**-------------------------------------- SOFTWARE QUALITY-----------------------------------------**

We examine the qualities that are pertinent to software products and software

production processes. These qualities will become our goals in the practice of

software engineering.

The basic goal of software engineering is to produce quality software. Software

quality is a broad and important field of software engineering addressed by several

standardization bodies, such as ISO, IEEE, ANSI, etc.

**4.3.1 Definition of Software Quality**

Software quality is the:

*Conformance to explicitly stated functional and performance requirements, explicitly*

*documented development standards, and implicit characteristics that are expected of all*

*professionally developed software.*

The above definition emphasizes three important points:

1. Software requirements are the foundation from which quality is measured.

Lack of conformance to requirements is lack of quality.

2. Specified standards define a set of development criteria that guide the

manner in which software is engineered. If the criteria are not followed,

lack of quality will almost surely result.

3. There is a set of implicit requirements that often goes unmentioned. If

software conforms to its explicit requirements but fails to meet implicit

requirements, software quality is suspect.

**4.3.2 Classification of Software Qualities**

There are many desirable software qualities. Some of these apply both to the

product and to the process used to produce the product. The user wants the

software products to be reliable, efficient, and easy to use. The producer of the

software wants it to be verifiable, maintainable, portable, and extensible. The

**90** SOFTWARE ENGINEERING AND TESTING

manager of the software project wants the process of software development to be

productive and easy to control.

In this section, we consider two different classifications of software-related

qualities: internal versus external and product versus process.

**External versus Internal Qualities**

We can divide software qualities into external and internal qualities. The external

qualities are visible to the users of the system: the internal qualities are those that

concern the developers of the system. In general, users of the software only care

about the external qualities, but it is the internal qualities, which deal largely with

the structure of the software, that help developers achieve the external qualities.

For example, the internal quality of verifiability is necessary for achieving the

external quality of reliability. In many cases, however, the qualities are related

closely and the distinction between internal and external is not sharp.

**Product and Process Qualities**

We use a process to produce the software product. We can also attribute some

qualities to the process, although process qualities often are closely related to

product qualities. For example, if the process requires careful planning of system

test data before any design and development of the system starts, products

reliability will increase. Some qualities, such as efficiency, apply both to the product

and to the process.

It is interesting to examine the word product here. It usually refers to what is

delivered to the customer. Even though this is an acceptable definition from the

customer’s perspective, it is not adequate for the developer who requires a general

definition of a software product that encompasses not only the object code and the

user manual that are delivered to the customer but also the requirements, design,

source code, test data, etc. In fact, it is possible to deliver different subsets of the

same product to different customers.

**4.3.3 Software Quality Attributes**

Software quality is comprised of six main attributes (called characteristics) as

shown in Figure 4.1. These six attributes have detailed characteristics which are

considered the basic ones and which can and should be measured using suitable

metrics. At the top level, for software products, these attributes can be defined as

follows:

1. ***Functionality:*** The capability to provide functions which meet stated and

implied needs when the software is used.

SOFTWARE RELIABILITY AND QUALITY ASSURANCE **91**

**FIGURE 4.1** Software Quality Attributes

2. ***Reliability:*** The capability to maintain a specified level of performance.

3. ***Usability:*** The capability to be understood, learned, and used.

4. ***Efficiency:*** The capability to provide appropriate performance relative to the

amount of resources used.

5. ***Maintainability:*** The capability to be modified for purposes of making

corrections, improvements, or adaptation.

6. ***Portability:*** The capability to be adapted for different specified environments

without applying actions or means other than those provided for this purpose

in the product.

**4.3.4 McCall’s Quality Factors**

McCall, Richards, and Walters [MCC77] propose a useful categorization of factors

that affect software quality. These software quality factors, shown in Figure 4.2,

focus on three important aspects of a software product: its operational characteristics,

its ability to undergo change, and its adaptability to new environments.

**FIGURE 4.2** McCall’s Software Quality Factors

One attempt to identify specific product qualities that are appropriate to

software has been that of James A. McCall. He grouped software qualities into

three sets of quality factors:

**92** SOFTWARE ENGINEERING AND TESTING

\_ Product operation qualities;

\_ Product revision qualities; and

\_ Product transition qualities.

The definitions below are those given by McCall, but the reader may come

across others. These are not all inclusive: sometimes other qualities might be of

interest.

**Product Operation Quality Factors**

\_ *Correctness:* The extent to which a program satisfies its specifications and

fulfills the user’s objectives.

\_ *Reliability:* The extent to which a program can be expected to perform its

intended function with required precision.

\_ *Efficiency:* The amount of computer resources required by the software.

\_ *Integrity:* The extent to which access to software or data by unauthorized

persons can be controlled.

\_ *Usability:* The effort required for learning, operating, preparing input, and

interpreting output.

**Product Revision Quality Factors**

\_ *Maintainability:* The effort required to locate and fix an error in an operational

program.

\_ *Testability:* The effort required to test a program to ensure it performs its

intended function.

\_ *Flexibility:* The efforts required to modify an operational program.

**Product Transition Quality Factors**

\_ *Portability:* The effort required for transferring a program from one hardware

configuration and software system environment to another.

\_ *Reusability:* The extent to which a program can be used in other applications.

\_ *Interoperability:* The efforts required to couple one system to another.

**4.3.5 Software Quality Criteria**

The software quality criteria of various quality factors are depicted in Table 4.1.

SOFTWARE RELIABILITY AND QUALITY ASSURANCE **93**

**TABLE 4.1 Software Quality Criteria**

**Quality Factor Software Quality Criteria**

Correctness Traceability, consistency, completeness

Reliability Error tolerance, consistency, accuracy, simplicity

Efficiency Execution efficiency, storage efficiency

Integrity Access control, access audit

Usability Operability, training, communicativeness, input/output volume,

input/output area

Maintainability Consistency, simplicity, conciseness, modularity, self-descriptiveness

Testability Simplicity, modularity, instrumentation, self-descriptiveness

Flexibility Modularity, generality, expandability, self-descriptiveness

Portability Modularity, self-descriptiveness, machine independence, software

system independence

Reusability Generality, modularity, software system independence, machine

independence, self-descriptiveness

Interoperability Modularity, communications commonality, data commonality

**4.3.6 Representative Qualities**

In this section, we present the most important qualities of software products and

processes.

1. ***Correctness.*** A program is functionally correct if it behaves according to the

specification of the functions it should provide (called functional requirements

specifications). It is common simply to use the term “correct” rather than

“functionally correct”; similarly, in this context, the term “specifications”

implies “functional requirements specification.” We will follow this convention

when the context is clear.

The definition of correctness assumes that a specification of the system is

available and that it is possible to determine unambiguously whether or not

a program meets the specifications. With most current software systems, no

such specification exists. If a specification does exist, it is usually written in an

informal style using natural language.

2. ***Reliability.*** Informally, software is reliable if the user can depend on it. The

specialized literature on software reliability defines reliability in terms of

statistical behavior—the probability that the software will operate as expected

over a specified time interval.

**94** SOFTWARE ENGINEERING AND TESTING

Figure 4.3 illustrates the relationship between reliability and correctness. This

figure shows that the set of all reliable programs includes the set of correct

programs, but not vice versa.

**FIGURE 4.3** Relationship Between Correctness and

Reliability in the Ideal Case

3. ***Robustness.*** A program is robust if it behaves “reasonably,” even in

circumstances that were not anticipated in the requirements specification—

for example, when it encounters incorrect input data or some hardware

malfunction (say, disk crash).

Obviously, robustness is a difficult-to-define quality; after all, if we could state

precisely what we should do to make an application robust, we would be able

to specify its “reasonable” behavior completely. The robustness would become

equivalent to correctness (or reliability in the sense of Figure 4.3).

4. ***Performance.*** Performance is important because it affects the usability of the

system. If a software system is too slow, it reduces the productivity of the

users, possibly to the point of not meeting their needs. If a software system

uses too much disk space, it may be too expensive to run. If a software system

uses too much memory, it may affect the other applications that are run on the

same system, or it may run slowly while the operating system tries to balance

the memory usage of the different applications. Performance is also important

because it affects the scalability of a software system.

5 ***Verifiability.*** A software system is verifiable if its properties can be verified

easily. For example, it is important to be able to verify the correctness or the

performance of a software system.

Verifiability is usually an internal quality, although it sometimes becomes an

external quality also. For example, in many security-critical applications, the

customer requires the verifiability of certain properties. The highest level of

the security standard for a trusted computer system requires the verifiability

of the operating system kernel.

SOFTWARE RELIABILITY AND QUALITY ASSURANCE **95**

6. ***Repairability.*** A software system is repairable if it allows the correction of

its defects with a limited amount of work. In many engineering products,

repairability is a major design goal. For example, automobile engines are built

with the parts that are most likely to fail as the most accessible. In computer

hardware engineering, there is a subspecialty called Repairability, Availability,

and Serviceability (RAS).

7. ***Evolvability.*** Like other engineering products, software products are modified

over time to provide new functions or to change existing functions. Indeed,

the fact that software is so malleable makes modifications extremely easy to

apply to an implementation.

8. ***Understandability.*** Some software systems are easier to understand than

others. Of course, some tasks are inherently more complex than others.

Given tasks of inherently similar difficulty, we can follow certain guidelines

to produce more understandable designs and to write more understandable

programs. For example, abstraction and modularity enhance a system’s understandability.

9. ***Interoperability.*** “Interoperability” refers to the ability of a system to coexist

and cooperate with other systems.

With interoperability, a vendor can produce different products and allow

the user to combine them if necessary. This makes it easier for the vendor

to produce the products, and it gives the user more freedom in exactly what

functions to pay for and to combine. Interoperability can be achieved through

standardization of interfaces.

10. ***Productivity.*** Productivity is a quality of the software-production process; it

measures the efficiency of the process and, as we said before, is the performance

quality applied to the process. An efficient process results in faster delivery of

the product.

Productivity offers many trade-offs in the choice of a process. For example, a

process that requires specialization of individual team members may lead to

productivity in producing a certain product, but not in producing a variety of

products. Software reuse is a technique that leads to the overall productivity of

an organization that is involved in developing many products, but developing

reusable modules is harder than developing modules for one’s own use, thus

reducing the productivity of the group that is developing reusable modules as

part of their product development.

11. ***Timeliness.*** Timeliness is a process-related quality that refers to the ability to

deliver a product on time.

**96** SOFTWARE ENGINEERING AND TESTING

Timeliness requires careful scheduling, accurate estimation of work, and

clearly specified and verifiable milestones.

12. ***Visibility.*** A software-development process is visible if all of its steps and its

current status are documented clearly. Another term used to characterize this

property is transparency. The idea is that the steps and the status of the project

are available and easily accessible for external examination.

**4.3.7 Importance of Software Quality**

We would expect quality to be a concern of all producers of goods and services.

However, the special characteristics of software, and in particular, its intangibility

and complexity, make special demands.

1. ***Increasing Criticality of Software.*** The final customer or user is naturally

anxious about the general quality of software, especially its reliability. This

is increasingly the case as organizations become more dependent on their

computer systems and software is used more and more in areas which are

safety critical; for example, to control aircraft.

2. ***The Intangibility of Software.*** This makes it difficult to know whether a

particular task in a project has been completed satisfactorily. The results of

these tasks can be made tangible by demanding that the developers produce

‘deliverables’ that can be examined for quality.

3. ***Accumulating Errors During Software Development.*** As computer system

development is made up of a number of steps where the output from one

step is the input to the next, the errors in the earlier deliverables will be added

to those in the later steps leading to an accumulating detrimental effect, and

generally, the later in a project that an error is found the more expensive it will

be to fix. In addition, because the number of errors in the system is unknown

the debugging phases of a project are particularly difficult to control.

For these reasons quality management is an essential part of effective overall

project management.