CHAPTER THREE

WHAT IS SOFTWARE QUALITY?

What is software quality? Well, the answer is that it depends on who you are asking. If you ask the user of the software, you will most likely get answers like, “it does what I expect it to do”, “easy to use”, “no bugs”, “fast in response”, and “reliable”. If you ask the IT-guy, you will get answers like, “easy to install”, “secure”, and “regularly updated”. If you ask the CFO of a company, you will hear “cheap” or “high return of investment”. If you ask the software developer, you will get answers like, “readable code”, “understandable code”, “neatly coded”, “version-controlled”, and “proper design”. If you ask the project manager of this development project, you will get answers like “in time” and “within budget”. This begs the question: who is right? I have got some news for you. All of them are right, exemplifying the diversity of software quality.

The quality of a system is the degree to which the system satisfies the stated and implied needs of its various stakeholders, and thus provides value.

retrieved 08-March-2020, https://iso25000.com/

The 1+3 Software Quality Model

To address all the different needs of stakeholders in software quality, I would want to introduce the 1+3 Software Quality Model (1+3 SQM).

Four quality-types are defined in this software quality model:

- Product Quality
- Design Quality
- Code Quality
- Organizational Quality

The reason why this software quality model is called the "1+3 SQM" is because *one* of the quality types (Organizational Quality) is the enabler of the other *three* quality types as reflected in Figure 3.1.

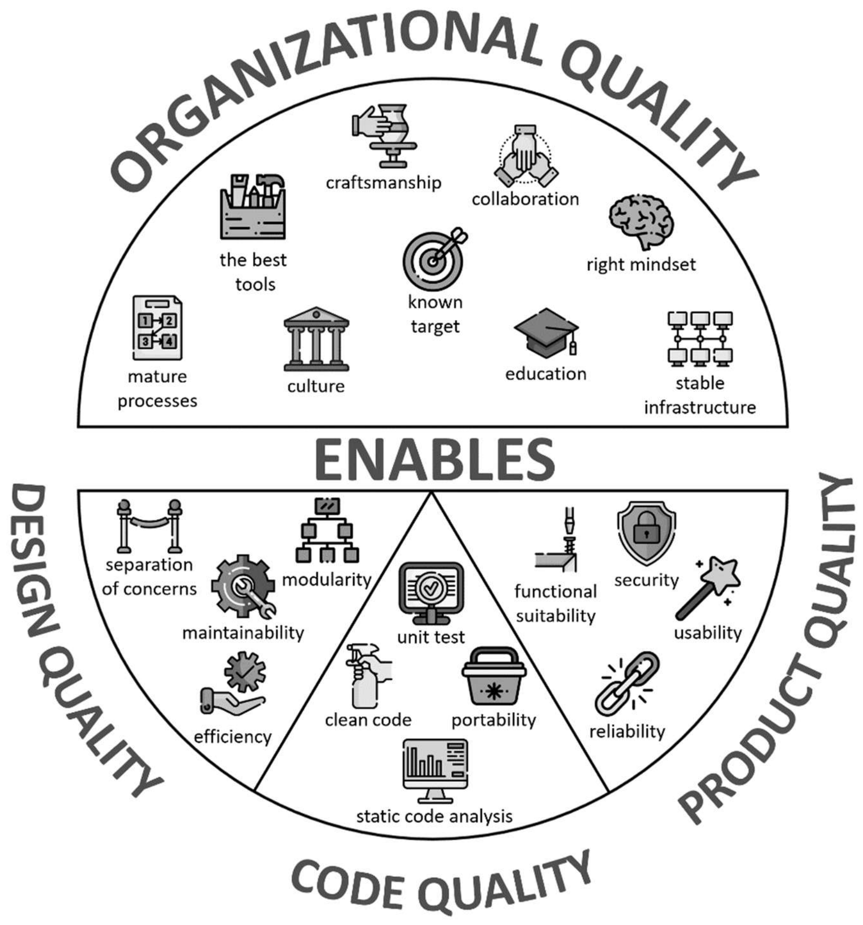
Product Quality

Product Quality is the quality of a product as perceived by the customer or user, thus implying the visibility of this quality type. A typical quality characteristic for Product Quality would be whether the product is behaving according to the expectation of the user, taking into account the fact that the product might be used from different perspectives.

Product Quality entails building the *right* product bereft of problems or errors.

Given that Product Quality is as perceived by the user or customer, one could infer that it is the *external* quality of the software.

Figure 3.1
The 1+3 SQM



Design Quality

Design Quality of software is the quality of the software design, which does include the architecture. As discussed in Chapter One “What is Software?”, separation of concerns is an important design principle applied in software development. A typical quality characteristic for Design Quality would be the efficacy with which the separation of concerns is applied in the design of the software at different design levels.

Design Quality is about mitigating complexity and understanding the structure of the software along with its internal dependencies. A software program with high levels of Design Quality is easier to maintain and to extend with new features.

Design Quality is not visible at the outside of the product and as such, it is *internal* quality of the software.

Code Quality

Like Design Quality, Code Quality refers to quality which is not outwardly visible; it is as well an *internal* quality type of the product.

Code Quality pertains to understandability of code. It is about how well the code is structured and written such that the reader of the code understands this code’s intent and operation. It is important because software engineers constantly read existing code as part of writing new code. They need to understand the existing code to be able to apply changes or to add new code.

Code of high quality is called Clean Code, which means it works, it is easy to understand and it is easy to modify and test.

Organizational Quality

The fourth type of software quality is of a different type. You might even consider it not being part of software quality, yet it is the most important quality type you can have. Here, we are talking about Organizational Quality: the ability of your organization to develop and maintain software. Organizational Quality can be divided in two subtypes: the developers' competence and the organization's competence.

Developers Competence

The most important part of Organizational Quality is your developers' competence level; the level of craftsmanship and professionalism of your developers. How well they know the domain of the software to be developed, how well they know the design and code of your product, the level of knowledge of used technologies, and their analytical skills to solve problems. In addition, their social skills and their ability to communicate effectively and collaborate well in the team are also important skills. Basically, the developers' competence is the basis for achieving good quality. Lack of developer competence is a guarantee for low software quality.

Organization Competence

To get the best out of the developers, the organization needs to provide an environment in which developers can flourish. It is important to provide a suitable environment of engineering practices, processes and tools. However, providing a culture in which developers feel secure, safe and appreciated is equally important. A culture, in which this is not the case, can completely kill the productivity of developers.

Enabling Quality

One could even state that if Organizational Quality is of very high level, the other three quality types do not need special attention. By means of the high level of craftsmanship and professionalism of your team, you will have high levels of Product Quality, Design Quality and Code Quality.

The level of Organizational Quality is the enabler or, worse case, the disabler of all three other types of software quality. As such, it is the *enabling* quality.

The Icons in the 1+3 SQM

The icons used in the 1+3 SQM illustrate important aspects for each quality-type. It does not have the intention of being complete, but provides examples for each quality type to help a better understanding of what software quality is.

The Icons of Organizational Quality

Craftsmanship represents the skill a software engineer uses to develop high-quality software. Craftsmanship is part of Organizational Quality, and is perhaps even the most important part of it.

With the right *mindset*, people will do the right things; for example, a mindset to create Clean Code.

A *Known Target* might be implemented by a Sprint Goal. It sets the direction and focus for the team.

A *culture* is important to enable people to flourish. How about a collaborative oriented culture? Or, is a competence-oriented culture more appropriate to your organization? Or perhaps even a control-oriented culture as in regulated industries?

Software programs are too big to be developed by only one person. Therefore, *collaboration* is important; within the team itself, but also with its stakeholders.

The best tools help you in doing your job. It is important to use the best tools. It is like with a carpenter: the saw needs to be sharp, otherwise it is quite difficult to saw through the wood.

A *Stable Infrastructure* of network and tooling is important for the development team. If the infrastructure is instable, each and every engineer will be hampered.

Education is the basis of knowledge. Knowledge is needed to be able to develop high-quality software. Education is an enabler in this.

A way-of-work is inevitable, whether it is an informal working together or a formal well-defined and documented process. Agreements as *mature processes* on how to develop the software need to be made in the team, and followed up.

The Icons of Product Quality

It is so important to build the right product: a product that complies to the expectations of the users, illustrated as *functional suitability*.

Software should be intuitive, in a way that it is easy to use. The *usability* of the software needs to be high.

The software should run stable without bugs. Especially in safety-critical software you cannot afford systems failing. Therefore, high *reliability* is required.

In a more and more connected world, in which computers and devices are connected through the internet, *security* is very important to prevent malicious parties from misusing these connected systems.

The Icons of Design Quality

One of the most important design paradigms for software development is *Separation of Concerns,* which mitigates complexity and is very important to be able to implement huge software systems.

High-cohesion and low-coupling are an important design paradigm to apply in software design. If applied in the right way, it will lead to *modularity* in systems.

In particular for software systems on which years or even decades development continues, maintainability is important. Maintainability is related to Technical Debt. High levels of Technical Debt decrease the maintainability of the system.

The structure of the design of the software has a big influence on the utilization of resources like CPU-cycles and memory illustrated by this *efficiency* icon.

The Icons of Code Quality

Clean Code is code that works and is easy to understand, modify and test. Source Code should be written such it is understood by other people.

Static code analysis tools analyze the source code and provide information about certain aspects of code quality like information about complexity or warnings for errors.

Unit tests are an integral part of the programming of code. In addition to the fact that solving errors, found by unit tests, is relatively easy, unit tests provide an environment in which code can be refactored in a 'safe' manner.

Portability refers to the capability of software to be ported to a different hardware and/or software platform. By nature, programming languages differ in terms of portability.

Transcendent Quality vs. Modeled Quality

Another view on quality could be transcendent versus modeled quality.

Transcendent Quality

Transcendent considers quality as something that is intuitive understood but nearly impossible to communicate, such as beauty or love.

S. Thomas Foster – Managing Quality: Integrating the Supply Chain

Transcendent quality is the type of quality which is understood by mind but difficult to articulate in words. It is subjective and thus, perceived by the observer. Having a look at the 1+3 SQM, one could think of examples of transcendent quality.

As an example, the quality of the intuitive usability of the software is transcendent quality in Product Quality. It is a subjective quality as perceived by the user. Some users will love the usage of a software product while others have difficulties in using it.

For Design Quality, one could consider the quality of the intended design as transcendent quality. Given that software provides many different ways of solving a problem, many different designs are indeed possible. Which is the best and why? Should you use Object Orientation or functional decomposition in your design? For practical purposes, software developers many times argue about best solutions in design.

Readability and understandability of code, or meaningful names of functions and variables in code, are examples of transcendent quality in Code Quality. Again, software developers can argue incessantly about the coding style.

I would like to mention craftsmanship as an example of transcendent quality in Organizational Quality.

Modeled Quality

Modeled quality is (kind of) objective and can be measured or monitored. Also, for modeled quality, one could think about different examples for each quality type out of the 1+3 SQM.

The number of found defects by testing would be an example for Product Quality, or even the number of found and not solved defects.

For Design Quality, one could think about the number of dependencies between components as a specific example for modeled quality.

Meanwhile the number of unresolved compiler warnings or coding standard violations as measured by static code analysis tooling is an example of modeled quality in Code Quality.

For Organizational Quality, the level of education in the field of software engineering of your developers is an example of modeled quality as well.

Both need to be considered

Later on in this book, there is a chapter on “Measuring Software Quality” (page 234). You would like to measure your software quality to get an opinion about it. One should consider both, transcendent and modeled quality to get an opinion about the level of software quality. There is a risk, when starting measuring, that the transcendent quality will not be considered.

ISO-25010

ISO-25010[31] is the quality model for software products as defined by the International Organization for Standardization (ISO).

The stated and implied needs of various stakeholders are represented into characteristics in the ISO-25010 software product quality model.

[31] https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

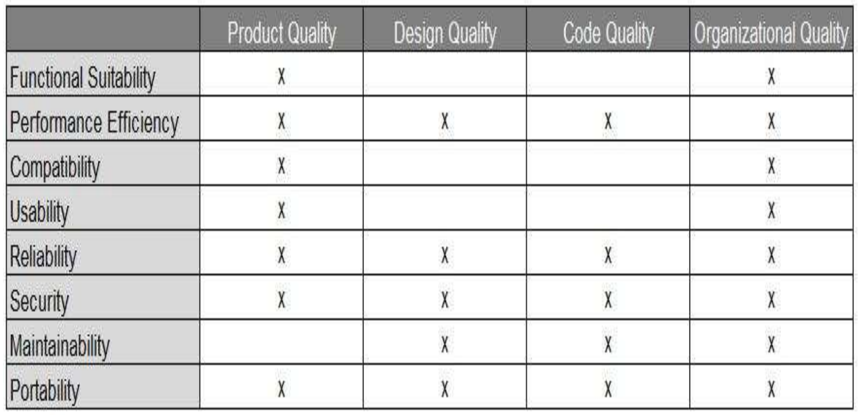
The represented characteristics in the ISO-25010 are as follows:

- Functional Suitability
- Performance Efficiency
- Compatibility
- Usability
- Reliability
- Security
- Maintainability
- Portability

The ISO-25010 model provides a different view on software quality as the 1+3 SQM does. However, there is a clear relationship between them, as expressed in the following table and a more detailed description for each ISO-25010 quality characteristic.

Table 3.1

ISO-25010 vs 1+3 SQM



	Product Quality	Design Quality	Code Quality	Organizational Quality
Functional Suitability	X			X
Performance Efficiency	X	X	X	X
Compatibility	X			X
Usability	X			X
Reliability	X	X	X	X
Security	X	X	X	X
Maintainability		X	X	X
Portability	X	X	X	X

Functional Suitability

How well the software product complies with the expectation of the product user, is reflected in this quality characteristic. You could think about completeness of features and usability. Typically, this quality characteristic is covered by the product's functional requirements. In the 1+3 SQM it is covered by the Product Quality type.

Performance Efficiency

Performance efficiency reflects the performance of the software in the context of available resources, such as memory and CPU-cycles. Especially in embedded devices, it is possible to have constrained resources for costs reasons like limited memory and small computer chips.

Performance efficiency is experienced by the user (possibly slow software with long waiting times) and highly influenced by design and code decisions. It requires additional skills and a sound understanding of the manner in which software runs on specific computer architectures, for which it has a strong relationship with all quality types of the 1+3 SQM.

Compatibility

This quality characteristic reflects how well the software collaborates with other systems. In many cases, this is accomplished by using standardized interfaces. Running a web-page or web-application in different browsers is an example that fits this quality characteristic.

Usability

Ease of use is important in order to ensure success. Intuitive user interfaces assume significance for the product. In the ISO-25010, this is covered by the usability quality characteristic contained by the Product Quality type in the 1+3 SQM. However, one needs to put lot of attention to user experience (UX) analysis and practices in the development, to achieve high levels of usability.

Reliability

Product maturity is clearly expressed by reliability. One important aspect of reliability is availability. For some systems, it is a requirement that they are available 24 hours a day and 7 days per week. Achieving such high availability requires high reliability and a high fault tolerance or recoverability in cases errors do occur. Error handling should be addressed explicitly in the design and the code of the product in order to achieve high levels of reliability.

Security

As products get more and more connected to the internet, security becomes increasingly important. Security is about being secure against being hacked. More specifically, it is about handling data in a manner that cannot and will not be abused. It is about authenticity, to ensure the system communicating with is the one it claims to be. A clear example here would be car-to-infrastructure communication. If the software of your car communicates with a traffic light, better be sure that the system communicating with indeed is the traffic light and not a hacker trying to deregulate traffic.

Security is related to Product Quality as users request secure systems. It is also related to the Design Quality and Code Quality because design and code need to be setup and created in order to be secure. Next, this requires additional specialized skills in addition to normal software engineering skills.

Maintainability

How well the software can be maintained and extended with new features is reflected by this quality characteristic. Clearly, this one pertains to the internal quality types Design Quality and Code Quality. Specifically, it is related to Technical Debt, which will be addressed more extensively at a later stage in this book.

Portability

Portability refers to the capability of software to be ported to a different hardware and/or software platform. Portability is also determined by the choice of programming language, e.g. Java is portable on different hardware platforms while programs in C are less portable. In particular, C-programming used constructs in coding to determine the level of portability.

Summary

Software quality can be considered from different viewpoints because different stakeholders have different needs. The 1+3 SQM is a model that defines four quality types: Product Quality, Design Quality, Code Quality and Organizational Quality.

Organizational Quality is the equivalent of the organization's capability to develop software, therefore *enabling* the other three quality types which are an integral part of the software product itself.

Design and Code Quality are *internal* quality types as they are not visible by the user of the software.

Product Quality refers to the level of quality as perceived by the user of the software and as such *external*.

When considering software quality, it is important to remember that some aspects of software quality cannot be measured as such due to their inherent subjectivity. These aspects are called transcendent quality and should not be neglected by myopically focusing on what can be measured.

ISO-25010 is a well-known quality standard for software products. It defines as many as eight different quality characteristics which may be of interest.