BITS Pilani
Pilani Campus

BITS Pilani presentation

Dr. Nagesh BS

innovate achieve lead

BITS Pilani
Pilani Campus

SE ZG501
Software Quality Assurance and Testing
Lecture No. 4

The need for a comprehensive definition of requirements

To Cover all attributes of software and aspects of the use of software, including usability aspects, reusability aspects, maintainability aspects, and so forth in order to assure the full satisfaction of the users.

The great variety of issues related to the various attributes of software and its use and maintenance, as defined in software requirements documents, can be classified into content groups called *quality factors.*

Classifications of software requirements into software quality factors

The classic model of software quality factors, suggested by McCall, consists of 11 factors.

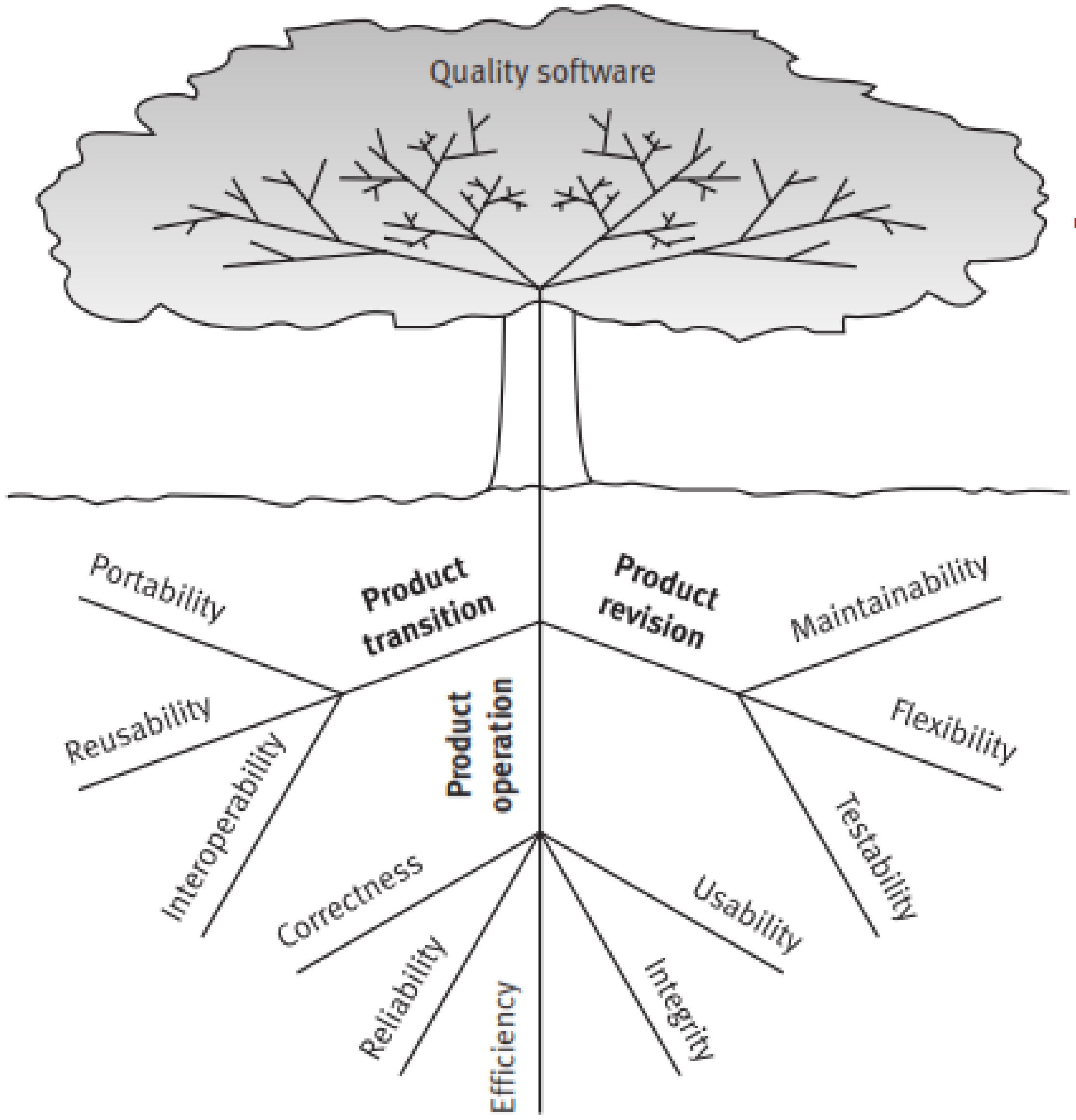
Subsequent models, consisting of 12 to 15 factors

The McCall factor model, despite the quarter of a century of its “maturation”, continues to provide a practical, up-to-date method for classifying software requirements

McCall's factor model

Classifies all software requirements into 11 software quality factors.

- Product operation factors: Correctness, Reliability, Efficiency, Integrity, Usability.
- Product revision factors: Maintainability, Flexibility, Testability.
- Product transition factors: Portability, Reusability, Interoperability.



1: McCall's factor model tree

Product operation software quality factors

Correctness : Correctness requirements are defined in a list of the software system’s **required outputs**.

For example:

- Displaying a customer's balance in the sales accounting information system.
- Regulating air supply as a function of temperature, as specified by the firmware of an industrial control unit

Output specifications are usually multidimensional

- The output mission (e.g., sales invoice printout, and red alarms when temperature rises above 250°F).
- The required accuracy of those outputs that can be adversely affected by inaccurate data or inaccurate calculations.
- The completeness of the output information, which can be adversely affected by incomplete data.
- The up-to-dateness of the information (defined as the time between the event and its consideration by the software system).
- The availability of the information (the reaction time, defined as the time needed to obtain the requested information or as the requested reaction time of the firmware installed in a computerized apparatus).
- The standards for coding and documenting the software system.

Example

The correctness requirements of a club membership information system consisted of the following:

- The output mission: A defined list of 11 types of reports, four types of standard letters to members and eight types of queries, which were to be displayed on the monitor on request.
- The required accuracy of the outputs: The probability for a non-accurate output, containing one or more mistakes, will not exceed 1%.
- The completeness of the output information: The probability of missing data about a member, his attendance at club events, and his payments will not exceed 1%.
- The up-to-dateness of the information: Not more than two working days for information about participation in events and not more than one working day for information about entry of member payments and personal data.
- The availability of information: Reaction time for queries will be less than two seconds on average; the reaction time for reports will be less than four hours.
- The required standards and guidelines: The software and its documentation are required to comply with the client's guidelines.

Reliability

Reliability requirements deal with failures to provide service.

They determine the **maximum allowed software system failure rate**, and can refer to the entire system or to one or more of its separate functions.

Example

(1) The failure frequency of a heart-monitoring unit that will operate in a hospital’s intensive care ward is required to be less than one in 20 years. Its heart attack detection function is required to have a failure rate of less than one per million cases.

(2) One requirement of the new software system to be installed in the main branch of Independence Bank, which operates 120 branches, is that it will not fail, on average, more than 10 minutes per month during the bank's office hours. In addition, the probability that the off-time (the time needed for repair and recovery of all the bank's services) be more than 30 minutes is required to be less than 0.5%.

Efficiency

Efficiency requirements deal with the hardware resources needed to perform all the functions of the software system in conformance to all other requirements.

computer's processing capabilities, data storage capability, data communication capability of the communication lines

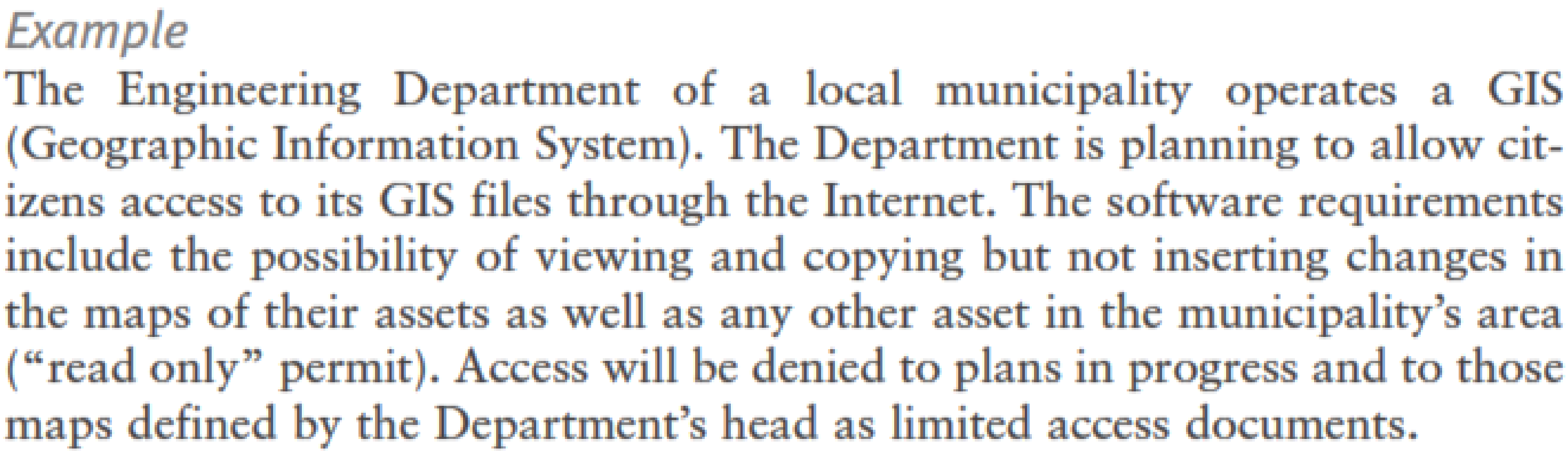
Another type of efficiency requirement deals with the time between recharging of the system's portable units, such as, information systems units located in portable computers, or meteorological units placed outdoors.

Examples

(1) A chain of stores is considering two alternative bids for a software system. Both bids consist of placing the same computers in the chain's headquarters and its branches. The bids differ solely in the storage volume: 20 GB per branch computer and 100 GB in the head office computer (Bid A); 10 GB per branch computer and 30 GB in the head office computer (Bid B). There is also a difference in the number of communication lines required: Bid A consists of three communication lines of 28.8 KBPS between each branch and the head office, whereas Bid B is based on two communication lines of the same capacity between each branch and the head office. In this case, it is clear that Bid B is more efficient than Bid A because fewer hardware resources are required.

Integrity

Integrity requirements deal with the software system security, that is, requirements to **prevent** access to unauthorized persons, to distinguish between the majority of personnel allowed to see the information (“**read permit**”) and a limited group who will be allowed to add and change data **(“write permit”),** and so forth.

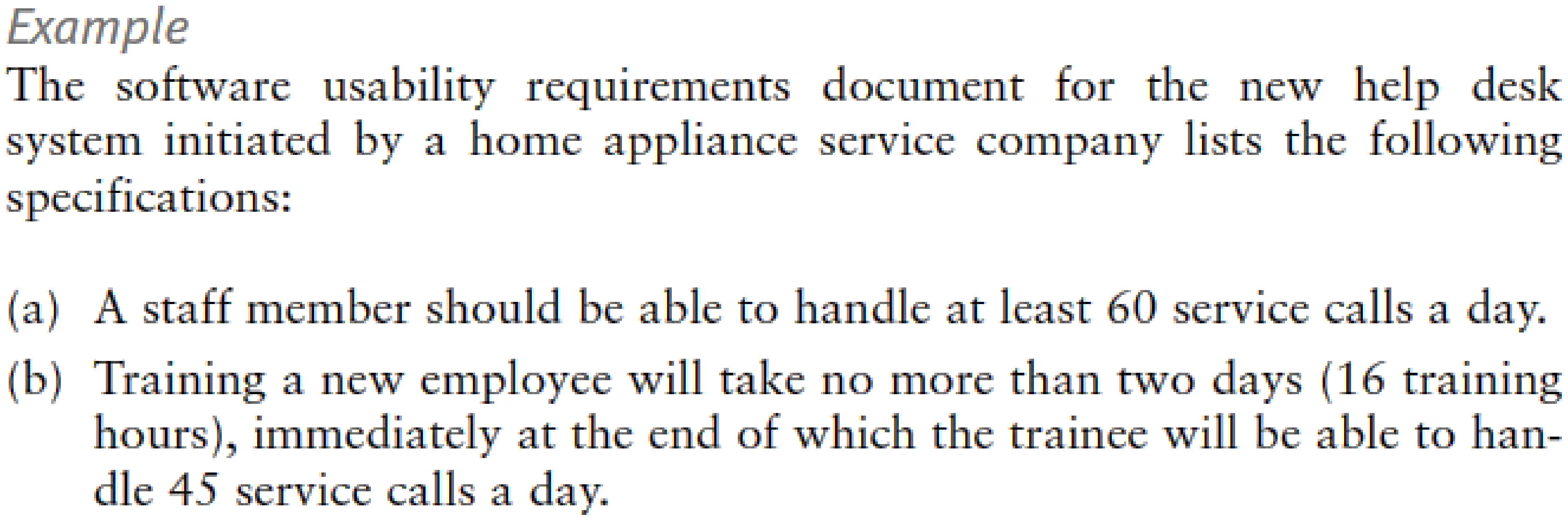


Example

The Engineering Department of a local municipality operates a GIS (Geographic Information System). The Department is planning to allow citizens access to its GIS files through the Internet. The software requirements include the possibility of viewing and copying but not inserting changes in the maps of their assets as well as any other asset in the municipality's area ("read only" permit). Access will be denied to plans in progress and to those maps defined by the Department's head as limited access documents.

Usability

Usability requirements deal with the scope of staff resources needed to train a new employee and to operate the software system.


Example

The software usability requirements document for the new help desk system initiated by a home appliance service company lists the following specifications:

(a) A staff member should be able to handle at least 60 service calls a day.

(b) Training a new employee will take no more than two days (16 training hours), immediately at the end of which the trainee will be able to handle 45 service calls a day.

Product revision software quality factors

Maintainability

Maintainability requirements determine the *efforts that will be needed by users and maintenance personnel to identify the reasons for software failures, to correct the failures, and to verify the success of the corrections*.

This factor’s requirements refer to the modular structure of software, the internal program documentation, and the programmer’s manual, among other items.

Example

Typical maintainability requirements:

(a) The size of a software module will not exceed 30 statements.

(b) The programming will adhere to the company coding standards and guidelines.

Flexibility

Flexibility in software means how easily it can be updated or adjusted with minimal effort. It includes adapting the software for different customers, scales of operation, or product ranges in the same industry with minimal resources like time or effort.

Example

TSS (teacher support software) deals with the documentation of pupil achievements, the calculation of final grades, the printing of term grade documents, and the automatic printing of warning letters to parents of failing pupils. The software specifications included the following flexibility requirements:

(a) The software should be suitable for teachers of all subjects and all school levels (elementary, junior and high schools).

(b) Non-professionals should be able to create new types of reports according to the schoolteacher's requirements and/or the city's education department demands.

Testability

Testability refers to how easily an information system can be tested during development and operation. Testability requirements focus on features that make testing easier, such as:

- Predefined intermediate results to verify specific parts of the system.
- Log files to track system behavior and identify issues.

These features help testers quickly find and fix problems.

Example

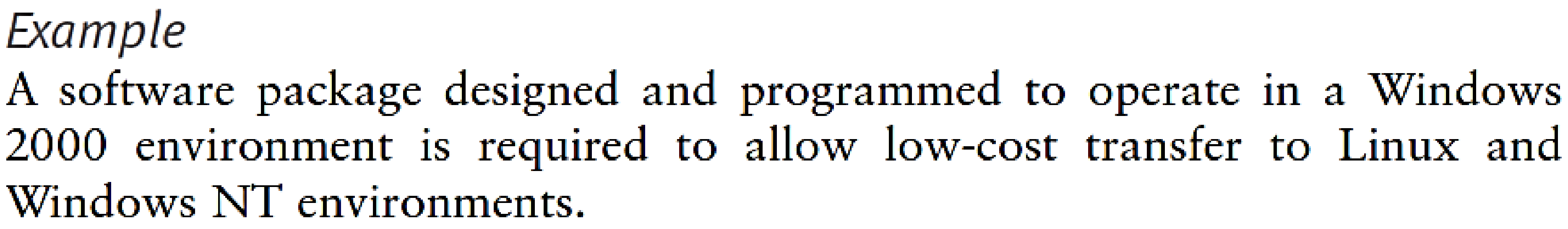
An industrial computerized control unit is programmed to calculate various measures of production status, report the performance level of the machinery, and operate a warning signal in predefined situations.

One testability requirement demanded was to develop a set of standard test data with known system expected correct reactions in each stage. This standard test data is to be run every morning, before production begins, to check whether the computerized unit reacts properly.

Product transition software quality factors

Portability

Portability requirements tend to the adaptation of a software system to other environments consisting of different hardware, different operating systems, and so forth.



Example

A software package designed and programmed to operate in a Windows 2000 environment is required to allow low-cost transfer to Linux and Windows NT environments.

Reusability

Reusability requirements deal with the use of software modules originally designed for one project in a new software project currently being developed.

They may also enable future projects to make use of a given module or a group of modules of the currently developed software.

The reuse of software is expected to save development resources, shorten the development period, and provide higher quality modules.

These benefits of higher quality are based on the assumption that most of the software faults have already been detected by the quality assurance activities performed on the original software, by users of the original software, and during its earlier reuses.

Example

A software development unit has been required to develop a software system for the operation and control of a hotel swimming pool that serves hotel guests and members of a pool club. Although the management did not define any reusability requirements, the unit's team leader, after analyzing the information processing requirements of the hotel's spa, decided to add the reusability requirement that some of the software modules for the pool should be designed and programmed in a way that will allow its reuse in the spa's future software system, which is planned to be developed next year.

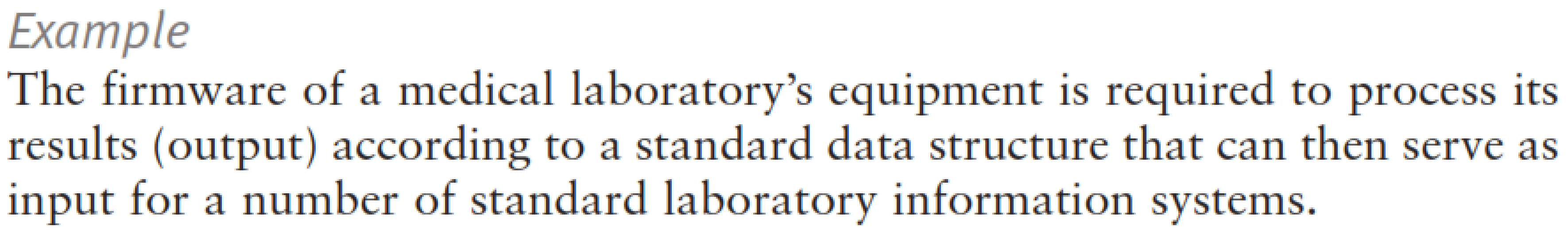
These modules will allow:

- Entrance validity checks of membership cards and visit recording.
- Restaurant billing.
- Processing of membership renewal letters.

Interoperability

Interoperability requirements focus on creating interfaces with other software systems or with other equipment firmware.

Interoperability requirements can specify the name(s) of the software or firmware for which interface is required.



Example

The firmware of a medical laboratory's equipment is required to process its results (output) according to a standard data structure that can then serve as input for a number of standard laboratory information systems.

Alternative models of software quality factors

- The Evans and Marciniak factor model (Evans and Marciniak, 1987).
- The Deutsch and Willis factor model (Deutsch and Willis, 1988).

Comparison of the alternative models

Both alternative models exclude only one of McCall’s 11 factors, namely the testability factor.

- The Evans and Marciniak factor model consists of 12 factors that are classified into three categories.
- The Deutsch and Willis factor model consists of 15 factors that are classified into four categories.

Taken together, five new factors were suggested by the two alternative factor models.

- Verifiability (by both models)
- Expandability (by both models)
- Safety (by Deutsch and Willis)
- Manageability (by Deutsch and Willis)
- Survivability (by Deutsch and Willis).

Table 3.1: Comparison of McCall's factor model and alternative models

No.	Software quality factor	McCall's classic model	Alternative factor models: Evans and Marciniak	Alternative factor models: Deutsch and Willis
1	Correctness	+	+	+
2	Reliability	+	+	+
3	Efficiency	+	+	+
4	Integrity	+	+	+
5	Usability	+	+	+
6	Maintainability	+	+	+
7	Flexibility	+	+	+
8	Testability	+		
9	Portability	+	+	+
10	Reusability	+	+	+
11	Interoperability	+	+	+
12	Verifiability		+	+
13	Expandability		+	+
14	Safety			+
15	Manageability			+
16	Survivability			+

Software testing (or “testing”) was the first software quality assurance tool applied to control the software product’s quality before its shipment or installation at the customer’s premises.

SQA professionals were encouraged to extend testing to the partial in-process products of coding, which led to software module (unit) testing and integration testing.

Definition

"Testing is the process of executing a program with intention of finding errors."

Software testing is a formal process carried out by a specialized testing team in which a software unit, several integrated software units or an entire software package are examined by running the programs on a computer.

All the associated tests are performed according to approved test procedures on approved test cases.

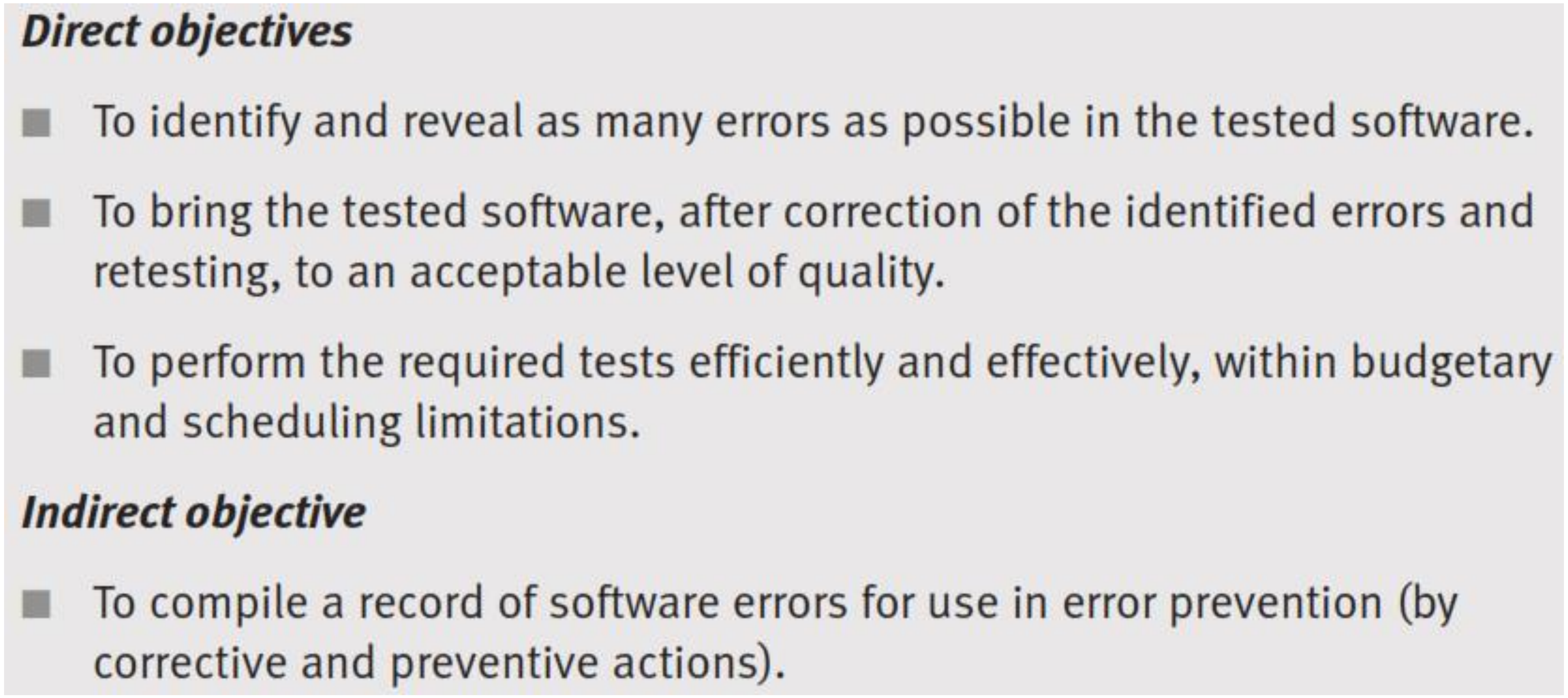
Formal – Software test plans are part of the project's development and quality plans, scheduled in advance and often a central item in the development agreement signed between the customer and the developer.

Specialized testing team – An independent team or external consultants who specialize in testing are assigned to perform these tasks mainly in order to eliminate bias and to guarantee effective testing by trained professionals.

Running the programs – Any form of quality assurance activity that does not involve running the software, for example code inspection, cannot be considered as a test.

- **Approved test procedures** – The testing process performed according to a test plan and testing procedures that have been approved as conforming to the SQA procedures adopted by the developing organization.
- **Approved test cases** – The test cases to be examined are defined in full by the test plan. No omissions or additions are expected to occur during testing.

Software testing objectives



Direct objectives

- To identify and reveal as many errors as possible in the tested software.
- To bring the tested software, after correction of the identified errors and retesting, to an acceptable level of quality.
- To perform the required tests efficiently and effectively, within budgetary and scheduling limitations.

Indirect objective

- To compile a record of software errors for use in error prevention (by corrective and preventive actions).

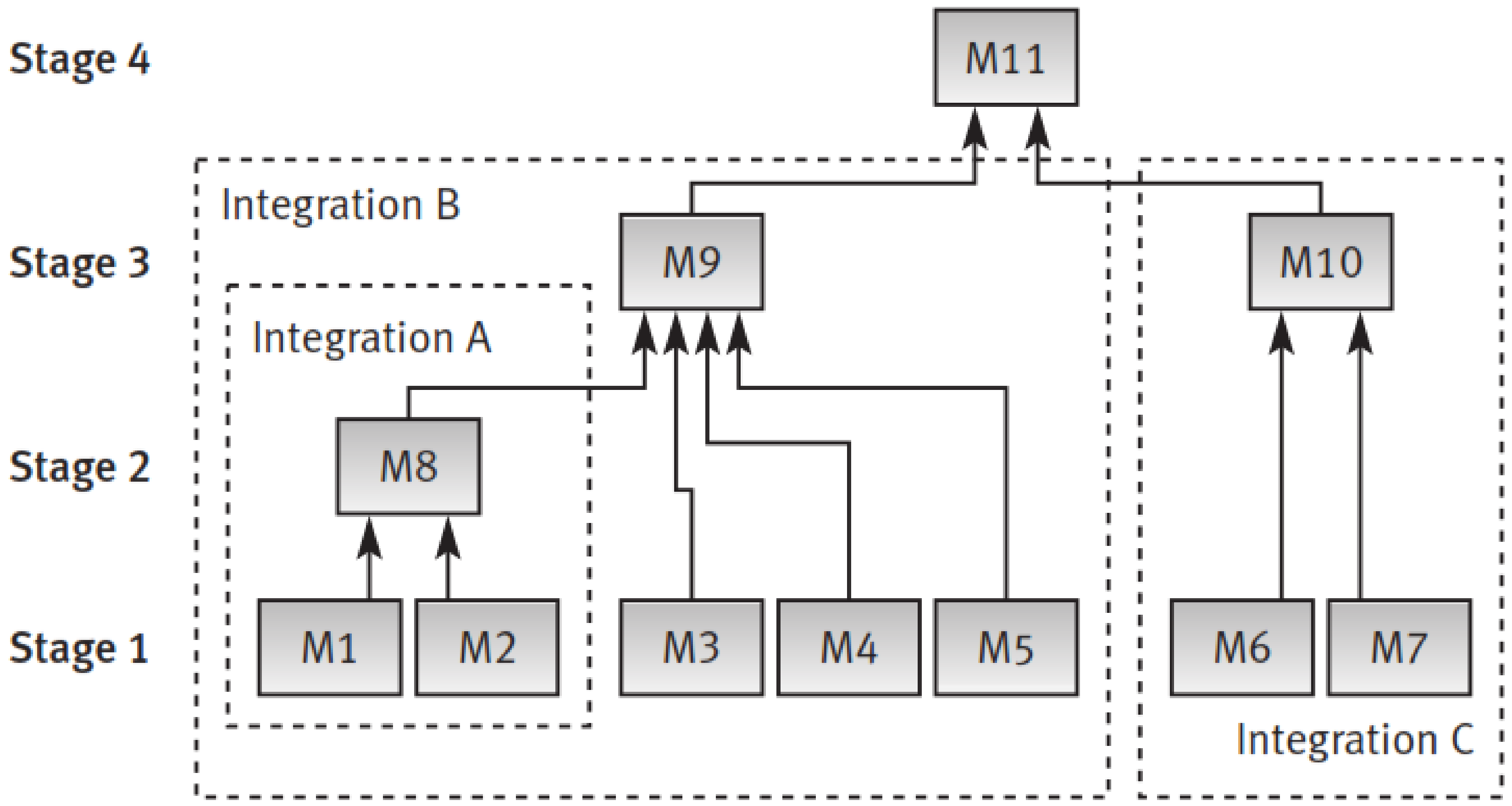
Software testing strategies

To test the software in its entirety, once the completed package is available; known as "big bang testing".

To test the software piecemeal, in modules, as they are completed (unit tests); then to test groups of tested modules integrated with newly completed modules (integration tests). This process continues until all the Package modules have been tested. Once this phase is completed, the entire package is tested as a whole (system test). This testing strategy is usually termed "incremental testing".

Incremental testing is also performed according to two basic strategies: bottom-up and top-down.

Both incremental testing strategies assume that the software package is constructed of a hierarchy of software modules.



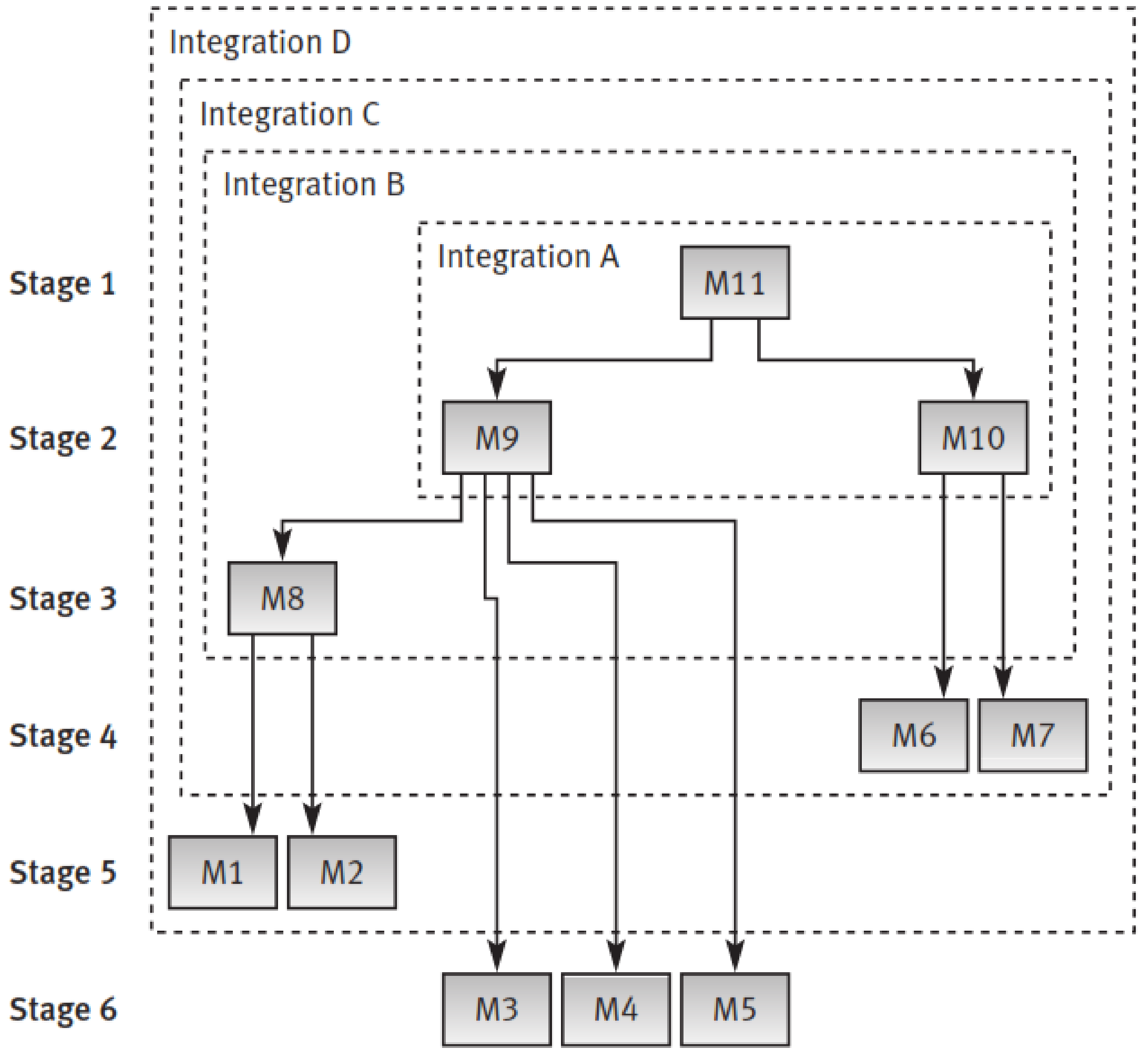
(a) Bottom-up testing

Stage 1: Unit tests of modules 1 to 7.

Stage 2: Integration test A of modules 1 and 2, developed and tested in stage 1, and integrated with module 8, developed in the current stage.

Stage 3: Two separate integration tests, B, on modules 3, 4, 5 and 8, integrated with module 9, and C, for modules 6 and 7, integrated with module 10.

Stage 4: System test is performed after B and C have been integrated with module 11, developed in the current stage.



(b) Top-down testing

Stage 1: Unit tests of module 11.

Stage 2: Integration test A of module 11 integrated with modules 9 and 10, developed in the current stage.

Stage 3: Integration test B of A integrated with module 8, developed in the current stage.

Stage 4: Integration test C of B integrated with modules 6 and 7, developed in the current stage.

Stage 5: Integration test D of C integrated with modules 1 and 2, developed in the current stage.

Stage 6: System test of D integrated with modules 3, 4 and 5, developed in the current stage.

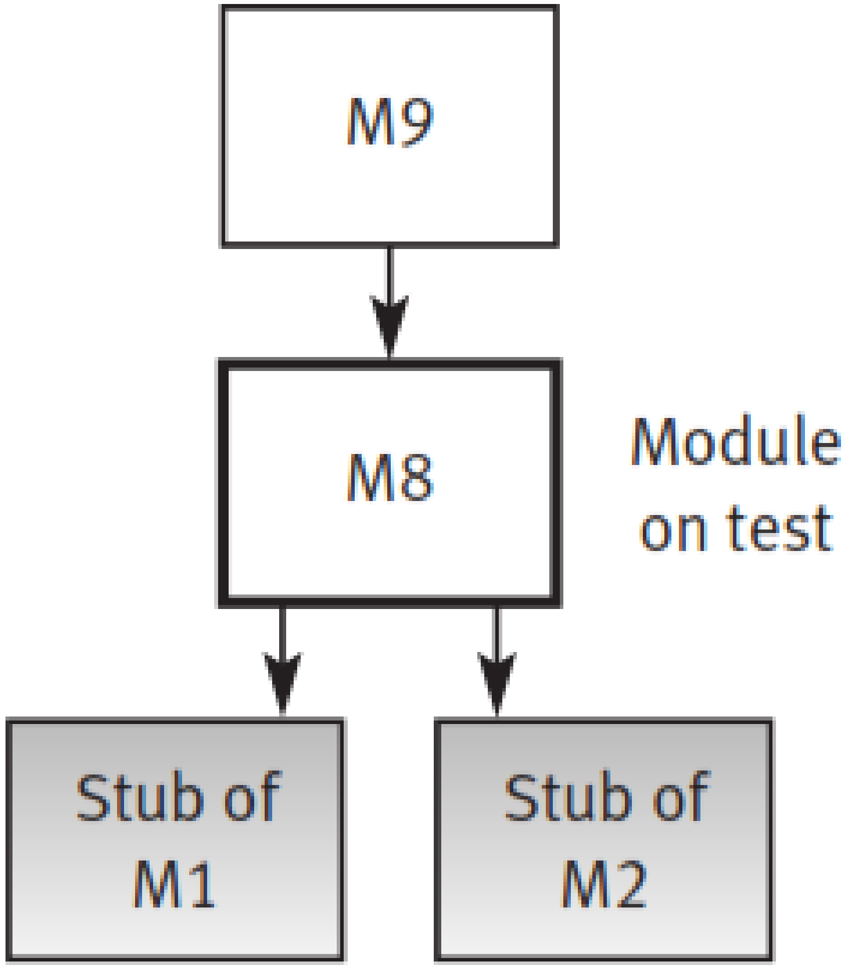
Stubs and drivers for incremental testing

Stubs and drivers are software replacement simulators required for modules not available when performing a unit or an integration test.

A stub (often termed a “dummy module”) replaces an unavailable lower level module, subordinate to the module tested.

Stubs are required for topdown testing of incomplete systems. In this case, the stub provides the results of calculations the subordinate module, yet to be developed (coded), is designed to perform.

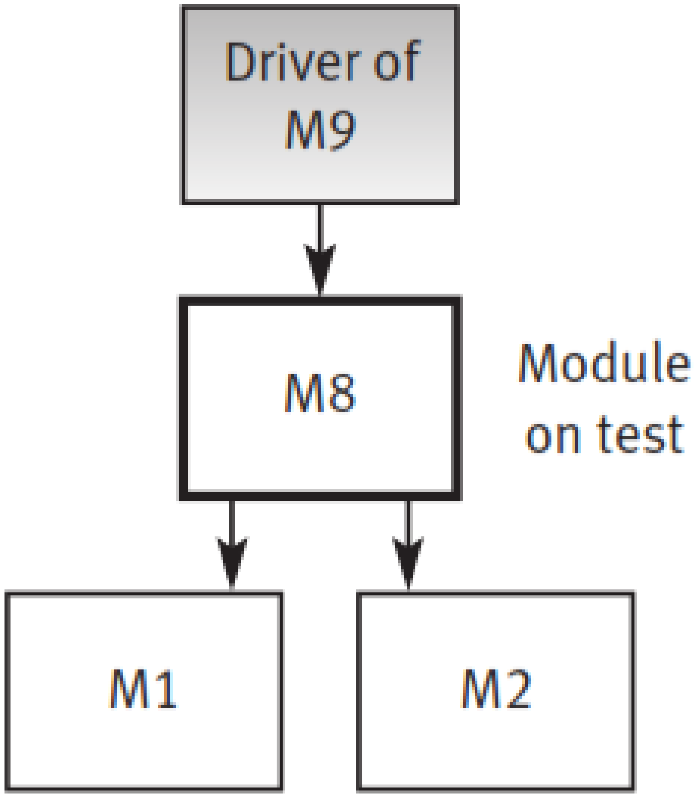
(a) Implementing top-down tests (Stage 3 testing of the example shown in Figure 9.1)



A driver is a substitute module but of the upper level module that activates the module tested. The driver is passing the test data on to the tested module and accepting the results calculated by it.

Drivers are Required in bottom-up testing until the upper level modules are developed (coded).

(b) Implementing bottom-up tests (Stage 2 testing of the example shown in Figure 9.1)



THE TESTING PROCESS

Testing is different from debugging.

- Removing errors from your programs is known as *debugging but testing aims to locate as yet undiscovered errors.*
- Starts from the requirements analysis phase and goes until the last maintenance phase.

Static testing: Requirement analysis and designing stage.

SRS is tested to check whether it is as per user requirements or not. We use techniques of code reviews, code inspections, walkthroughs, and software technical reviews (STRs) to do static testing

Dynamic Testing: This begins when the code or a specific unit/module is ready. It involves executing the code to validate its functionality. Common techniques for dynamic testing include black-box testing (focusing on inputs and outputs without knowing internal details), gray-box testing (a mix of internal knowledge and external testing), and white-box testing (testing internal structures and logic)..

Five distinct levels of testing

a. Debug: It is defined as the successful correction of a failure.

b. Demonstrate: The process of showing that major features work with typical input.

c. Verify: The process of finding as many faults in the application under test (AUT) as possible.

d. Validate: The process of finding as many faults in requirements, design, and AUT.

e. Prevent: To avoid errors in development of requirements, design, and implementation by self-checking techniques, including “test before design.”

Popular equation of software testing

Software Testing = Software Verification + Software Validation

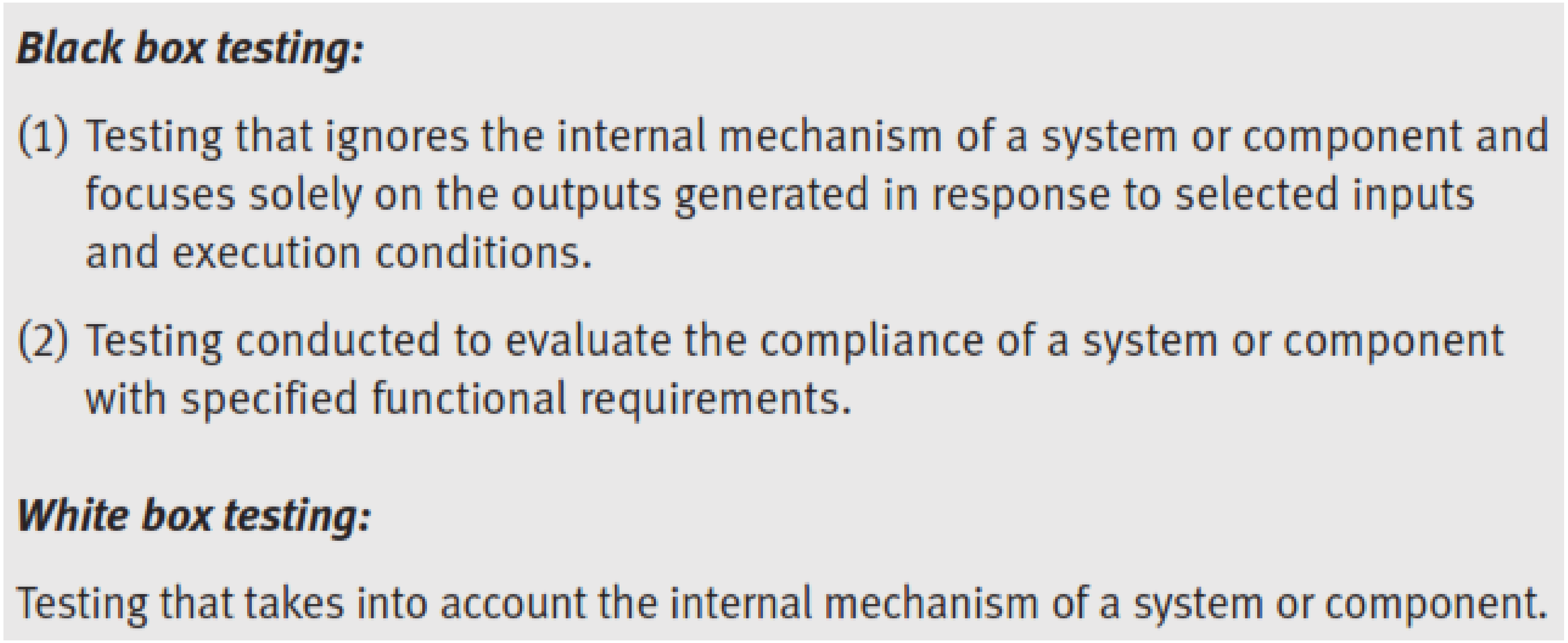
- Software Verification "It is the process of evaluating, reviewing, inspecting and doing desk checks of work products such as requirement specifications, design specifications and code."
- Verification means **Are we building the product right?**

software validation

- “It is defined as the process of evaluating a system or component during or at the end of development process to determine whether it satisfies the specified requirements. It involves executing the actual software. It is a computer based testing process.”
- Validation means **Are we building the right product?**
- Both verification and validation (V&V) are complementary to each other.

Software test classifications

Classification according to testing concept



Black box testing:

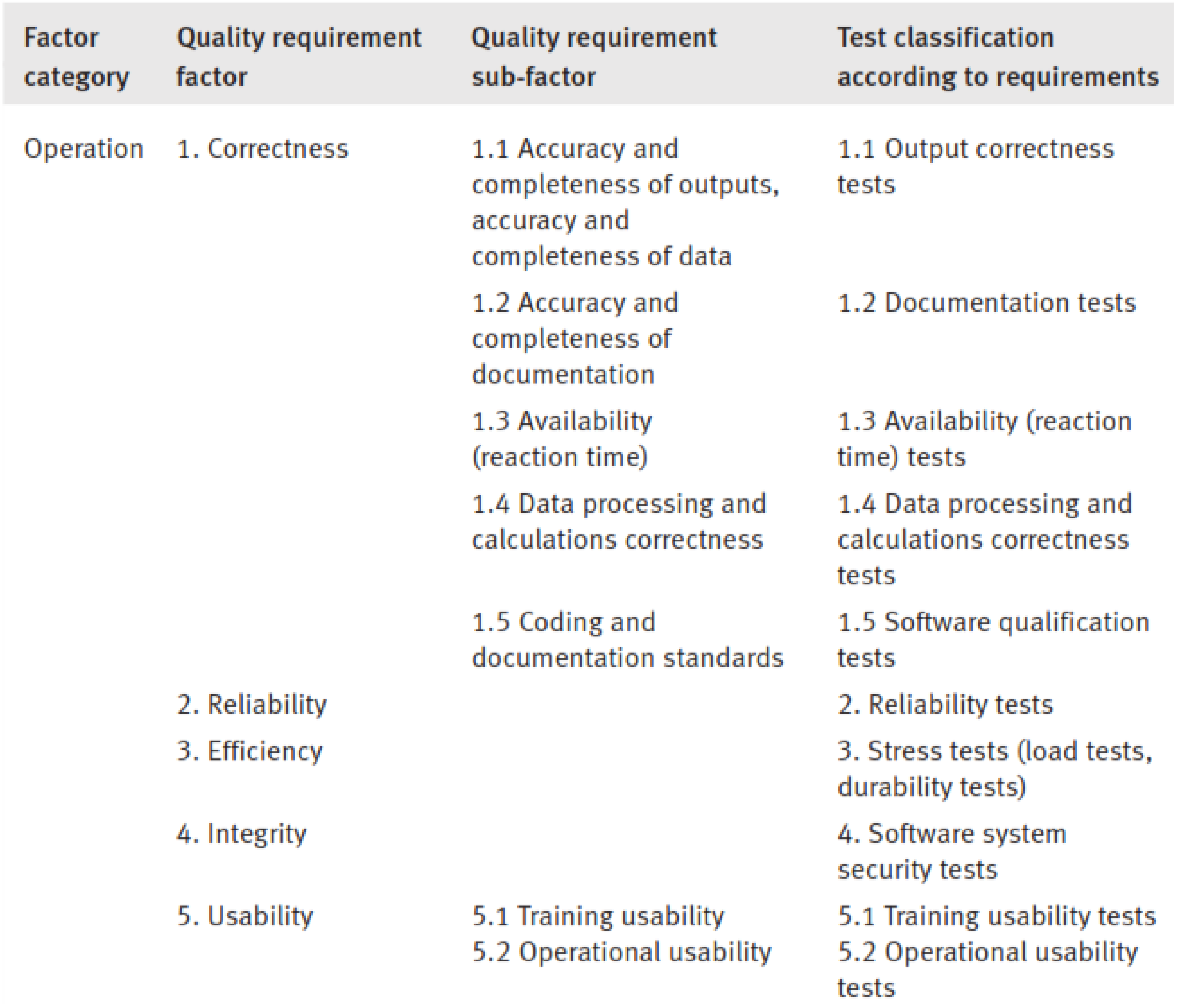
(1) Testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions.

(2) Testing conducted to evaluate the compliance of a system or component with specified functional requirements.

White box testing:

Testing that takes into account the internal mechanism of a system or component.

Classification according to requirements



Factor category	Quality requirement factor	Quality requirement sub-factor	Test classification according to requirements
Operation	1. Correctness	1.1 Accuracy and completeness of outputs, accuracy and completeness of data	1.1 Output correctness tests
		1.2 Accuracy and completeness of documentation	1.2 Documentation tests
		1.3 Availability (reaction time)	1.3 Availability (reaction time) tests
		1.4 Data processing and calculations correctness	1.4 Data processing and calculations correctness tests
		1.5 Coding and documentation standards	1.5 Software qualification tests
	2. Reliability		2. Reliability tests
	3. Efficiency		3. Stress tests (load tests, durability tests)
	4. Integrity		4. Software system security tests
	5. Usability	5.1 Training usability 5.2 Operational usability	5.1 Training usability tests 5.2 Operational usability tests

Revision	6. Maintainability		6. Maintainability tests
	7. Flexibility		7. Flexibility tests
	8. Testability		8. Testability tests
Transition	9. Portability		9. Portability tests
	10. Reusability		10. Reusability tests
	11. Interoperability	11.1 Interoperability with other software	11.1 Software interoperability tests
		11.2 Interoperability with other equipment	11.2 Equipment interoperability tests

THANK YOU