CHAPTER

5

Testing of Software Characteristics

Contents

A standard definition of the quality of a product is the degree to which the product fulfills the expectations the customers and users have for it. We test to get information about the quality, that is, information about the fulfillment of the specified requirements, expectations, and/or implied needs.

The better the requirements, expectations, and implied needs are known, understood, and documented, and the better these specifications are, the easier it is for the developers to produce a satisfactory product and the easier it is for testers to test it. This applies both for sequential, iterative, and agile development.

This chapter is about quality characteristics or quality attributes, as they are also called. Quality attributes represent a way of structuring and expressing the expectations for a product.

In a way this chapter is superfluous for testers. It is, however, a very good idea for testers to understand what quality attributes are and how they may be expressed. With this understanding testers can contribute to the quality of a product in a number of ways from the very start of the development life cycle, for example, by expressing requirements in a testable way, reviewing requirements specifications for completeness and testability, preparing of tests to cover requirements, expectations, and implied needs, and dynamic testing of these.

The quality of a product should be measured both with regard to what the product shall do—the functional quality attributes—and with regard to how the functionality shall present itself and behave—the nonfunctional or functionality-sustaining attributes.

The *ISO 9126* standard, a standard providing a quality model for product quality, lists the following quality attributes (also sometimes called quality factors):

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

This standard is used as the basis for this chapter.

The test analyst is concerned with the functional and the usability quality attributes, and the technical test analyst is concerned with the other four quality attributes. Both are concerned with security testing, though from different perspectives.

ISO 9126 expresses that compliance to relevant standards and regulations should be verified for all the quality attributes. This is regarded as a part of the requirements specification to be tested under all circumstances and will not be discussed further here.

5.1 Quality Attributes for Test Analysts

The functionality is what the product can do. Without functionality we don't have a product at all. The functionality supports the users in their daily work or their leisure, as the case might be.

Functionality may be tested at almost all test levels. In component testing we can test the functionality implemented in single components; in system testing we can test functionality implemented in single systems across a number of integrated components; and in product testing we can test functionality implemented in the entire product. In testing how the functionality meets the expectations we can use most of the testing techniques discussed in Chapter 4.

In ISO 9126 language, the quality attributes cover the existence of a set of functions and their specified properties in the product. The functions are those that satisfy stated or implied needs, from the point of view of a stated or implied set of users.

ISO 9126 breaks the functionality attribute into the following sub-attributes:

- Suitability
- Accuracy
- Interoperability
- Security

This list may be used as a checklist for producing requirements specifications, as well as for structuring the functionality testing, if this is not being based strictly on a specification. Each of the subattributes is discussed in more detail later.

Usability, a nonfunctional attribute, is handled at the end of the section.

5.1.1 Functional Testing

5.1.1.1 Suitability Testing

In suitability testing we test the requirements or needs concerned with the presence and appropriateness of a set of functions for specified tasks and user objectives. In other words we determine whether the software is suitable for helping the user execute his or her intended tasks.

It must be remembered that (software) products are never the goal in itself. They are produced to support their users in doing their real job. Suitability of a product is about how the functionality of the product supports the tasks the users are performing.

An accountant's job is to keep accounts. This can be done entirely using pen and paper. The accountant may however use a computer system to help. When the accountant has entered the data, the system may be able to store them, calculate sums from them, and produce reports presenting the data on the screen and on paper.

If the computer system does not calculate the needed sums, the suitability is lower than if it did.

Usually the largest part by far of the requirements is requirements belonging to the suitability attribute.

The basic way all software products work is to have and/or to allow entry of some data, to handle the data in appropriate ways, and to present the data and the handling results.

The very short story about what detailed suitability could concern is:

- Data availability (i.e., how do we get data, both in terms of background or reference data and/or data from other system information and in terms of data input and change facilities and the associated levels of data validations)
- Data handling (i.e., what is data used for, for example, as event-driven interrupts or signals and—not least: calculations, actions, and so forth based on data and other input)
- Result presentation (i.e., output facilities, for example, in terms of windows and reports)

The list is by no means exhaustive. It should, however, give an idea of how functionality requirements may be structured. Requirements tools, such as UML, provide help in expressing the suitability requirements.

A few examples of suitability requirements could be:
[D.72] The product shall contain a list of all postal codes in the United Kingdom.
[K.397] A postal code must be entered as part of an address.
[K.398] The postal code may be typed directly or selected from the list of postal codes.

Suitability is often expressed in use cases, because use cases provide an excellent way of describing what the users' tasks are and how the software product should support these.

Suitability testing can take place at all testing levels. All of the techniques discussed in Chapter 4 may be used in suitability testing, though use case testing seems to be the best.

ISO 9126 references ISO 9241-10 and states that its definition of suitability corresponds to suitability for the task in that standard. ISO 9241-10 defines that suitability for the task means that the dialogue should be suitable for the user's task and skill level.

This implies that suitability is closely related to operability, a subattribute of usability discussed in Section 5.1.2.

5.1.1.2 Accuracy Testing

In accuracy testing we test the requirements or needs concerned with the product's ability to provide the right or agreed upon results or effects.

A more detailed accuracy specification could concern:

- Algorithmic accuracy: Calculation of a value from other values and the correctness of function representation
- Calculation precision: Precision of calculated values
- Time accuracy: Accuracy of time related functionality
- Time precision: Precision of time related functionality

Accuracy requirements are often implied. If in doubt about these attributes during testing it is better to ask, rather than to assume, especially if you are not an absolute domain expert.

A few examples of accuracy requirements could be:
[230] During calculation all money values shall be rounded; except in the Japanese version, where money values always shall be truncated.
[232] All money calculations in euros shall be performed with three decimals.

[233] All money calculations in currencies other than euros shall be performed with two decimals.
[234] The system shall present all values for money with two decimals on the user interface and in reports.

Accuracy can be tested at all testing levels, the earlier the better. Some accuracy testing may even take place as static tests of design and code. Many of the techniques discussed in Chapter 4 may be used in accuracy testing.

5.1.1.3 Interoperability Testing

In interoperability testing we test the requirements or needs concerned with the ability of our software system to interact with other specified systems.

No software system stands alone; it will always have to interact with other systems in the intended deployment environment, such as hardware, other software systems like operating systems, database systems, browsers, and bespoken systems, external data repositories, and network facilities.

The interoperability attributes are concerned with the specifications of all the interfaces the software system has to the external world at the time of deployment. The external systems may be part of the product being produced, or already existing products that we need to interface with.

Detailed interoperability could concern:

- Inbound interoperability: Ability to use output from standard, third party, or in-house products as input
- Outbound interoperability: Ability to produce output in the format used by standard, third-party, or in-house products
- Spawnability: Ability to activate other products
- Activatability: Ability to be activated by other products

It can be a huge task to test the interoperability because of the sheer number of possible combinations of interfaces for a system. It is often practically and/or economically impossible to achieve full coverage of all possible combinations. The Allpairs testing technique discussed in Section 4.1.7 can be used to select combinations. The intercomponent testing technique discussed in Section 4.2.9 is useful for designing test cases for interoperability testing.

A few examples of high-level interoperability requirements could be:
[I.509] The system shall obtain a record of a patient's hospitalization history from the central health register.
[I.87] When a patient is discharged, his or her hospitalization history shall be updated in the central health register.
[I.4] A discharge letter shall be produced by the letter-writer module when a patient is discharged.

Interoperability testing usually takes place at the system integration testing level.

Interoperability should not be confused with adaptability or replaceability, discussed in Section 5.2.6.

5.1.1.4 Functional Security Testing

In security testing we test the requirements or needs concerned with the ability to prevent unintended access and resist deliberate attacks intended to gain unauthorized access to confidential information, or to make unauthorized modifications to information or to the program so as to provide the attacker with some advantage or so as to deny service to legitimate users.

Detailed security could be concerning:

- Activity auditability: Log facilities for activities, actors, and so forth
- Accessability: Access control mechanisms
- Self-protectiveness: Ability to resist deliberate attempts to violate access control mechanisms
- Confinement: Ability to avoid accidental unauthorized access to facilities outside the application
- Protectiveness: Ability to resist deliberate attempts to access unauthorized facilities outside the application
- Data integrity: Protection against deliberate damage of data
- Data privacy: Protection of data against unauthorized access

Security testing can be split into functional security testing and technical security testing. In functional security testing we test the fulfillment of security attributes that can be explicitly expressed in requirements. These are typically requirements pertaining to the two first and the last of the attributes listed earlier.

A few examples of functional security requirements could be:
[623] The system shall ensure that each registered user is member of at least one of the specified user groups.
[65] The system shall ensure that only users with delete privilege may discharge patients.
[72] The system shall ensure that only users with extended read privilege may see the full hospitalization history for a patient.

Functional security testing can be performed at all testing levels, again the earlier the better, also using static test of design and code. Many of the techniques discussed in Chapter 4 may be used in functional security testing, not the least of which are the defect-based techniques discussed in Section 4.3 and the experience-based techniques discussed in Section 4.4.

5.1.2 Usability Testing

Usability is the suitability of the software for its users, in terms of the effectiveness, efficiency, and satisfaction with which specified users can achieve specified goals in particular environments or contexts of use.

The *effectiveness* of a software product is its capability to enable users to achieve specified goals with accuracy and completeness. The *efficiency* of a product is its capability to enable users to expend appropriate amounts of resources in relation to the effectiveness achieved. The *satisfaction* of a product is its capability to satisfy users.

5.1.2.1 Users Concerned with Usability

The usability attribute is related to users. It is important to get a complete overview of potential user groups and to take any kind of user characteristics into account when working with usability.

A user group for a product is a group of people who will be affected in similar ways by the product. A user group is not just the people entering data into the product and looking at the screen, though this is certainly an important user group. This group may indeed be divided into frequent users, occasional users, and rare users, or other relevant subgroups. User groups may also consist of those in charge of installing the product and those monitoring and maintaining it. Groups may be those getting information from the product, for example, in the form of reports or letters; and it may be those having to be near the product without actually interfering with it.

The list of appropriate user groups is of course very product-sensitive, and care should always be taken not to forget a potential user group.

For each of the user groups it is necessary to look at different characteristics. This could, for example, include:

- Age (e.g., preschool, children, teens, young adults, mature adults, and elderly)
- Attitude (e.g., hostile, neutral, enthusiastic)
- Cultural background
- Education (e.g., no education yet, illiterate, basic education, middle education, workman, university education)
- Disabilities (e.g., people who are dyslexic, color-blind, blind, partially sighted, deaf, mobility-impaired, or cognitively disabled)
- Gender
- Intelligence

5.1.2.2 Usability Subattributes

The ISO 9126 standard classifies usability as a nonfunctional quality attribute. It has got to do with how the functionality presents itself to the users. However, there is more to it than meets the eye; usability covers much more than just the look and feel of the product.

ISO 9126 breaks the usability attribute into the following subattributes:

- Understandability
- Learnability
- Operability
- Attractiveness

Understandability has got to do with how difficult it is to recognize the logical concept and find out how to apply it in practice. This may cover the:

- Extent to which the system maps the concepts employed in the business procedures
- Extent to which existing nomenclature is used
- Nature and presentation of structure of entities to work with
- Presentation of connections between entities

Learnability concerns the learning curve for the product. This may cover the:

- Extent to which a user of the system can learn how to use the system without external instruction
- Presence and nature of on-line help facilities for specified parts of the system
- Presence and nature of off-line help facilities for specified parts of the system
- Presence and nature of specific manuals

Operability is about what the product is like to use and control in deployment. This may cover the:

- Presence and nature of facilities for interactions with the product
- Consistency of the man-machine interface
- Presence, nature, and ordering of elements on each form
- Presence and nature of input and output formats
- Presence and nature of means of corrections of input
- Presence and nature of navigational means
- Number of operations and/or forms needed to perform a specified task

- Format, contents, and presentation of warnings and error messages
- Presence and nature of informative messages
- Pattern of human operational errors over stated periods of time under stated operational profiles according to defined reliability models

Attractiveness has got to do with how the users like the system and what may make them choose to acquire it in the first place. This may cover the:

- Use of colors
- Use of fonts
- Use of design elements, such as drawings and pictures
- Use of music and sounds
- Use of voices (male and female), languages, and accents
- Layout of user interfaces and reports
- Presence and nature of nontechnical documentation material
- Presence and nature of technical documentation material
- Presence and nature of specified demonstration facilities
- Presence and nature of marketing material

5.1.2.3 Accessibility

In recent years there has been more and more focus on equal opportunities, not the least of which are for people with disabilities. This also concerns software systems, which must be accessible and operable for everybody.

Rules are, for example, expressed in the Disability Discrimination Act covering United Kingdom and Australia, and Section 508 for the United States.

A special and important attribute for usability of a product is therefore accessibility, even though this is not explicitly mentioned in ISO 9126.

In this context accessibility is the ease with which people with disabilities can operate the product.

Accessibility may cover the:

- Use of colors, especially mixtures of red and green
- Possibility of connecting special facilities, such as speaker reading the text aloud, Braille keyboard, voice recognition, and touch screens
- Possibility of using the product entirely by key strokes and/or voice commands
- Facilities for multiple key pressure using only one finger or other pointing device
- Possibility of enlarging forms and/or fonts
- Navigation consistency

A number of standards cover various aspects of accessibility aspects, including Web Contents Accessibility Guidelines from the World Wide Web Consortium (W3C), an international consortium working on Web standards.

5.1.2.4 Establishing Usability Requirements

Like all other requirements, usability requirements should be expressed as explicitly as possible. Usability requirements can be derived from usability assessments (usually called by the very misleading name usability test, and also known as formative evaluation).

Usability assessment is a requirements elicitation technique—and it should be performed early, not on the finished product.

A usability assessment is performed by representative users who are given tasks to complete on a prototype of the products. This can be hand-drawn sketches of forms or mock-ups of the forms made in, for example, PowerPoint. Any thoughts and difficulties the users have in completing the tasks are recorded. This is best done if the users can be made to "think aloud" during the assessment.

After the usability assessment the comments are analyzed; the prototype may be changed and assessed again; and finally the usability requirements are derived.

Usability assessments can be done very primitively by review of prototypes and storyboards, or very sophisticatedly in purpose-built usability labs with two-way mirrors and video equipment.

People with different skills, for example, specialists in sociology, psychology, and ergonomics, may participate in the elicitation and documentation of usability requirements, specific standards, or special considerations to be made.

The usability requirements must be measurable. A requirement like this:

~~The user interface shall be nice to look at.~~

is seen all too often, but it is no good.

Here are a few examples of measurable usability requirements:

{UR.518} At least 95% of the primary users of the product shall answer either "very good" or "good," when asked about their opinion of the look and feel of the user interface in the survey to be carried out two months after deployment.

{UR.523} At least 80% of estate agents with a minimum of five years experience shall be able to complete the task described in the use case {UC.78} in less than 30 minutes after 20 minutes instruction.

{UR.542} All push buttons shall be placed right aligned at the bottom of the forms.

{UR.557} All forms shall have an online help facility describing the purpose and the syntax of all the fields on the form.

{UR.516} All tasks described in Section 4.1 shall be completable with a maximum of five clicks.

Note that these usability requirements are nonfunctional, or functionality-sustaining. They cannot be expressed independently of functionality but must refer to the functionality to which they apply.

5.1.2.5 Testing Usability

Usability may be tested in various ways during the development life cycle. Techniques to use may be:

- Static tests
- Verification and validation of the implementation
- Surveys and questionnaires

Static tests can be performed as reviews and inspections of usability specifications. These may include so-called heuristic evaluation, where the design of the user interface is verified against recognized usability principles. Static testing finds defects early and is hence very cost-effective, not least of all for usability where mistakes in the user interface may be very expensive to remedy late in the development life cycle. Static testing is further discussed in Chapter 6.

The *verification and validation* of the implementation of the usability requirements are performed on the working system. Here the focus is on the usability requirements associated with, but not identical to, the functional requirements. Test procedures, use cases, or scenarios may be used to express what is to be done, whereas the actual usability testing is about whether the usability requirements are fulfilled.

The usability requirements may be in the form of requirements stated in natural languages, as the examples above show, but they may also be expressed in terms of prototypes or drawings. These may be more difficult to test against, but it is important to verify that an implementation is in fact reflecting the prototype agreed on by the future users.

In this form of usability testing it is particularly important that the test environment reflects the operational environment, not least of all in terms of space, light, noise, and other disturbing factors.

Coverage may be measured using the usability requirements as the coverage element.

The ultimate validation is the user acceptance test where the finished product should be accepted by the users as being the system that fulfills their requirements, expectations, and needs. Obviously great care should be taken all along the development to ensure that acceptance may actually be the results of the acceptance testing. Serious defects and failures identified at this point of time may turn out to be very expensive.

Other tests like syntax tests of input fields and tests of messages to users may be combined with usability, even though they (in a strictly ISO 9126-speaking sense) belong to functional testing.

Not all requirements can be tested in a static and dynamic testing. *Surveys and questionnaires* may be used where subjective measures, such as the percentage of representative future users who like or dislike the user interface, are needed.

The questions must be worded to reflect what we want to know about the users' feelings towards the product. We can make our own, or we may use standardized surveys such as SUMI or WAMMI.

SUMI, *The Software Usability Measurement Inventory,* is a tested and proven method of measuring software quality from the end user's point of view. It can assist with the detection of usability flaws before a product is shipped, and it is backed by an extensive reference database embedded in an effective analysis and report-generation tool. SUMI provides concrete measurements of usability, and these may be used as inspiration for usability requirements or completion criteria. See more on http://sumi.ucc.ie.

WAMMI is a Web analytics service to help Web site owners accomplish their business goals by measuring and tracking user reactions to Web site ease of use. See more on http://www.wammi.com.

5.2 Quality Attributes for Technical Test Analysts

As important as the functionality of a product may be, it cannot stand alone. The functionality will always behave and present itself in certain ways. This is what we call the nonfunctional or functionality-sustaining attributes of the product.

Historically these attributes have been neglected when requirements have been specified, and testers have "tested" some of the nonfunctional quality attributes based on their experience. This testing was very often just a negative test: The basic idea was to get the product to fail to see how much it could cope with without knowing what the needs and expectations were.

This ought not to happen. Nonfunctional requirements should be defined for all the functionality for the product in the requirements specification.

There are many suggestions for what nonfunctional requirements should cover.

Some classics standards, which have been around for quite some time, are listed here with the quality attributes they include:

ISO 9126	Functionality, reliability, usability, efficiency, maintainability, portability
McCall and Matsumoto	Integrity, correctness, reliability, usability, efficiency, maintainability, testability, flexibility, portability, interoperability, reusability
IEEE 830	Performance, reliability, availability, security, maintainability, portability
ESA PSS-05	Performance, documentation, quality, safety, reliability, maintainability

A working group under British Computer Society is currently working on a new standard for nonfunctional quality attributes. This includes the following:

BCS working group	memory management, performance/ stress, procedure, reliability, security, interoperability, usability, portability, compatibility, maintainability, recovery, installability, configuration, disaster recovery, conversion

More information about this work can be found at:
www.testingstandards.co.uk

The nonfunctional requirement types covered in this section are:

- Reliability
- Efficiency
- Maintainability
- Portability

in accordance with ISO 9126. Usability testing is covered in the previous section, since it is of interest to test analysts.

Furthermore the technical aspects of security testing are covered here.

5.2.1 Technical Testing in General

In principle the nonfunctional testing is identical to the functional testing; it should be based on requirements and needs and use the test case design techniques discussed in Chapter 4.

However, testers can, and should, help developers and analysts define these requirements from the beginning; and we can review the requirements to ensure that they are comprehensive and testable.

It is important that the nonfunctional requirements are measurable and testable. This is, however, not always easy, at least not in the beginning. All too many nonfunctional requirements are expressed using words like "good," "fast," or "most of the time."

To overcome this it must be remembered that each nonfunctional requirement must be expressed using a scale, a specific goal, and possibly also acceptable limits. The circumstances under which the goal is to be achieved must also be specified. Further information, for example, stretch, achieved records, and future goal, could also be given to put the specified goal in perspective.

Let's look at an example. First a typical way of expressing a performance expectation:

"Reports must not take too long to be created."

This should make you wonder which reports we are talking about, what "too long" is, and if this is a general expectation no matter the circumstances.

The requirements could be reworded to:

"[P.65] The creation of a full report of all the clients as specified in Req. [F.89] shall not take more than 15 minutes if launched between 8:00 and 16:00 on normal weekdays."

This is much better. Now we know which report we are talking about, namely the one specified in the functional requirement [F.89]; we know that 15 minutes is acceptable, and we know that this is the expectation for a normal working day. The requirement could be even more specific, but the improvement in testability is already significant.

It can become a sport to dig behind imprecise nonfunctional expectations and find out what the real need is.

Apart from assisting during the requirements specification, technical testers must test the actual implementation of the nonfunctional requirements. This can be done at different testing levels depending on the types of requirements.

The coverage for the nonfunctional testing can be measured using nonfunctional requirements as the coverage element.

The testing of the nonfunctional quality attributes must be executed in a realistic environment reflecting the specified circumstances. This can be in terms of, for example, hardware, network, other systems, timing, place, load patterns, and operational profiles.

An operational profile is a description of:

- How many
- Of which user groups
- Will use what parts of the system
- When
- How much and/or how often

If care is not taken to ensure realistic circumstances the testing may be a complete waste of time and, even worse, create a false sense of confidence in the product.

5.2.1.1 Random Input Technique

In order to perform much of the technical testing to be discussed below, we have to set up test cases matching the operational profiles we are going to test and measure under. This means that in principle we have to define the same statistical distributions of input as those defined in the requirements.

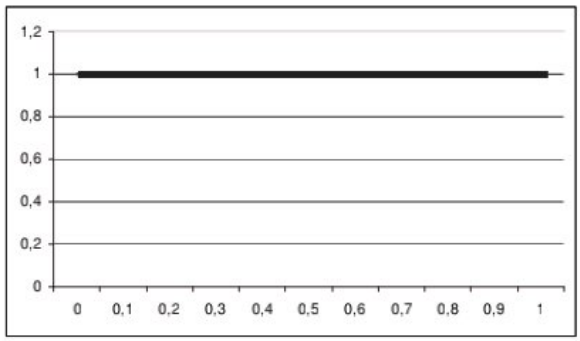
The random input technique can assist in generating input data based on a model of the input domain that defines all possible input values and their operational distribution.

Random input is not "out of the blue" random!

Random input follows the input distribution; the input values are constrained and guided by this.

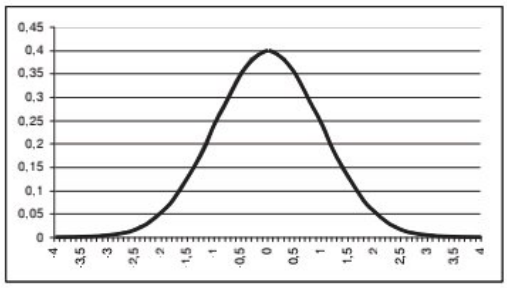
Expected input patterns can be estimated or may be known before deployment, and that knowledge can be used in testing. There are many possible distributions, but the most common ones are the uniform distribution and the normal distribution.

In uniform distribution the probability of each value in the value domain is equal.



The outcome of throwing a die follows an equal distribution. The probability of getting a 6 is the same as that of getting any of the other values.

In the normal distribution there is an average value, which we have a high probability of getting. The further away from the average a value is, the lower is the possibility of getting that value.



This is, for example, the case for the height of men in their forties. Most of these men are 1.80 meters. Only a few men stand 2 meters tall, and likewise only few reach no more than 1.6 meters.

For random testing we select input values for the test cases randomly from the input domain according to the input distribution.

If the distribution is unknown, we can always use a uniform distribution.

The pitfall of random input generation is that it can be very cumbersome to determine what the expected results of the test cases are. Random input is mainly interesting when the objective is to get the system to crash. A large number of test cases can be generated quite quickly, and long sequences of input can be run.

The expected result and the actual result are, however, usually not that important when we are performing reliability or performance testing.

The benefit of random input is that it is very cost-effective, especially if automated; the input to the test cases is cheap to develop (though the expected results may not be), the input requires little maintenance, and it gets around in the system in a trustworthy way.

Another benefit is that random input testing may find "unexpected" combinations and sequences and may detect initialization problems. It usually gives high code coverage. If automated it is a very persistent testing, and long test runs may also find resource problems, such as memory leaks or list overflows.

5.2.2 Technical Security Testing

In security testing we test the requirements or needs concerned with the ability to prevent unintended access and resist deliberate attacks intended to gain unauthorized access to confidential information, or to make unauthorized modifications to information or to the program so as to provide the attacker with some advantage or so as to deny service to legitimate users.

Detailed security attributes may be:

- Activity auditability: Log facilities for activities, actors, and so on
- Accessability: Access control mechanisms
- Self-protectiveness: Ability to resist deliberate attempts to violate access control mechanisms
- Confinement: Ability to avoid accidental unauthorized access to facilities outside the application
- Protectiveness: Ability to resist deliberate attempts to access unauthorized facilities outside the application
- Data integrity: Protection against deliberate damage of data
- Data privacy: Protection of data against unauthorized access

Security testing can be split into functional security testing and technical security testing. In functional security testing we test the fulfillment of security attributes that can be explicitly expressed in requirements. In technical security testing we take on a much broader perspective.

It is impossible to express all technical security issues as testable requirements; there are simply too many ways in which things can go wrong and too many ways in which people with dishonest intentions can try to get to our valuables.

The valuables we have in software systems are data, both in the form of data and in the form of running code. This is what we need to protect, and technical security testing is about finding out if the system is able to withstand threads to the data.

Data can be jeopardized in a number of ways, typically:

- Read to obtain other valuables (e.g., credit card numbers or sensitive data)
- Copied for use without paying (e.g., music or entire products)
- Added for harmful effects (e.g., viruses or access requests)
- Changed (e.g., sensitive information or code instructions)
- Deleted (e.g., data)

The difficulty with technical security testing is that most of the testing techniques discussed in Chapter 4 usually have very little probability of finding security defects. The majority of the security defects are not defects in the sense that the product does not fulfill states requirements or expectations in a narrow sense. The problems arise from the product having or allowing functionality that is not wanted by the future users and hence not specified, but implemented to ease the implementation without regard to the possible security side effect such an implementation might have.

In a product protected by user identities and passwords, the passwords are normally stored in an encrypted way. As long as a user is logged on, however, the password is stored in an unencrypted way in order to obtain a better performance.

Already at the design of the system care should be taken to reduce the risks of these things happening. Designers need to take on the most pessimistic and malicious minds they possibly can. This is not as easy as it sounds. Most people trust their neighbor to a large extent and find it difficult to think of ways in which they themselves may be cheated or threatened. This is why some professional hackers who have changed their ways may be employed as security consultants.

Note that there may be conflicts between security requirements and performance requirements, in the sense that higher security may cause lower performance.

Testers can perform static tests on design and code implementation to look for vulnerabilities in the system at an early stage. Things to look out for may include:

- Use of temporary storage of sensitive data or information supposed to be kept secret (for example, passwords or encryption keys)
- Possible buffer overflows, both in input and internal data handling
- Exposure of sensitive data for example over networks
- Acceptance of unvalidated data either via a user interface or an external interface

The list is by no means exhaustive.

A special security issue is logical bombs or so-called Easter eggs, where harmful code has been written into the components during development. Static testing is the only technique that may find this.

Examples of logical bombs may be:
An IF clause that is only true on specific data, where data might be erased.
An IF clause that is only true for a specific bank account number, to which an extra amount of money is debited.

The testing technique of attacks, discussed in Section 4.4.4 is very useful for technical security testing. Checklists of effective attacks should be kept up-to-date.

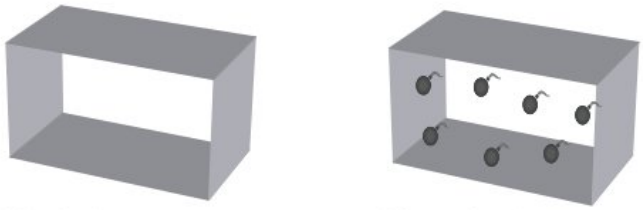
A more systematic approach perhaps is one supported by tools. A profile of the product in term of versions of operating software and middleware, identifications of developers and users, and details about internal networks is created by the use of a tool. Based on this, tools can scan the product for

known vulnerabilities, and this information can be used to develop targeted attack plans.

It is not possible to measure coverage for technical security testing, except in the form of items on checklists covered by a test.

5.2.3 Reliability Testing

Reliability is the probability that software will not cause the failure of a system for a specified time under specified conditions.



No faults
= 100% reliability

Many faults
= *x*% reliability

Concerning reliability we have got to be realistic: It is impossible to produce 100% fault-free products!

Functional testing and reliability testing are connected. The goal of functional testing is to obtain the highest possible reliability of the product within the given limits. The goal of reliability testing is to valuate the reliability we have obtained.

The test object is the complete product. The reliability testing should be based on operational profiles specified for the product, and reliability goals expressed in requirements. This is not always easy or efficient to set up. In fact it is sometimes impossible to perform reliability testing before the product is in operation. Usually this is acceptable, and the reliability test will be part of the final acceptance test.

A specific reliability may be used as a test completion or test exit criterion for the system testing, allowing for earlier reliability testing.

During reliability testing it is essential to collect measurements for the evaluation. Failures are registered when they appear "spontaneously," and they must be categorized and countable. We must also collect other measurements as appropriate, such as time and number of transactions.

The test must go on until "reliable" data has been obtained. This can take quite a while, certainly days and maybe even weeks or months.

The ISO 9126 standard breaks the reliability attributes down into a number of subattributes. There are:

- Maturity
- Fault tolerance (robustness)
- Recoverability

The standard explains each of them. These explanations are primarily intended as inspiration for nonfunctional requirements. They can also be used as checklists for static testing of reliability requirements, design, and implementation, and as inspiration for checklist-based reliability testing.

5.2.3.1 Maturity Testing

Maturity is the frequency of failures as a result of faults in the software.

A product's expected maturity is often expressed in terms of

- Mean time between failures (MTBF)
- Failures per test hour
- Failures per production time (typically months)
- Failures per number of transactions

The metrics may be further detailed by categorizing the accepted types of failures, for example, by severity.

Ex.

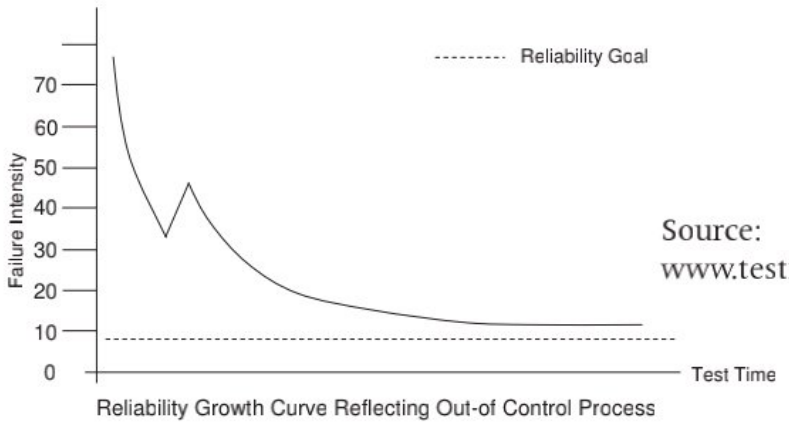
A few examples of reliability requirements are:
[34] The MTBF for the product shall be more than one month on average in the first year of production.
[72] The product shall have no more than two failures of severity 1 reported in the first six months of production.
[281] The product shall have less than three failures per 10,000 transactions.

Ex.

For testing completion criteria we may have reliability goals like:
The testing can stop when less than one failure of severity 1 has been found during 20 hours of testing on average over at least two weeks of testing.

The results of the reliability testing can be graphic presentations of the measurements using a reliability growth model.

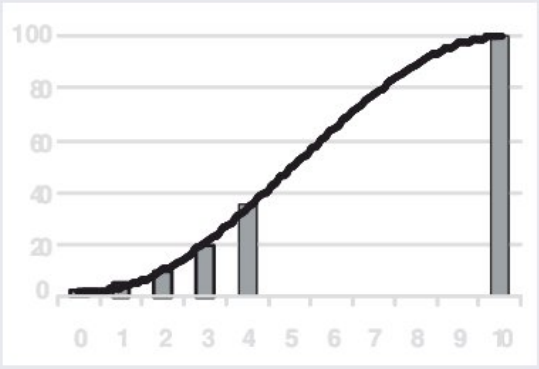
The following figure shows a reliability growth curve for a product where the reliability is getting better, but not achieving its goal.



Reliability Growth Curve Reflecting Out-of Control Process

Source:
www.testingstandards.co.uk

Another way to present reliability data is in an S-curve. Here the total number of failures found over time is plotted against time.



The figure shows an S-curve for the total number of failures for each test week. The expected number of failures after 10 weeks' testing is 100. So far it looks as if the pattern of failures found follows the expectations.

S-curves are often used to determine when to stop the test. They are also useful for predicting when a reliability level will be reached. Furthermore, S-curves may be used to demonstrate the impact on reliability of a decision to deliver the software NOW!

In cases where late or complete reliability testing is not acceptable we must either reword the requirements, use statistical proof, or use analysis.

Reliability evaluation or reliability estimation is an activity where we analyze the fault-finding curves we have produced. We extrapolate from the curves and predict how many defects are left in the product.

Another way to predict remaining defects is to use estimation models based on the structure of the program, for example, knowledge of the size, the complexity, and the data transactions.

5.2.3.2 Robustness Testing

Fault tolerance or robustness is the product's ability to maintain a specified level of performance in the presence of software defects or infringement of a specified interface.

This may cover the product's:

- Containment of defects to specified parts of the system
- Reactions to failures of a given severity
- Self-monitoring of the operations and self-identification of defects
- Ability to allow specified work to continue after a failure of specified severity for specified parts of the system under specified conditions
- Loss of specified operations (functionality requirement or set of functionality requirements) in case of failure of specified severities in specified periods of time for specified parts of the system

- Loss of specified data in case of failure of specified severities in specified periods of time for specified parts of the system

The failures to consider in technical robustness testing are those forced on the product from external sources. Failures due to internal defects should be handled in the functional testing.

Failures from external sources could, for example, be caused by lack of external storage capacity, external data storage not found, external services not available, or lack of memory.

Robustness testing can, like the other technical tests, start at the requirements level, with testers reviewing robustness requirements.

Even more important is review of design. Design can be made more or less defensive, that is more or less robust to external circumstances. A simple, though often overlooked way to make systems more robust is to check the return code of all system routine calls and take action in the code when system call is unsuccessful.

Lack of memory may be due to memory leaks. Possible memory leaks could be found using dynamic analysis, for example, during component testing.

Testing robustness in system testing and/or acceptance testing may require the use of simulators or other tools to expose the product to external problems otherwise difficult to produce.

5.2.3.3 Recoverability Testing

Recoverability is the product's ability to reestablish its required level of performance and recover the data directly affected after a failure.

This may cover aspects like:

- Downtime after a failure of specified severities in specified periods of time for specified parts of the system
- Uptime during specified periods of time for specified parts of the system over a specified period of time
- Downtime during specified periods of time for specified parts of the system over a specified period of time
- Time to reestablish consistent data in case of failure causing inconsistent data
- Built-in backup facilities
- Need for duplication (standby machine)
- Redundancy in the software system
- Reporting of effects of a crash
- "Advice" in connection with restart

Downtime after a failure, including time to reestablish data, is often measured as mean time to repair (MTTR). This is closely linked to analyzability, discussed in Section 5.2.5.

5.2.4 Efficiency Testing

In efficiency testing we test the requirements or needs concerned with the product's ability to provide appropriate performance, relative to the amount of resources used, under stated conditions.

The ISO 9126 standard breaks the efficiency attributes down into the sub-attributes:

- Time behavior (performance)
- Resource utilization

The standard also explains each of these. The explanations are primarily intended as inspiration for related nonfunctional requirements. They can also be used as checklists for static testing of efficiency requirements, design, and implementation, and as inspiration for checklist-based efficiency testing.

5.2.4.1 Performance Testing

Time behavior or performance consists of the expectations towards the product's ability to provide appropriate response and processing time and throughput rates when performing its functions under stated conditions.

In short, performance is concerned with how fast specified parts of the functionality are. This may cover requirements concerning:

- The response time for specified online tasks under specified conditions
- The elapsed time for transfer of specified data under specified circumstances
- The internal processing time (for example, in CPU cycles) for specified tasks under specified conditions
- The elapsed processing time for specified batch tasks under specified conditions

Expectations towards performance must be expressed in performance requirements.

[P.65] The creation of a full report of all the clients as specified in Req. {F.89} shall not take more than 15 minutes if launched between 8:00 and 16:00 on normal weekdays.
[P.72] The creation of a full report of all the clients as specified in Req. {F.89} shall not use more than 35% of the CPU time.

Performance testing can be very expensive and time-consuming to perform. It can be especially costly to establish the correct environment to test in.

It is important to be absolutely sure that the environment and conditions are correctly specified in performance requirements and correctly established for the performance testing, not least in terms of hardware, operating system, middleware, and concurrent use patterns or operational profiles.

This is something that should be specified in connection with the requirements specification in order for the performance requirements and hence the performance test to be reliable.

Many tools support test of performance. The tools can report on response times and execution times, for example, for database lookups or data transfer, and they can identify bottlenecks in the functions, either internally in the product and/or in the network for Web-based products. Tools are discussed in Chapter 9.

In principle performance test verifies that performance requirements are fulfilled. The coverage item is performance requirements, and the coverage is hence measured as the percentage of all the performance requirements tested in a test. As for all testing, failures must be reported, and retest and regression test must be performed after corrections.

Load Testing

Load testing is a special subtype of performance testing concerned with the product's behavior under specified load conditions.

Load has two aspects, namely:

- Multiuser with ordinary realistic numbers of users
- Large, though still realistic number (volume) of concurrent users

The loads that the product is expected to be able to handle and the applicable response times must be expressed in load requirements.

We need to think about load early on during requirements specification. It is important to strike a balance in the load requirements so that they are within reason, but still take the future into account.

Load testing can be quite expensive, and it is important not to go overboard. Sometimes we may have to question the requirements: Are they realistic or "wearing both belt and braces?" In order to keep the load testing expenses under control we should use risk analysis to prioritize and plan the testing tasks.

Tools may be used to generate and/or simulate loads.

Load testing and performance testing are sometimes related in the sense that the load can be part of the condition specifications for performance requirements.

Stress Testing

Stress is an expression of the product's capability for handling extreme situations.

When planning requirements for stress handling it is a good idea to remember Murphy's law:

"If anything can go wrong, it will."

or

"The unthinkable sometimes happens anyway."

In stress-handling requirements we are confronting the risk of the system not being able to handle extreme situations. We are dealing with risks concerning people, money, data, and the environment, risks that will materialize if the system can't cope in a stress situation.

[87] The system shall not crash but issue an error message and stop execution after acknowledgment, if too much data is loaded in the data load described in requirement [DL931].

Stress handling is closely related to other nonfunctional areas. The relationship between reliability and stress is that stress looks at reliability in extreme situations. With respect to usability, stress is about handling failures with grace, so that the user is not left in the dark about what is happening. Stress in connection with load is concerned with peak load over a short span of time.

When working with stress-related requirements we will have to think about what can go wrong. For data this could be in situations with too much data, too little data, or faulty data. For usage it could be too many simultaneous users, extended use (same operation "umpteen" times, system to run without restart for many hours/days/months), or maybe dropping the system or part of the system on the floor. External events such as power failure may also cause stress in the system.

Stress-handling is particularly important for Web applications, such as e-products and e-business; in telecommunication, safety- and security-critical systems; and real-time systems.

Stress on a system can to a large extent be handled by defensive programming. This means that stress testing can start early with review of design and code.

Stress testing should have high "product" coverage. Even if some stress prevention works in one place it may not work in another part of the system.

Stress testing should be imaginative, but on the other hand it should not go overboard. When planning the stress test we need to make a risk analysis. Even the unthinkable may happen too rarely to warrant a test.

Scalability Testing

Scalability is the product's ability to meet future efficiency requirements. This is not really a test, since naturally we cannot perform it against existing requirements.

This technique is more a scalability assessment; what we are aiming at is finding out how the product reacts to growth in, for example, number of users or amounts in data.

The product's ability to keep its performance requirements may also be monitored on an ongoing basis during deployment to provide us with the opportunity to take action before the product may break under the load.

5.2.4.2 Resource Utilization Testing

Resource utilization is the expectations towards the product's use of appropriate amounts of resources when the software performs its functions, under stated conditions.

In short, resource utilization has got to do with what is used and what is needed. This may cover requirements concerning:

- The amount of CPU resources used for specified functions
- The amount of internal memory resources used for specified functions
- The amount of external memory resources used for specified functions
- Levels of memory leakage
- The presence, appropriateness, and availability of human resources, peripherals, external software, various material

In modern systems memory is rarely a problem, but it may be in, for example, games and also in real-time embedded systems. In the latter, memory usage, also referred to as memory footprint, may be the object of precise specification and thorough testing.

5.2.5 Maintainability Testing

In maintainability testing we test the requirements or needs concerned with the product's ability to be analyzed and modified. Modifications may include corrections of defects, improvements or adaptations of the software to changes in the environment, and enhancements in requirements and functional specifications.

The ISO 9126 standard breaks the efficiency attributes down into the subattributes:

- Analyzability
- Changeability

- Stability
- Testability

The standard also explains each of these. The explanations are primarily again intended as inspiration for related nonfunctional requirements and they may be used for checklists.

Maintainability testing can be performed as static testing, where the structure, complexity, and other attributes of the code and the documentation are reviewed or undergoing inspections based on the pertaining maintenance requirements. Static analysis of the code may also be used to ensure its adherence to coding standards and to obtain measurements, such as complexity measures.

The maintainability test may also be performed dynamically in the sense that specified maintenance procedures are executed and compared to pertaining requirements. What is measured in this type of maintenance testing is typically the effort involved in the maintenance activities.

The dynamic maintenance testing may be combined with other tests, typically functional testing, where the failures found and the underlying defects to be corrected may serve as those the maintainability procedures are tested with.

5.2.5.1 Analyzability Testing

Analyzability is the ability of maintainers to identify deficiencies, diagnose the cause of failures, and identify areas requiring modification to implement required changes. A way to measure analyzability is MTTR, mean time to repair.

Analyzability may cover aspects like:

- Understandability: Making the design documentation, including the source code, understood by maintainers
- Design standard compliance: Adherence to defined design standards
- Coding standard compliance: Adherence to defined coding standards
- Diagnosability: Presence and nature of diagnostic functions in the code
- Traceability: Presence of traces between elements, for example, between requirements and test cases, and requirements and design and code
- Technical manual helpfulness: Nature of any technical manual or specification

5.2.5.2 Changeability Testing

Changeability is the capability for implementation of a specified modification in the product.

Changeability may cover aspects like:

- Modularity: The structure of the software
- Code change efficiency: Capability for implementing required changes
- Documentation change efficiency: Capability for documenting implemented changes

5.2.5.3 Stability Testing

Stability is the capability of the product to avoid unexpected effects from modifications of the software, that is, to the risk of unexpected effects from modifications.

Stability may cover aspects like:

- Data cohesion: Usage of data structures
- Refailure rate: Pattern of new failures introduced as an effect of implementation of required changes

In other words, stability has got to do with the structure of the software and, not least, of the data. We might get an impression of the stability of the product during regression testing after defect correction.

Experience shows that in general 50% of the original number of defects remain in a product after defects are corrected. These are distributed like this:

- Original defects remaining 20%
- Existing defects revealed after correction of others 10%
- New defects introduced during defect correction 20%

Stability has got to do with new defects introduced during defect correction, that is, how likely is that developers or maintenance staff accidentally make new mistakes and hence place new defects in the product when they are in fact engaged in correcting identified defects.

It is of course important for an organization to get its own measurements for the rate of new defect introduction and also for the effectiveness of defect correction.

5.2.5.4 Testability Testing

Testability is the capability of validating the modified system, that is, how easy it is to perform testing of changes, either new tests or confirmation test, and how easy it is to perform regression testing.

This is influenced both by the structure of the product itself and by the

structure of the test specification and other testware.

Testability is also influenced by how configuration management is performed. The better the control over the testware, not least the test data, and the product, and the documentation of the relationships between product versions and testware versions is, the better is the testability.

5.2.6 Portability Testing

In portability testing we test the requirements or needs concerned with the product's ability to be transferred into its intended environment. The environment may include the organization in which the product is used and the hardware, software, and network environment.

The porting may be the first porting from a development or test environment into a deployment environment, or it may be the porting from one deployment environment to another at a later point in time.

Portability is primarily an issue for software products or software subsystems, not so much for, for example, hardware subsystems. These are usually part of the environment into which the software subsystem or product is being ported.

The ISO 9126 standard breaks the portability attributes down into the subattributes:

- Installability
- Coexistence
- Adaptability
- Replaceability

The standard also explains each of these. The explanations are again primarily intended as inspiration for related nonfunctional requirements and for checklists.

5.2.6.1 Installability Testing

Installability is the capability of installing the product in a specified environment.

Installability may cover aspects like:

- *Space demand:* Temporary space to be used during installation of the software in a specified environment.
- *Checking prerequisites:* Facilities to ensure that the target environment is meeting the demands of the product, for example, in terms of operating system, hardware, and middleware.
- *Installation procedures:* Existence and understandability of installation aids such as general or specific installation scripts, installation manuals, or wizards. This may also include requirements concerning the time and effort to be spent on the installation task.
- *Completeness:* Facilities for checking that an installation is complete,

for example, in terms of checklists from configuration management
- *Installation interruption:* Possibility of interrupting an installation. and rolling any work done back to leave the environment unchanged
- *Customization:* The capability of setting or changing parameters at installation time in a specified environment.
- *Initialization:* The capability of setting up initial information at installation time, both internal and external in a specified environment.
- *Deinstallation:* Facilities for removing the product partly (downgrading) or completely from the environment.

5.2.6.2 Coexistence Testing

Coexistence is the software product's capability to coexist with other independent software products in a common environment sharing common resources. Today with powerful servers, PCs, and portables and with more and more functionality in everything being controlled by software, the coexistence of systems is a growing issue.

As cars began to have more and more software installed to control the functions of the car, a common joke was that the windshield wipers would only work if the passenger in the right backseat weighed less than 80 kilos.

This example may seem far out, but failed coexistence may have the strangest effects. Such failures may be caused by systems using the same area of memory and thus getting corrupted data, or by systems affecting each other's performance.

Coexistence can be very difficult to specify, since we don't always know into which environment our software product is being placed. Precautions can be taken in the form of resource utilization requirements, which may then be compared to resource utilization of any other systems our system is going to coexist with, and the available resources in the target environment.

Coexistence testing can also be very difficult to perform, since it is usually impossible to establish correct test environments for this. Often coexistence is tested after acceptance testing of the product and the installation in the target environment. This is obviously risky, and the decision to do so should be made based on a risk analysis.

5.2.6.3 Adaptability

Adaptability is the capability of the software product to be adapted to different specified environments without applying actions or means other than those provided for this purpose for the system.

Products and systems are rarely permanent and unchangeable these days and it will often happen that a system our system interfaces with will have to

be replaced by a newer version or a completely different system. In this case our system will have to be adapted to interface with the new system in the environment instead of with the old one.

Ease of changing interfacing systems may be achieved by using communication standards, such as HTML, or by constructing the software in such a way that it can itself detect and adjust to external communication needs.

Adaptability is primarily of interest to organizations developing commercial off-the-shelf (COTS) products or systems of systems.

Adaptability may cover aspects like:

- *Hardware dependency:* Dependence on specific hardware for the system's adaptation to a different specified environment
- *Software dependency:* Dependence on specific external software for the system's adaptation to a different specified environment
- *Representation dependency:* Dependence on specific data representation for the system's adaptation to a different specified environment
- *Standard language conformance:* Conformance to the formal standard version of a programming language
- *Dependency encapsulation:* The isolation of dependent code from independent code
- *Text convertability:* The capability for converting text to fit a specified environment

5.2.6.4 Replaceability Testing

Replaceability is the capability of the product to be used in place of another specified product for the same purpose in the same environment.

This is the opposite of adaptability, because in this case our system replaces an old one. The issues for adaptability therefore also apply for replaceability. There is also a certain overlap with installability.

Furthermore replaceability may cover aspects like:

- *Data loadability:* Facilities for loading existing data into permanent storage in our system
- *Data convertability:* Facilities for converting existing data to fit into our system

Questions

1. What are the quality attributes defined by ISO 9126?
2. What are the subattributes under functionality?
3. What are the main concerns for suitability requirements and testing?
4. What do we need to be aware of concerning accuracy testing?