, NOT AFTERTHOUGHT, ("BUILT-IN, NOT BOLTED-ON")

The quality attributes of a software system are not features that can be tacked on after implementation if such a need arises during delivery or if a problem occurs during operation.

For example, security cannot be **“bolted on”** or **“patched in”** after an attack has occurred [Allen, Barnum, Ellison, McGraw, Mead, 2008, Section 3.2.1]. They need to be forethought and built-in from the beginning (at the requirements phase).

5. FUNCTIONALITY

Definition [Functionality] [ISO/IEC, 2001]. The capability of the software product to provide functions that meet stated and implied needs when the software is used under specified conditions.

For example, the necessary functions may be stated in a software requirements specification (SRS).

Functionality has been decomposed into the following quality attributes:

- Suitability
- Accuracy
- Interoperability
- Security

5.1. SUITABILITY

Definition [Suitability] [ISO/IEC, 2001]. The capability of the software product to provide an appropriate set of functions for specified tasks and user objectives.

In the definition, a suitable 'set of functions for specified tasks and user objectives' can be obtained by a proper requirements elicitation process.

For example, by an appropriate elicitation of use cases that are aligned with users' goals, and their implementation, a software system can provide most 'set of functions for specified tasks and user objectives'.

5.2. ACCURACY

Definition [Accuracy] [ISO/IEC, 2001]. The capability of the software product to provide the right, or agreed results, or effects with the needed degree of precision.

The accuracy of a software system can be assured by an appropriate combination of **debugging, inspections, and testing**.

The previous definition of accuracy does **not** distinguish between accuracy and precision. In fact, it motivates the need to understand the **difference** between accuracy and precision.

5.2.1. ACCURACY VERSUS PRECISION

The terms accuracy and precision are two **different** concepts, and the difference is considered relevant to software engineering [Fenton, Pfleeger, 1997; IEEE, 2014].

MODEL 1

The "accuracy indicates **proximity** of measurement results to the true value; precision to the **repeatability** or reproducibility of the measurement" [Wikipedia].

In various technical fields, accuracy refers to the **closeness** of a given measurement to its true value; precision refers to the **stability** of that measurement when repeated many times [Wikipedia].

EXAMPLE

It is possible to use a **shooting target metaphor** to illustrate the difference between accuracy and precision. This is illustrated in Figure 3.

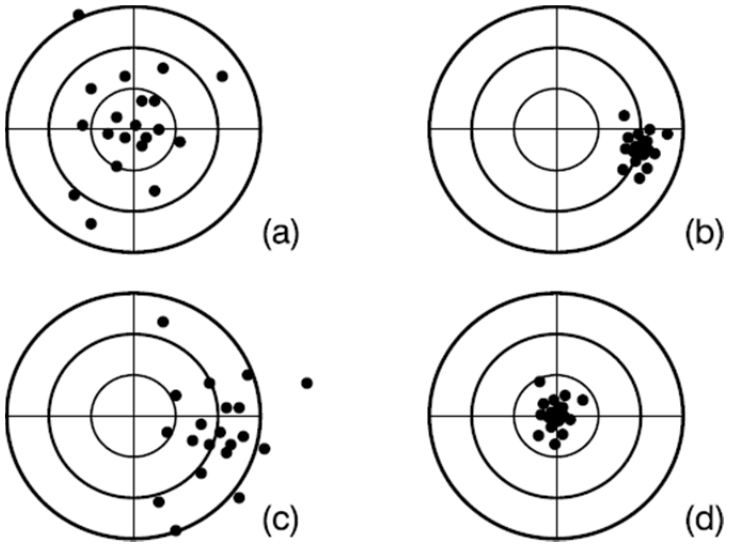


Figure 3. (a) low precision, high accuracy; (b) high precision, low accuracy; (c) low precision, low accuracy; (d) high precision, high accuracy. (Source: [Drosg, 2009, Section 7.6].)

MODEL 2

The difference between accuracy and precision can also be highlighted **numerically** [Fenton, Pfleeger, 1997; Anastassiou, Mezei, 2015, Section 1.4].

Definition [Significant Digits]. The number of digits, starting with the leftmost nonzero digit, and ending with the rightmost (zero or nonzero) correct digit, is called the number of significant digits.

The **accuracy** of a number *x* is given by the **number of significant decimal (or other) digits** to the right of the decimal point in *x*.

The **precision** of a number *x* is given by the **total number of significant decimal (or other) digits**.

Accurate to n Decimal Places: This means the n digits to the right of the decimal point are meaningful (that is, the digits can be trust to be exact).

Accurate to n Significant Digits: This means that n digits, starting with leftmost nonzero digit, are meaningful (that is, the digits can be trust to be exact).

EXAMPLE

Let a millimeter ruler be used to measure some lengths, as shown in Figure 4. This means the measurements using the ruler are **accurate to 1mm**. If the lengths are given in meters, then the measurements are accurate to **three decimal places**, since 1mm equals 0.001m.

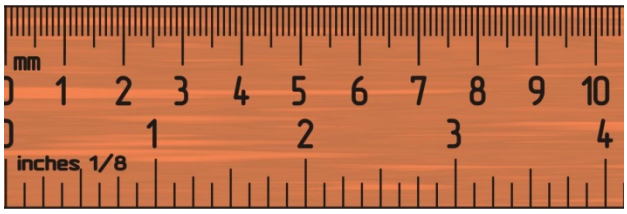


Figure 4. A part of a physical millimeter ruler. (Source: Google Images.)

A measurement such as 173.1234567m would be as accurate as 173.123m since the ruler is accurate to 1mm only. In this example, the measurement of 173.123m is accurate to three decimal places, and is accurate to six significant digits. The measurement 0.023m is accurate to three decimal places, and is accurate to two significant digits.

5.3. INTEROPERABILITY

Definition [Interoperability] [ISO/IEC, 2001]. The capability of the software product to interact with one or more specified systems.

It is evident that interoperability of a software system is a property that **cannot** be discussed in **isolation**, but only by taking into account the software system and its **environment**, that is, the **ecosystem** in which the software system exists. In this ecosystem, only other software systems, and especially **operating system utilities and application software systems**, are of concern.

For example, **Microsoft Office** is an example of a suite in which the individual software systems are highly interoperable with each other.

REMARKS

It is expected that the use of **standards** contributes to interoperability [DeNardis, 2011].

5.4. SECURITY

Definition [Threat] [ISO/IEC/IEEE, 2010]. A negative set of circumstances, a negative set of events, or a risk that will have a negative impact on a project objective if it occurs.

REMARKS

- It follows from the aforementioned definition that a threat is a **possibility, not a surety**.

- There can be different kinds of threats [Ashbaugh, 2009, Section 4.2].

- There are different ways of identifiying threats, including **STRIDE** [Shostack, 2014, Chapter 3], **attack trees** [Shostack, 2014, Chapter 4], and **attack libraries** [Shostack, 2014, Chapter 5]. The threats could be **intentional or unintentional**, and **malicious or nonmalicious**.

- The **Web Application Security Consortium** provides a **classification of threats**[2] to Web Applications.

[2] URL: http://projects.webappsec.org/Threat-Classification .

Definition [Security] [ISO/IEC, 2001]. The capability of the software product to **protect** information so that unauthorized persons or systems cannot read or modify it, and authorized persons or systems are not denied access to it.

REMARKS

- It is evident that security of a software system is a property that **cannot be discussed in isolation**, but only after taking into account the software system and its environment, that is, the **ecosystem** in which the software system exists.

- There can be two opposing directions of approaching security: **positive and negative**. The ISO/IEC 9126-1 approaches security from a positive viewpoint ("about what to permit"), while the ISO/IEC 25010 approaches security from a negative viewpoint ("about what to forbid").

Definition [Asset] [Refsdal, Solhaug, Stølen, 2015]. Anything of value to a party (person, group, or organization).

There are a number of different types of assets that may typically be found in a software project [Ashbaugh, 2009, Section 3.2] carried out by an organization. These types of assets include, but are not limited to, the following:

- Information Assets
- Business Rules
- Services or Functions
- Software
- Proprietary Formulas
- Encryption Methods and Keys
- Databases
- People, or Specifically the Knowledge or Skillset Possessed by Individuals
- Accounts and the Funds Associated with those Accounts
- Transactions

The purpose of security is **protection of assets against threats**.

5.4.1. A MODEL FOR SECURITY

As a quality attribute, security is defined at a rather high-level and can be decomposed into other, low-level, quality attributes, for a better understanding [Brost, Hoffmann, 2015; Felderer, Büchler, Johns, Brucker, Breu, Pretschner, 2016].

A collection of these low-level quality attributes is:

- **Confidentiality:** The assurance that access controls are enforced so that certain data is not disclosed to unauthorized entities. (The "entities" could be an individual, group, or organization.)

- **Integrity:** The assurance that data is protected from unauthorized modification or deletion.

- **Availability:** The assurance that data and operation of the system are available without interruption during the times they are needed. (It could be noted that availability is related to and relevant for reliability.)

- **Authenticity:** The assurance that data is authentic and coming from an entity that is guaranteed to be the one it claims to be. (For example, in an anonymous or secret data transmission, authenticity cannot be guaranteed.)

- **Nonrepudiability:** The assurance that an entity will fulfill its obligations to a contract. (It could be noted that nonrepudiation is related to and relevant for legality.)

  [Let there be three people, A, B, and C.]
  "Nonrepudiation is about A showing to B a proof that some data really comes from A, such that not only B is convinced, but B also gets the assurance that he could show the same proof to C, and C would be convinced, too, even if C does not trust B."

  — Post on Information Security Stack Exchange.

REMARKS

It could be noted that authenticity and nonrepudiability are related but different concepts. This is highlighted in Table 2.

Authenticity	Nonrepudiability
A technical concept.	A non-technical (legal) concept.
The approaches to provide authenticity include the use of a Message Authentication Code.	The approaches to provide authenticity include the use of a Digital Signature.

Table 2. A comparison between authenticity and nonrepudiability.

5.4.2. A SECURITY ANALOGY: MILITARY BASE AND SOFTWARE SYSTEM

There is a strong **analogy** between **security of a military base** and the **security of a software system**.

The means to achieve protection is to create a **'boundary'** around a software system. This implies that certain external entities do, and the others do not, have access to the software system.

The notion of a **boundary and access** are intimately related. The notion of a boundary around a software system is **recursive**. For example, other boundaries (for different sub-systems or services) can be created. The notion of access to the software system is **conditional**. For example, access to one or more boundaries depends on the conditions that an external entity can satisfy.

5.4.3. SECURITY ENGINEERING

It could be said that security has become a **ubiquitous concern** in almost any computer use [Lincke, 2015, Section 1.1].

For example, most current computer systems, distributed either with **antivirus software** (such as desktop or laptop computers running Microsoft Windows) or with **integrated system security software** (such as Apple Tablets running iOS), are **over-protective by design**.

Indeed, the significance of security in software systems has spawned the discipline of **security engineering** [**Howard, Lipner, 2006**; **Schumacher, Fernandez-Buglioni, Hybertson, Buschmann, Sommerlad, 2006**; Allen, Barnum, Ellison, McGraw, Mead, 2008; Anderson, 2008; Brost, Hoffmann, 2015; Felderer, Büchler, Johns, Brucker, Breu, Pretschner, 2016]. Figure 5 shows interrelated elements of a security engineering framework.

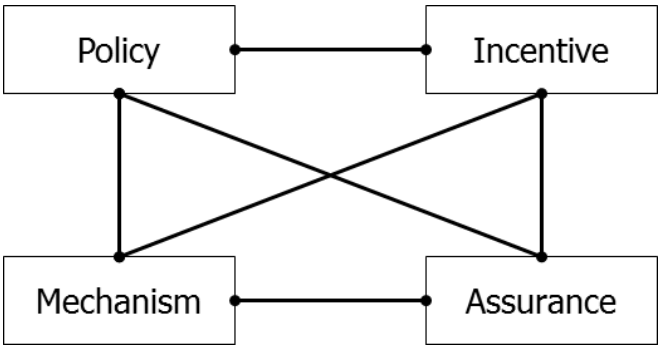


Figure 5. A security engineering framework [Anderson, 2008, Section 1.2].

- **Policy:** This entails goals (say, of an organization, for itself, for its customers, and for its products and services) for addressing security.

- **Incentive:** This entails motives of those who aim to breach a security policy, and the commitment of those who aim to prevent it.

- **Mechanism:** This entails technical aspects, such as algorithms, ciphers, hardware and software infrastructure, and strategies for addressing security.

- **Assurance:** This entails the amount of reliance that can be placed on a mechanism for addressing security.

5.4.4. STRATEGIES FOR (SOFTWARE) SECURITY

The means to achieve security are manifold and depend on a number of factors, including the budget for the software project, type of software system, the nature and degree of interaction with the software system, and so on [Beckers, 2015].

There are two **strategies for (software) security**, at two **extremes** of a spectrum:

1. **Default Permit.** "Everything, Not Explicitly Forbidden, Is Permitted."
   This approach sacrifices security at a cost of certain other quality attributes, and is recommended only if there is a need for very low security.

2. **Default Forbid.** "Everything, Not Explicitly Permitted, Is Forbidden."
   This approach improves security at a cost of certain other quality attributes, and is recommended only if there is a need for very high security.

REMARKS

It can be noted that, even though there are overlaps, **security, safety, and privacy** are **not pairwise synonymous** [Stalla-Bourdillon, Phillips, Ryan, 2014; Refsdal, Solhaug, Stølen, 2015, Section 4.4].

For example, **banks** are primarily concerned about **security and privacy (and not safety)**, **airports** are primarily concerned about **safety (and not security or privacy)**, and **hospitals** are primarily concerned about **safety and privacy (and not security)**. (There are exceptions, as usual.)

Therefore, each of these quality attributes should be addressed **separately**.

6. RELIABILITY

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

The following pair of definitions is relatively more insightful:

Definition [Reliability] [ISO/IEC/IEEE, 2010].
(1) The ability of a system or component to perform its required functions under stated conditions for a specified period of time.
(2) The **probability** that software will not cause the failure of a system for a specified time under specified conditions.
Note: The probability is a function of the **inputs to** and **use of** the system as well as a function of the **existence of faults** in the software. The inputs to the system determine whether existing faults, if any, are encountered.

Reliability has been decomposed into the following quality attributes:

- Maturity
- Fault Tolerance
- Recoverability

6.1. MATURITY

Definition [Maturity] [ISO/IEC, 2001]. The capability of the software product to avoid failures as a result of faults in the software.

It follows from the aforementioned definition of maturity that understanding the concepts of **fault and failure** is important. Therefore, these concepts are examined in some detail.

Definition [Fault] [Fenton, Pfleeger, 1997, Page 156]. A fault occurs when a human error results in a mistake in some software product.

Definition [Failure] [ISO/IEC/IEEE, 2010].
(1) [The] termination of the ability of a product to perform a required function or its inability to perform within previously specified limits.
(2) An event in which a system or system component does not perform a required function within specified limits.

It could be noted that a fault is a **necessary, but not a sufficient, condition** for failure. In other words, if there is a failure, then there must be a fault; however, if there is a fault, then it does not mean there will be a failure.

This is illustrated in Figure 6.

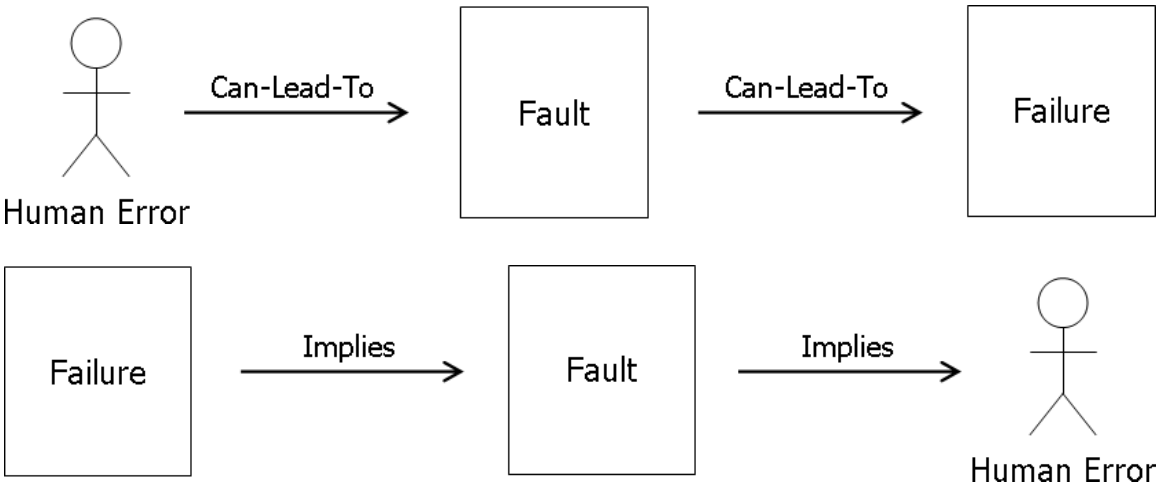


Figure 6. The relationship between error, fault, and failure [Fenton, Pfleeger, 1997, Page 156].

6.2. FAULT TOLERANCE

Definition [Fault Tolerance] [ISO/IEC, 2001]. The capability of the software product to maintain a specified level of performance in case of software faults or of infringement of its specified interface.

The motivation for fault tolerance is **acceptance of unanticipated circumstances** beyond its original specification, and thereby **incorporation of some degree of flexibility** in a software system. Therefore, the potential for (reasonable) **variations** in the environment in which a software system resides must be acknowledged, absorbed, and acted upon.

The (reasonable) variations could, for example, come in form of surge in usage, power outage, and so on.

For example, a server that is designed to handle 100 simultaneous connections, but crashes at the 101st connection, is **not fault tolerant**.

6.2.1. FAULT TOLERANCE VERSUS ERROR TOLERANCE

Definition [Fault Tolerance] [ISO/IEC/IEEE, 2010]. The ability of a system or component to continue normal operation despite the **presence of hardware or software faults**.

Definition [Error Tolerance] [ISO/IEC/IEEE, 2010]. The ability of a system or component to continue normal operation despite the **presence of erroneous inputs**.

REMARKS

The notion of fault tolerance is related to robustness:

Definition [Robustness] [ISO/IEC/IEEE, 2010]. The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions.

6.3. RECOVERABILITY

Definition [Recoverability] [ISO/IEC, 2001]. The capability of the software product to re-establish a specified level of performance and recover the data directly affected in the case of a failure.

For example, **Adobe Acrobat, GNU Emacs, certain HTTP servers, and Microsoft Word** currently have some type of built-in capability for recovering data.

For example, a text processor that does not save user data or user preferences in the event of an **operating system crash** does **not** have appropriate support for recoverability.

6.4. HARDWARE VERSUS SOFTWARE RELIABILITY

There are similarities as well differences between hardware and software reliability. The similarities exist as both are products, and differences exist primarily because hardware is material and software is immaterial.

A **similarity** between hardware and software reliability is that both are **probabilistic**.

However, there also are **differences** between hardware and software reliability because software **does not wear out over time** [Laird, Brennan, 2006, Section 8.7; Yamada, 2014, Page 6; Verma, Ajit, Karanki, 2016, Section 6.4; Yamada, Tamura, 2016, Section 1.2], and **reliability of software cannot be improved by redundancy with identical versions** [Yamada, Tamura, 2016, Section 1.2]. This can affect the **shape** of the bathtub curve[3].

6.5. A MODEL FOR QUANTITATIVE UNDERSTANDING OF RELIABILITY

To have a **mathematical interpretation** of software reliability can be helpful in its verification [Fenton, Pfleeger, 1997; Laird, Brennan, 2006, Section 8.7; Laplante, 2007; Yamada, 2014; Verma, Ajit, Karanki, 2016, Chapter 6].

Definition [Failure Intensity] [Laird, Brennan, 2006]. The number of failures per unit.

The notion of **"unit"** is the natural unit for the system under study.

For telephone networks, failure intensity could be the number of call failures per million of calls handled correctly. For price scanning systems, failure intensity could be the number of incorrect prices per 100 items scanned. For airplanes, failure intensity could be the number of crashes per millions of kilometers flown.

Let f(t) be the **probability distribution function of failures**, F(t) be the **probability of failure** by time t, and R(t) be the **probability of no failure** by time t.

THE PROBABILITY DISTRIBUTION FUNCTION OF FAILURES

There can be different kinds of probability distribution function of failures.

Let the underlying **assumption** be that the failures are apt to occur **randomly** over time, that is, the failures are **independent of the past events**. The probability of failure is equivalent for each time period t, given that the system has not failed before time t.

The **exponential distribution function of failures** given by:

$$f(t) = \lambda e^{-\lambda t},\ t \geq 0.$$

[3] URL: https://users.ece.cmu.edu/~Ekoopman/des_s99/sw_reliability/ .

In definition of f, the **horizontal axis** represents time, and the **vertical axis** represents the expected **failure intensity** at that time. The factor λ is a **system-dependent parameter**.

THE NATURE OF THE PROBABILITY DISTRIBUTION FUNCTION OF FAILURES

The nature of the function f can be explained as follows. The failure intensity is initially **high**, as the software system is new and faults are detected during testing. However, the **failure intensity decreases over time**, as faults (corresponding to the failures) are repaired. This behavior is depicted in Figure 7.

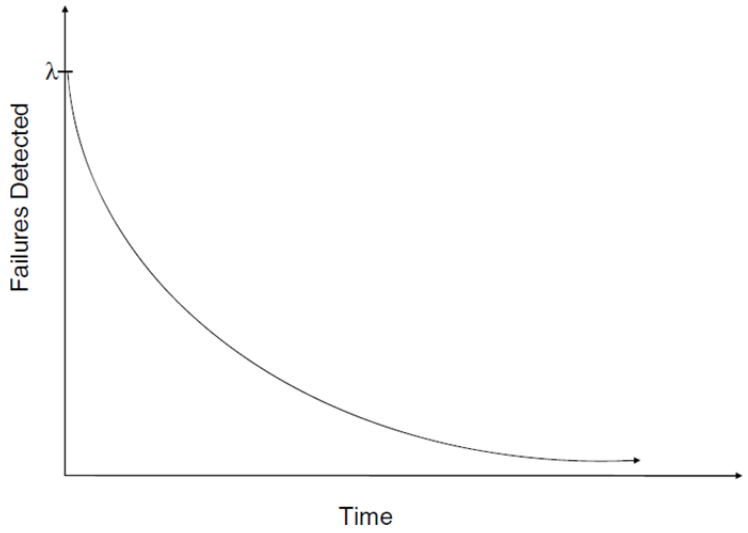


Figure 7. A model of reliability in terms of failures (during development). (Source: [Laplante, 2007].)

However, once a software system is **released, delivered, and operated**, the failure intensity can **increase**, if the system is **not maintained** and starts to **'age' and 'decay'**.

This phenomenon, known as **software aging** [**Parnas, 1994**; Brooks, 1995] and similar to that of **living organisms** and the **products of industrial engineering**, is depicted in Figure 8. The graph of f in Figure 8 is called a **bathtub curve**.

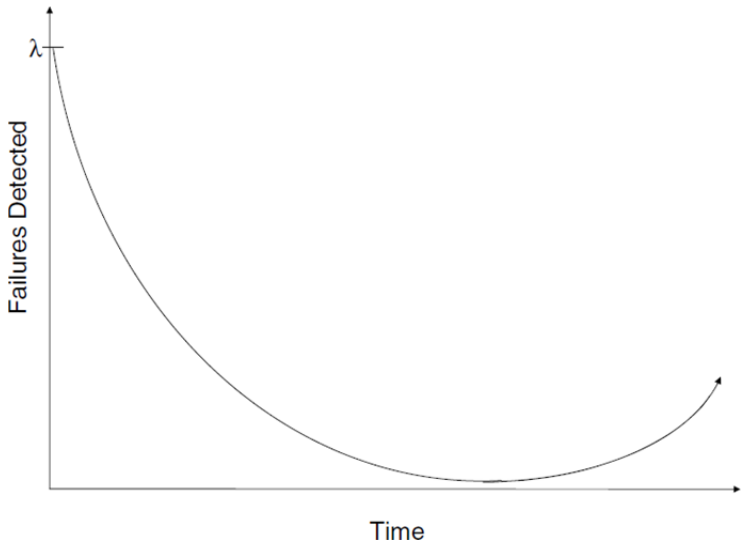


Figure 8. A model of reliability in terms of failures (after development). (Source: [Laplante, 2007].)

THE PROBABILITY OF FAILURE

F(t) is the probability of failure by time t, and is equivalent to the probability of failure during the interval 0 to t.

Let T denote a specific time, such that T > 0.

F(t) for an exponential distribution function is given by:

$$F(t) = \int_0^T \lambda e^{-\lambda t} dt = 1 - e^{-\lambda T}.$$

THE SYSTEM-DEPENDENT PARAMETER

The factor λ is the **reciprocal** of the **mean-time-to-failure (MTTF)**:

$$\lambda = \frac{1}{\text{MTTF}}.$$

If there is failure data with n different failure times t_1 through t_n, then

$$\text{MTTF} = \sum_{i=1}^{n} \frac{t_i}{n}.$$

If there is a probability distribution function f(t), then MTTF is the **expected value** of f(t):

$$\text{MTTF} = \int_0^{\infty} tf(t)dt.$$

In particular, MTTF for an exponential distribution function is given by:

$$\text{MTTF} = \int_0^{\infty} \lambda te^{-\lambda t}dt = \frac{1}{\lambda}.$$

EXAMPLE

Let MTTF be 2 days. Then,

$$\lambda = 1/\text{MTTF} = 0.5.$$

Therefore, for the exponential distribution,

$$f(t) = 0.5e^{-0.5t}.$$

The probability of failure on the 10th day is given by:

$$f(10) = 0.5e^{-5} = 0.0033 = 3.3\%.$$

THE RELIABILITY FUNCTION

The **reliability function** is the probability the system has **not** failed by time t. Therefore, the reliability function equals one minus the probability it **has** failed by time t.

In other words,

$$R(t) = 1 - F(t).$$

R(t) for an exponential distribution function is given by

$$R(t) = 1 - (1 - e^{-\lambda t}) = e^{-\lambda t}.$$

EXAMPLE

Let $\lambda = 0.1$. Then,

$$F(t) = 1 - e^{-0.1t} \text{ and } R(t) = e^{-0.1t}.$$

The probability of failure through day 7 is $1 - e^{-0.7}$.

The probability that the system has not failed through day 7 is $e^{-0.7}$.

7. USABILITY

UNIX is user-friendly; it just chooses its friends.
— Andreas Bogk

Before software can be reusable it first has to be usable.
— Ralph Johnson

Definition [Usability] [ISO/IEC, 2001]. The capability of the software product to be understood, learned, and attractive to the user, when used under specified conditions.

Usability has been decomposed into the following quality attributes:

- Understandability
- Learnability
- Operability
- Attractiveness

7.1. UNDERSTANDABILITY

If you can't explain it simply, you don't understand it well enough.
— Richard P. Feynman

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

For example, presence of a **tutorial** (including representative **examples of use**) and provision for **context-sensitive help** can contribute towards understandability.

7.2. LEARNABILITY

Definition [Learnability] [ISO/IEC, 2001]. The capability of the software product to enable the user to learn its application.

For example, conducting **user testing**, providing **user documentation** describing significant use cases, and having **context-sensitive help**, can each contribute towards learnability of a software system.

7.3. OPERABILITY

Definition [Operability] [ISO/IEC, 2001]. The capability of the software product to enable the user to operate and control it.

For example, a modeler that does not allow repositioning a construct placed on the canvas does not have appropriate support for operability.

In general, any **resistance in a legitimate use** of a software system is **inversely proportional** to the operability of a software system.

For another example, for a given software system, a **command-line interface (CLI)** gives a user **more control** than a **graphical user interface (GUI)**, assuming that each are designed and implemented properly.

7.4. ATTRACTIVENESS

Definition [Attractiveness] [ISO/IEC, 2001]. The capability of the software product to be attractive to the user.

The property of attractiveness is related intrinsically to the **interaction design** of a software system.

From a practical standpoint, attractiveness of a software system can be useful if it **persuades users to use** the system, or **motivates users to continue using the system**, despite its **shortcomings**.

REMARKS

La bellezza è la somma delle parti per cui niente necessita di essere modificato,aggiunto, o rimosso. (Beauty is a summation of the parts where nothing is needed to be altered, added, or taken away.)
— Elio Carletti

It is said that beauty is in the **eyes of the beholder**.

For example, **all** cars made by **Alfa Romeo** are considered to be beautiful:

For example, **some** cars made by **Ferrari** are considered to be beautiful:

For example, **Euler's Identity** has been voted to be **the most beautiful theorem** in mathematics:

$$e^{i\pi} + 1 = 0.$$

7.5. USABILITY VERSUS SECURITY

It has conventionally been the case that initiatives towards security and usability have **not** been aligned with each other, and, indeed, **compete** with each other [Wiegers, 2003]. In particular, usually, a strict security check comes at a price of usability.

For example [De Luca, Lindqvist, 2015]:

> The **challenge** faced by system designers is to make **smartphone authentication both secure and usable**.
>
> [It is possible to make authentication] much more secure than it is now by ignoring user needs, but given that **the average user unlocks his or her phone 50 times per day, authentication must be fast and convenient or most users will disable it**.

However, this **situation is changing** with the realization that, in certain cases, lack of security is, at times, also a manifestation of lack of usability [**Garfinkel, Lipford, 2014**].

For example, **not checking user input for validity** is a security issue (as a user can supply **malicious data** that may corrupt the system), as well as usability issue (as a user may accidentally make a mistake of entering malicious data, of which the user should be informed preventatively, in a timely manner) [Cranor, Garfinkel, 2005].

Indeed, it has been pointed out [Howard, LeBlanc, Viega, 2010]:

> It is essential that the human interface be designed for ease of use, so that users routinely and automatically apply the protection mechanisms correctly. [...]
>
> **Security only works if the secure way also happens to be the easy way.**

8. EFFICIENCY

You know you have achieved perfection in design, not when you have nothing more to add, but when you have nothing more to take away.
— Antoine de Saint Exupéry

[A program] should run **as fast as necessary, but no faster**; something important is always traded away to increase speed.
— Richard E. Pattis

Definition [Efficiency] [ISO/IEC, 2001]. The capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions.

In general, efficiency refers to how **economically** the software utilizes the **given** constraints of the computer (such as available memory, disk space, and processor speed, none of which are unlimited).

Efficiency has been decomposed into the following quality attributes:

- Time-Behavior
- Resource-Utilization

8.1. TIME BEHAVIOR

Definition [Time Behavior] [ISO/IEC, 2001]. The capability of the software product to provide appropriate response and processing times and throughput rates when performing its function, under stated conditions.

The expectations of time behavior are related to several factors, including the **nature of hardware system** (such as, stationary/non-stationary), **nature of the operating system** (such as, sequential/non-sequential), and the **nature of software system** (such as, distributed/non-distributed). In case the **software system is distributed** yet another factor is the **state of the network** (such as, congested/non-congested).

8.2. RESOURCE UTILIZATION

Definition [Resource Utilization] [ISO/IEC, 2001]. The capability of the software product to use appropriate amounts and types of resources when the software performs its function under stated conditions.

It is said (and implied by the following adage) that **no amount of space is sufficient**; given any amount of space, it will be filled in eventually.

Corollary to Parkinson's Law: "… data expands to fill the space available for storage".

9. MAINTAINABILITY

Definition [Maintainability] [ISO/IEC, 2001]. The capability of the software product to be modified.

The modifications include **corrections, improvements, or adaptations** of software to (1) changes in environment, or (2) changes in requirements specifications or design descriptions.

Maintainability has been decomposed into the following quality attributes:

- Analyzability
- Changeability
- Stability
- Testability

9.1. ANALYZABILITY

Definition [Analyzability] [ISO/IEC, 2001]. The capability of the software product to be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified to be identified.

It is evident that a software system can be effectively analyzed if it, and its associated products, are **adequately structured** and are **adequately perceivable** (for example, readable).

9.2. CHANGEABILITY

Definition [Changeability] [ISO/IEC, 2001]. The capability of the software product to enable a specified modification to be implemented.
NOTE 1 Implementation includes coding, designing and documenting changes.
NOTE 2 If the software is to be modified by the **end user**, changeability may affect **operability**.

It is evident that a software system can be changed effectively if it, and its associated products, are **adequately structured** and is **adequately perceivable** (for example, readable and understandable).

It is **not useful** to simply claim that a system is changeable, as any system can be changed (1) if there is sufficient time, effort, and resources, and (2) if the side-effects of the change are ignored.

The changeability depends on the stakeholder involved in making the change. In **end-user software engineering**, the users are themselves involved in the development of the software system. For example, Wikipedia is a result of end-user development.

In a **Social Web** environment, a professional consumer can become a co-producer or (as a portmanteau) a **'prosumer'**. Therefore, any changes to a software system that are predisposed to a specific prosumer may impact other consumers.

9.3. STABILITY

Definition [Stability] [ISO/IEC, 2001]. The capability of the software product to avoid unexpected results from modifications of the software.

It is evident that stability of a software system is related to minimizing the **propagation** of changes. In case of changes made during testing, this is related to **regression testing**.

9.4. TESTABILITY

Definition [Testability] [ISO/IEC, 2001]. The capability of the software product to enable modified software to be validated.

The **Test-Driven Development (TDD)** [Beck, 2003], a test-first approach to programming, is a practice intrinsic to **Extreme Programming (XP)** [Beck, Andres, 2005]. TDD supports testability of a software system in a number of ways, including (1) by **writing tests early** in the software process, and (2) by **determining the progress** of the software project based on the tests that pass.

10. PORTABILITY

Definition [Portability] [ISO/IEC, 2001]. The capability of the software product to be transferred from one environment to another.

Portability has been decomposed into the following quality attributes:

- Adaptability
- Installability
- Co-Existence
- Replaceability

10.1. ADAPTABILITY

Definition [Adaptability] [ISO/IEC, 2001]. The capability of the software product to be adapted for different specified environments without applying actions or means other than those provided for this purpose for the software considered.

Let the stakeholders of a software system be classified into the categories of **producers and consumers**.

From a **producer's viewpoint**, a commitment to a **closed environment** is potentially **detrimental to broad adaptability** of a software system. From a **consumer's viewpoint**, a **capability to personalize** is **favorable to adaptability** of a software system.

10.1.1. SCALABILITY

In ISO/IEC 9126-1, scalability is **subsumed** by adaptability.

Definition [Scalability] [Menascé, Almeida, Dowdy, 2004]. A software system is said to be scalable if its performance does not degrade significantly as the number of users, or equivalently, the load on the system increases.

For example, Figure 9 illustrates a **scalable and non-scalable** system. The response time of **System A** increases **nonlinearly** with the load. The response time of **System B** exhibits a much more controlled (linear) growth.

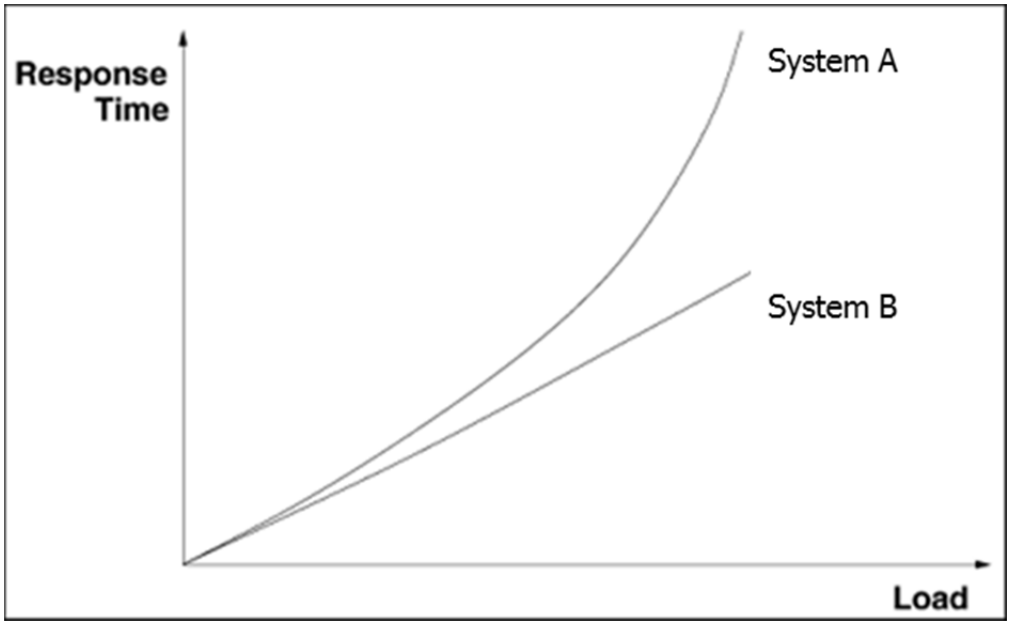


Figure 9. The System A is **not scalable**; the System B is scalable. (Source: [Menascé, Almeida, Dowdy, 2004].)

REMARKS

The need for scalability is especially relevant to **distributed software systems**, such as **Web Applications** [Ejsmont, 2015, Chapter 1].

10.2. INSTALLABILITY

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

The installability of general-purpose software systems has over time become a **decreasing concern, or, in some cases, even a non-concern**, because of the availability of automated or partially automated installers for such systems, and especially because of the rise of **Software-as-a-Service (SaaS)**. For example, such systems include certain operating systems, programming language compilers, book readers, media players, and so on.

However, the installability of a software system **can** be a **major concern** if it is based on assumptions that cannot (or cannot easily) be met by the environment it is expected to operate in.

For example, this can happen if a software system (goes significantly **beyond the 'lowest common denominator'** and) uses character set, fonts, libraries, or other resources that are relatively **scarce** at the time of installation.

The installation of a software product by a novice can affect the **operability** of that software product for that person.

10.3. CO-EXISTENCE

Definition [Co-Existence] [ISO/IEC, 2001]. The capability of the software product to coexist with other independent software in a common environment sharing common resources.

This is **similar to interoperability** of a software system S, **except** that the **scope** of interoperation has been **broadened** to include software systems in **other** environments for which S was **not** originally developed for.

10.4. REPLACEABILITY

I've finally learned what "upward compatible" means. It means we get to keep all our old mistakes.
— Dennie van Tassel

You can't trust code that you did not totally create yourself.
— Ken Thompson

Definition [Replaceability] [ISO/IEC, 2001]. The capability of the software product to be used in place of another specified software product for the same purpose in the same environment.

The notion of replaceability is especially relevant towards **upgrade**.

Let S be a software system installed on an operating system O. It is **not** automatic that S will be replaceable (including that it will be backward compatible). For example, a new version of S, say S', may require changes to O, and/or O may not read files generated by S.

11. ISO/IEC 9126-1 IN PRACTICE

The ISO/IEC 9126-1 Standard has found broad use in a number of contexts [Naik, Tripathy, 2008, Chapter 17; Kanellopoulos, Antonellis, Antoniou, Makris, Theodoridis, Tjortjis, Tsirakis, 2010; Mens, Serebrenik, Cleve, 2014, Section 3.2.4.2].

12. ISO/IEC 9126-1 IN PERSPECTIVE

The adoption of ISO/IEC 9126-1 is not automatic and, indeed, presents a few **challenges**.

It has been claimed [Al-Qutaish, 2010] that "the ISO 9126-1 quality model is the **most useful** one since it has been build based on an international consensus and agreement from all the country members of the ISO".

However, the viability of has been **questioned** in a number of contexts [Al-Kilidar, Cox, Kitchenham, 2005; Madeyski, 2010; Wagner, 2013, Section 2.1.7; Mens, Serebrenik, Cleve, 2014, Page 74; Ruhe, Wohlin, 2014, Chapter 6].

In particular, the issues related to the ISO/IEC 9126-1 are the following:

- Scope
- Notion of 'Product'
- Suitability for Software Systems for Certain Domains
- Organization of Quality Characteristics
- Interdependency of Standards

12.1. SCOPE

The ISO/IEC 9126-1 (and even ISO/IEC 25010) is seen as rather general to be useful. It has been pointed out that **generality of a standard is inversely proportional to its utility** contexts [Al-Kilidar, Cox, Kitchenham, 2005].

12.2. NOTION OF 'PRODUCT'

The ISO/IEC 9126-1 provides the following definition of a software product:

> The set of computer programs, procedures, and possibly associated documentation and data.

It is **unclear** if the quality attributes in the quality model put forth by the ISO/IEC 9126-1 apply, or even make sense in their application, to **all** members of the set.

12.3. SUITABILITY FOR SOFTWARE SYSTEMS FOR CERTAIN DOMAINS

There is no mention of **accessibility** in ISO/IEC 9126-1. This limits the use of the ISO/IEC 9126-1 for **interactive systems**. However, **accessibility** is part of the ISO/IEC 25010.

There are certain quality attributes that have been deemed significant to the development of certain types of software systems, but have not been considered by **any** current standards.

There is no mention of **privacy** in ISO/IEC 9126-1 or in ISO/IEC 25010. This limits the use of these standards for **legally-sensitive systems**. There is no mention of **safety** in ISO/IEC 9126-1 or in ISO/IEC 25010. This limits the use of these standards for **safety-critical systems**. There is no mention of **sustainability** [Penzenstadler, Raturi, Richardson, Tomlinson, 2014] in ISO/IEC 9126-1 or in ISO/IEC 25010. This limits the use of these standards for **green systems** (that is, software systems aiming to support the environment). There is no mention of **dependability** [Knight, 2012] in ISO/IEC 9126-1 or in ISO/IEC 25010. This limits the use of these standards for **trustworthy systems**.

12.4. ORGANIZATION OF QUALITY CHARACTERISTICS

This **inclusion and relative positioning** of quality characteristics and sub-characteristics in the ISO/IEC 9126-1 quality model is **not rationalized** [Kitchenham, Pfleeger, 1996].

This can give a perception that the **quality model development process is ad-hoc** and **the quality model itself is arbitrary**. It also presents an obstacle towards the measurement of quality characteristics and sub-characteristics [Veenendaal, Hendriks, Vonderen, 2002; Jung, Kim, Chung, 2004; Pfleeger, Atlee, 2006].

This is reminiscent of the **criticisms of standards** in general, especially the perception that, at times, they are based on the **opinions of 'experts' rather than on rigorous scientific experimentation** [Pfleeger, Fenton, Page, 1994].

12.5. INTERDEPENDENCY OF STANDARDS

The ISO/IEC 9126-1 Standard is **not isolated** from the other International Standards from the ISO/IEC, such as the rest in the series of ISO/IEC 9126 Standards and the series of ISO/IEC 15504 Standards.

In other words, a **commitment** to the ISO/IEC 9126-1 Standard has an **indirection**, namely the necessity to (1) **understand** the other standards on which it depends upon (for theoretical purposes), and (2) be **consistent** with the other standards that depend on it (for practical purposes).

13. EVOLUTION OF ISO/IEC 9126-1

The ISO/IEC 9126-1 Standard has **evolved** to the ISO/IEC 25010 Standard [ISO/IEC, 2011]. Further details of the ISO/IEC 25010 Standard are given in **Appendix A**.

ACKNOWLEDGEMENT

The inclusion of images from external sources is solely for non-commercial educational purposes, and their use is hereby acknowledged.

REFERENCES

[Al-Kilidar, Cox, Kitchenham, 2005] The Use and Usefulness of the ISO-IEC 9126 Quality Standard. By H. Al-Kilidar, K. Cox, B. Kitchenham. The Fourth International Symposium on Empirical Software Engineering (ISESE 2005). Noosa Heads, Australia. November 17-18, 2005.

[Allen, Barnum, Ellison, McGraw, Mead, 2008] Software Security Engineering: A Guide for Project Managers. By J. H. Allen, S. Barnum, R. J. Ellison, G. McGraw, N. R. Mead. Addison-Wesley. 2008.

[Alley, 1996] The Craft of Scientific Writing. By M. Alley. Third Edition. Springer Science+Business Media. 1996.

[Al-Qutaish, 2010] Quality Models in Software Engineering Literature: An Analytical and Comparative Study. By R. E. Al-Qutaish. The Journal of American Science. Volume 6. Number 3. 2010. Pages 166-175.

[Anastassiou, Mezei, 2015] Numerical Analysis Using Sage. By G. A. Anastassiou, R. A. Mezei. Springer International Publishing. 2015.

[Anderson, 2008] Security Engineering: A Guide to Building Dependable Distributed Systems. By R. J. Anderson. Second Edition. John Wiley and Sons. 2008.

[Ashbaugh, 2009] Security Software Development: Assessing and Managing Security Risks. By D. A. Ashbaugh. CRC Press. 2009.

[Avery, 2011] The Evolution of Flight Management Systems. By D. Avery. IEEE Software. Volume 28. Number 1. 2011. Pages 11-13.

[Bass, Clements, Kazman, 2003] Software Architecture in Practice. By L. Bass, P. Clements, R. Kazman. Second Edition. Addison-Wesley. 2003.

[Beck, 2003] Test-Driven Development: By Example. By K. Beck. Addison-Wesley. 2003.

[Beck, Andres, 2005] Extreme Programming Explained: Embrace Change. By K. Beck, C. Andres. Second Edition. Addison-Wesley. 2005.

[Beckers, 2015] Pattern and Security Requirements: Engineering-Based Establishment of Security Standards. By K. Beckers. Springer International Publishing. 2015.

[Brooks, 1995] The Mythical Man-Month: Essays on Software Engineering. By F. P. Brooks, Jr. Second Edition. Addison-Wesley. 1995.

[Brost, Hoffmann, 2015] Identifying Security Requirements and Privacy Concerns in Digital Health Applications. By G. S. Brost, M. Hoffmann. In: Requirements Engineering for Digital Health. S. A. Fricker, C. Thümmler, A. Gavras (Editors). Springer International Publishing. 2015. Pages 133-154.

[Cranor, Garfinkel, 2005] Security and Usability. By L. F. Cranor, S. Garfinkel. O'Reilly Media. 2005.

[Critchley, 2015] High Availability IT Services. By T. Critchley. CRC Press. 2015.

[De Luca, Lindqvist, 2015] Is Secure and Usable Smartphone Authentication Asking Too Much? By A. De Luca, J. Lindqvist. Computer. Volume 48. Number 5. 2015. Pages 64-68.

[DeNardis, 2011] Opening Standards: The Global Politics of Interoperability. By L. DeNardis (Editor). The MIT Press. 2011.

[Deng, 2010] Privacy Preserving Content Protection. By M. Deng. Ph.D. Thesis. University of Leuven. Leuven, Belgium. 2015.

[Drosg, 2009] Dealing with Uncertainties: A Guide to Error Analysis. By M. Drosg. Second, Enlarged Edition. Springer-Verlag. 2009.

[Ejsmont, 2015] Web Scalability for Startup Engineers: Tips and Techniques for Scaling Your Web Application. By A. Ejsmont. McGraw-Hill Education. 2015.

[Felderer, Büchler, Johns, Brucker, Breu, Pretschner, 2016] Security Testing: A Survey. By M. Felderer, M. Büchler, M. Johns, A. D. Brucker, R. Breu, A. Pretschner. Advances in Computers. Volume 101. 2016. Pages 1-51.

[Fenton, Pfleeger, 1997] Software Metrics: A Rigorous and Practical Approach. By N. E. Fenton, S. L. Pfleeger. International Thomson Computer Press. 1997.

[Fernandes, Machado, 2016] Requirements in Engineering Projects. By J. M. Fernandes, R. J. Machado. Springer International Publishing. 2016.

[Fricker, Thümmler, Gavras, 2015] Requirements Engineering for Digital Health. By S. A. Fricker, C. Thümmler, A. Gavras (Editors). Springer International Publishing. 2015.

[Garfinkel, Lipford, 2014] Usable Security: History, Themes, and Challenges. By S. Garfinkel, H. R. Lipford. Morgan and Claypool. 2014.

[Gorton, 2011] Essential Software Architecture. By I. Gorton. Springer-Verlag. 2011.

[Hansen, Jonasson, Neukirchen, 2011] An Empirical Study of Software Architectures' Effect on Product Quality. By K. M. Hansen, K. Jonasson, H. Neukirchen. Journal of Systems and Software. Volume 84. Number 7. 2011. Pages 1233-1243.

[Hirsch, 2012] Web Application Privacy Best Practices. By F. Hirsch (Editor). World Wide Web Consortium (W3C) Working Group Note. July 03, 2012.

[Howard, LeBlanc, Viega, 2010] 24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them. By M. Howard, D. LeBlanc, J. Viega. McGraw-Hill. 2010.

[Howard, Lipner, 2006] The Security Development Lifecycle. By M. Howard, S. Lipner. Microsoft Press. 2006.

[ICO, 2014] Conducting Privacy Impact Assessments Code of Practice, Version 1.0. Information Commissioner's Office (ICO). U.K. February 2014.

[IEEE, 1994] IEEE Standard 1228-1994. IEEE Standard for Software Safety Plans. The Institute of Electrical and Electronics Engineers (IEEE) Computer Society. 1994.

[IEEE, 2014] Guide to the Software Engineering Body of Knowledge (SWEBOK) Version 3.0. The Institute of Electrical and Electronics Engineers (IEEE) Computer Society. 2014.

[ISO, 2008] ISO 9241-20:2008. Ergonomics of Human-System Interaction -- Part 20: Accessibility Guidelines for Information/Communication Technology (ICT) Equipment and Services. The International Organization for Standardization (ISO). 2008.

[ISO/IEC, 2001] ISO/IEC 9126-1:2001. Software Engineering -- Product Quality -- Part 1: Quality Model. The International Organization for Standardization (ISO)/The International Electrotechnical Commission (IEC). 2001.

[ISO/IEC/IEEE, 2010] ISO/IEC/IEEE 24765:2010. Systems and Software Engineering -- Vocabulary. The International Organization for Standardization (ISO)/The International Electrotechnical Commission (IEC)/IEEE Computer Society. 2010.

[ISO/IEC, 2011] ISO/IEC 25010:2011. Systems and Software Engineering -- Systems and Software Quality Requirements and Evaluation (SQUARE) -- System and Software Quality Models. The International Organization for Standardization (ISO)/The International Electrotechnical Commission (IEC). 2011.

[Jung, Kim, Chung, 2004] Measuring Software Product Quality: A Survey of ISO/IEC 9126. By H.-W. Jung, S.-G. Kim, C.-S. Chung. IEEE Software. Volume 21. Number 5. 2004. Pages 88-92.

[Kanellopoulos, Antonellis, Antoniou, Makris, Theodoridis, Tjortjis, Tsirakis, 2010] Code Quality Evaluation Methodology using the ISO-IEC 9126 Standard. By Y. Kanellopoulos, P. Antonellis, D. Antoniou, C. Makris, E. Theodoridis, C. Tjortjis, N. Tsirakis. International Journal of Software Engineering and Applications. Volume 1. Number 3. 2010. Pages 17-36.

[Kitchenham, Pfleeger, 1996] Software Quality: The Elusive Target. By B. Kitchenham, S. L. Pfleeger. IEEE Software. Volume 13. Issue 1. 1996. Pages 12-21.

[Knight, 2012] Fundamentals of Dependable Computing for Software Engineers. By J. Knight. CRC Press. 2012.

[Laird, Brennan, 2006] Software Measurement and Estimation: A Practical Approach. By L. M. Laird, M. C. Brennan. John Wiley and Sons. 2006.

[Laplante, 2007] What Every Engineer Should Know About Software Engineering. By P. A. Laplante. CRC Press. 2007.

[Leveson, 1995] Safety as a System Property. By N. G. Leveson. Communications of the ACM. Volume 38. Number 11. 1995. Page 146.

[Leveson, 2011] Applying Systems Thinking to Analyze and Learn from Events. By N. G. Leveson. Safety Science. Volume 49. Issue 1. 2011. Pages 55-64.

[Lincke, 2015] Security Planning: An Applied Approach. By S. Lincke. Springer International Publishing. 2015.

[Madeyski, 2010] Test-Driven Development: An Empirical Evaluation of Agile Practice. By L. Madeyski. Springer-Verlag. 2010.

[Madhavji, Fernández-Ramil, Perry, 2006] Software Evolution and Feedback: Theory and Practice. By N. H. Madhavji, J. C. Fernández-Ramil, D. E. Perry (Editors). John Wiley and Sons. 2006.

[Menascé, Almeida, Dowdy, 2004] Performance by Design: Computer Capacity Planning by Example. By D. A. Menascé, V. A. F. Almeida, L. W. Dowdy. Prentice-Hall. 2004.

[Mens, Serebrenik, Cleve, 2014] Evolving Software Systems. By T. Mens, A. Serebrenik, A. Cleve (Editors). Springer-Verlag. 2014.

[Naik, Tripathy, 2008] Software Testing and Quality Assurance: Theory and Practice. By K. Naik, P. Tripathy. John Wiley and Sons. 2008.

[Parnas, 1994] Software Aging. By D. L. Parnas. The Sixteenth International Conference on Software Engineering (ICSE 1994). Sorrento, Italy. May 16-21, 1994.

[Penzenstadler, Raturi, Richardson, Tomlinson, 2014] Safety, Security, Now Sustainability: The Nonfunctional Requirement for the 21st Century. By B. Penzenstadler, A. Raturi, D. Richardson, B. Tomlinson. IEEE Software. Volume 31. Issue 3. 2014. Pages 40-47.

[Pfleeger, Fenton, Page, 1994] Evaluating Software Engineering Standards. By S. L. Pfleeger, N. Fenton, S. Page. Computer. Volume 27. Issue 9. 1994. Pages 71-79.

[Pfleeger, Atlee, 2006] Software Engineering: Theory and Practice. By S. L. Pfleeger, J. M. Atlee. Prentice-Hall. 2006.

[Pfitzmann, Hansen, 2010] A Terminology for Talking about Privacy by Data Minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management. By A. Pfitzmann, M. Hansen. Technical Report. Technische Universität Dresden, Dresden, Germany and Unabhängiges Landeszentrum für Datenschutz Schleswig-Holstein, Kiel, Germany. August 10, 2010. URL: http://dud.inf.tu-dresden.de/Anon_Terminology.shtml.

[Refsdal, Solhaug, Stølen, 2015] Cyber-Risk Management. By A. Refsdal, B. Solhaug, K. Stølen. Springer International Publishing. 2015.

[Ruhe, Wohlin, 2014] Software Project Management in a Changing World. By G. Ruhe, C. Wohlin (Editors). Springer-Verlag. 2014.

[Sangwan, 2015] Software and Systems Architecture in Action. By R. S. Sangwan. CRC Press. 2015.

[Schumacher, Fernandez-Buglioni, Hybertson, Buschmann, Sommerlad, 2006] Security Patterns: Integrating Security and Systems Engineering. By M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, P. Sommerlad. John Wiley and Sons. 2006.

[Shaw, Garlan, 1996] Software Architecture: Perspectives on an Emerging Discipline. By M. Shaw, D. Garlan. Prentice-Hall. 1996.

[Shostack, 2014] Threat Modeling: Designing for Security. By A. Shostack. John Wiley and Sons. 2014.

[Stalla-Bourdillon, Phillips, Ryan, 2014] Privacy vs. Security. By S. Stalla-Bourdillon, J. Phillips, M. D. Ryan. Springer. 2014.

[Veenendaal, Hendriks, Vonderen, 2002] Measuring Software Product Quality. By E. van Veenendaal, R. Hendriks, R. van Vonderen. Software Quality Professional. Volume 5. Number 1. 2002. Pages 6-13.

[Verma, Ajit, Karanki, 2016] Reliability and Safety Engineering. By A. K. Verma, S. Ajit, D. R. Karanki. Second Edition. Springer-Verlag. 2016.

[Wagner, 2013] Software Product Quality Control. By S. Wagner. Springer-Verlag. 2013.

[Wiegers, 2003] Software Requirements. By K. Wiegers. Second Edition. Microsoft Press. 2003.

[Wuyts, Verhenneman, Scandariato, Joosen, Dumortier, 2012] What Electronic Health Records Don’t Know Just Yet: A Privacy Analysis for Patient Communities and Health Records Interaction. By K. Wuyts, G. Verhenneman, R. Scandariato, W. Joosen, J. Dumortier. Health and Technology. Volume 2. Issue 3. 2012. Pages 159-183.

[Wuyts, 2015] Privacy Threats in Software Architectures. By K. Wuyts. Ph.D. Thesis. University of Leuven. Leuven, Belgium. 2015.

[Yamada, 2014] Software Reliability Modeling: Fundamentals and Applications. By S. Yamada. Springer. 2014.

[Yamada, Tamura, 2016] OSS Reliability Measurement and Assessment. By S. Yamada, Y. Tamura. Springer International Publishing. 2016.

APPENDIX A. THE ISO/IEC 25010 QUALITY MODEL

The ISO/IEC 25010 Standard [ISO/IEC, 2011] is a successor of the ISO/IEC 9126-1 Standard [ISO/IEC, 2001].

The ISO/IEC 25010 quality model consists of a number of **hierarchically structured quality characteristics** and **sub-characteristics**, as shown in Table 3.

Quality			
Functional Suitability	**Performance Efficiency**	**Compatibility**	**Usability**
• Functional Completeness • Functional Correctness • Functional Appropriateness	• Time Behavior • Resource-Utilization • Capacity	• Co-Existence • Interoperability	• Appropriateness Recognizability • Learnability • Operability • User-Error Protection • User Interface Aesthetics • Accessibility

Quality			
Reliability	**Security**	**Maintainability**	**Portability**
• Maturity • Availability • Fault Tolerance • Recoverability	• Confidentiality • Integrity • Non-Repudiation • Accountability • Authenticity	• Modularity • Reusability • Analyzability • Modifiability • Testability	• Adaptability • Installability • Replaceability

Table 3. The ISO/IEC 25010 quality model.

There are several **differences** between the ISO/IEC 9126-1 quality model and the ISO/IEC 25010 quality model. For example, **accessibility** is not part of the ISO/IEC 9126-1 quality model, but is part of the ISO/IEC 25010 quality model. In ISO/IEC 25010, accessibility is considered a part of **usability**. The ISO/IEC 25010 quality model also makes the **difference between compatibility and portability** clear.

APPENDIX B. ACCESSIBILITY

Definition [Accessibility] [ISO, 2008]. The usability of a product, service, environment, or facility by people with the widest range of capabilities.

Note 1. The concept of accessibility addresses the full range of user capabilities and is **not limited to users** who are formally recognized as **having a disability**.

Note 2. The usability-oriented concept of accessibility aims to achieve **levels of effectiveness, efficiency, and satisfaction that are as high as possible** considering the specified context of use, while paying particular attention to the **full range of user capabilities** within the user population.

B.1. ACCESSIBILITY VERSUS USABILITY

There are evident **overlaps** between accessibility and usability [ISO/IEC, 2011].

The exclusive target in both accessibility and usability is humans, and the purpose of each is the improvement of the interaction between humans and machines (computers).

However, the **emphasis** in accessibility and the emphasis in usability are **different**:

- **Accessibility.** In any interaction between humans and machines (computers), **individual capabilities** (mental and physical characteristics) of humans are a **primary concern**.

- **Usability.** In any interaction between humans and machines (computers), **individual capabilities** (mental and physical characteristics) of humans are **not a primary concern**: it is assumed that the users are **'normal'**.

This phenomenon is illustrated in Figure 10.

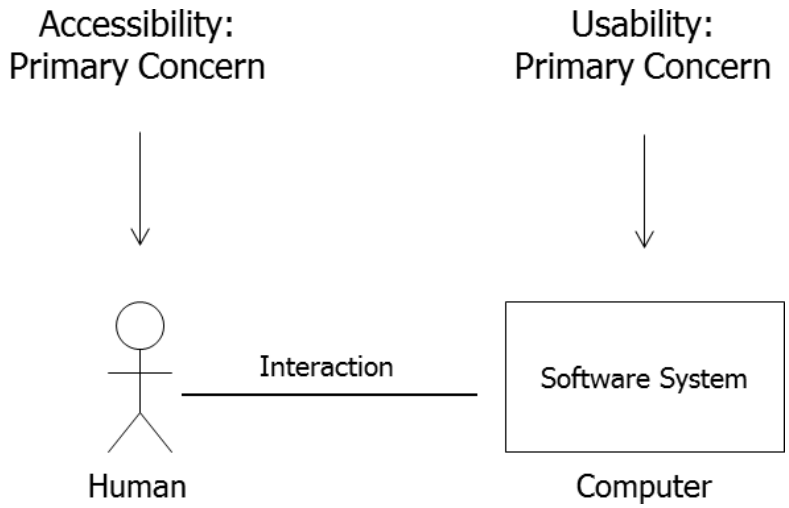


Figure 10. The attention in accessibility is on humans, whereas the attention in usability is on machines (computers).

APPENDIX C. PRIVACY

The origins of privacy lie in a **person's right to be left alone** and **have a private life** [ICO, 2014; Stalla-Bourdillon, Phillips, Ryan, 2014].

The terminology related to privacy, in comparison to other quality attributes, is still evolving [Pfitzmann, Hansen, 2010].

C.1. DEFINITION OF PRIVACY

Definition [Privacy] [Wikipedia]. The ability of an individual, group, or an organization to seclude themselves, or information about themselves, and thereby express themselves selectively.

The notion of privacy can be **specific to context** [ICO, 2014].

Definition [Informational Privacy] [Wikipedia]. The "relationship between technology and the legal right to, or public expectation of, privacy in the collection and sharing of information about one's self".

The use of the Internet, in general, and the Web, in particular, has introduced new challenges for privacy [Hirsch, 2012].

For example, a person may not want others to know about his or her **shopping habits or avenues**.

For another example, a person may not want those beyond his or her employers (and the government) to know about his or her **income**.

For yet another example, a person may not want other patrons of a café to know about his or her **credit card type or number**.

(Source: Google Images.)

Definition [Medical Privacy] [Wikipedia]. The practice of keeping information about a patient confidential.

The medical privacy involves "conversational discretion on the part of healthcare providers; the degree of disclosure to insurance companies, employers, and other third parties; and the security of medical records" [Wikipedia]. It is considered important for medical information systems [Fricker, Thümmler, Gavras, 2015].

For example, a chronically ill person may not want those beyond the medical staff to know about the fact that he or she is ill or the **nature of illness**. For another example, a patient may not want his or her **medical record** (age, blood type, allergies, if any, and so on) to be public knowledge.

C.2. CLASSIFICATION OF PRIVACY

In a **general context**, privacy can be classified as physical, informational privacy, and decisional privacy [Deng, 2010]:

- **Physical Privacy:** This is related to aspects of a person's physical location, relative to the others, such as spatial seclusion and solitude.

- **Informational Privacy:** This is related to confidentiality, secrecy, data protection, and control over personal information.

- **Decisional Privacy:** This is related to limited intrusion into decision making about things such as sex, family, religion, and health.

In a **software engineering context**, privacy can be classified as hard privacy and soft privacy [Wuyts, 2015]:

- **Hard Privacy:** This is based on the assumption that minimal personal data is divulged by the subject to third parties. The system model of hard privacy is that a data subject provides as little data as possible in order to reduce the need to "trust" other entities.

- **Soft Privacy:** This is based on the assumption that the subject has already lost control of his or her personal data, and has to trust the honesty and competence of the data controllers.

C.3. UNDERSTANDING PRIVACY

To create a better understanding of privacy, it can be decomposed into a number of attributes [Wuyts, 2015]:

- Unlinkability
- Anonymity and Pseudonymity
- Plausible Deniability.
- Undetectability and Unobservability
- Confidentiality
- Awareness
- Compliance

C.4. PRIVACY AND SOFTWARE DEVELOPMENT PROCESS

There is increasing interest in addressing privacy **early** in software development process. The LINDDUN methodology is a proposal in that direction [Deng, 2010; Wuyts, 2015].

C.5. PRIVACY AND TECHNOLOGY

The advent and use of the **Social Web** for electronic health record (EHR) and patient health record (PHR) opens new vistas for the doctors and other medical staff, as well as for the patients and their family and friends, but also brings new challenges to privacy [Wuyts, Verhenneman, Scandariato, Joosen, Dumortier, 2012].

C.6. TOOLS

Tor[4] is software for enabling anonymous communication. […] Tor's use is **intended to protect the personal privacy of users**, as well as their freedom and ability to conduct confidential communication by keeping their Internet activities from being monitored.

Tor directs Internet traffic through a free, worldwide, volunteer network consisting of thousands of relays to conceal a user's location and usage from anyone conducting network surveillance or traffic analysis.

Web Cookies Scanner[5] provides audit of HTTP cookies used by a Web Site.

[4] URL: https://torproject.org/ .
[5] URL: https://webcookies.org/ .

APPENDIX D. SAFETY

We need to place more emphasis on understanding the effects of the technology we create on the world. While caught up in the fervor and excitement of developing a revolutionary new technology with the potential to change the world in profound ways, we might be excused for **concentrating on the technical to the exclusion of the social**. But we have now matured to the point where we need to start assuming responsibility for what we do.
— Nancy G. Leveson

There are several definitions of safety [Leveson, 2011]. In the following, safety, as it pertains to software systems, is defined **recursively**:

Definition [Accident] [IEEE, 1994]. An unplanned event or series of events that results in **death, injury, illness**, environmental damage, or damage to or loss of equipment or property.

Definition [Software Hazard] [IEEE, 1994]. A software condition that is a prerequisite to an accident.

Definition [Software Safety] [IEEE, 1994]. [The] freedom from software hazards.

NATURE OF SOFTWARE AND SAFETY

It is important to note that **software is neither safe nor unsafe**. This is because software is **intangible**, and so by itself **cannot harm anyone physiologically**. It is only the **machines that are controlled by software** that can cause harm. Therefore, safety of a particular software should be considered in a **larger context** in which that software operates [Leveson, 1995]. This makes safety an emergent property.

In **safety-critical systems**, such as **healthcare (hardware and software) systems**, nuclear power plants, safety is among the quality attributes with highest priority.

LEVELS OF SAFETY

There can be different **levels of safety**. The presence of different levels of safety helps **associating criticality** with a safety requirement, which, in turn, helps **prioritizing safety requirements**.

For example, one proposal for safety levels for an aircraft is shown in Table 4 [Avery, 2011].

Safety Level	Category
A	**Catastrophic** (Meaning: A failure may lead to a catastrophe, such as a crash.)
B	**Hazardous** (Meaning: A failure may be hazardous to the safe operation of the aircraft, causing serious or fatal injuries.
C	**Major** (Meaning: A failure is not hazardous, but may lead to uncomfortable travel conditions for the crew and the passengers.)
D	**Minor** (Meaning: A failure may inconvenience the crew and the passengers by causing flight delays.)
E	**No Effect** (Meaning: A failure is not related to the safe operation of the aircraft.)

Table 4. A model for aircraft safety levels.

In some cases, safety **depends** on **other quality attributes**, such as **reliability**. For example, an aircraft that does not meet its reliability requirements will not be certified as safe for operation.