**Definitions**

**A**

**Abstract test case:** See *high level test case*.

**Acceptance:** See *acceptance testing*.

**Acceptance criteria:** The exit criteria that a component or system must satisfy in order to be

accepted by a user, customer, or other authorized entity. [IEEE 610]

**Acceptance testing:** Formal testing with respect to user needs, requirements, and business

processes conducted to determine whether or not a system satisfies the acceptance criteria

and to enable the user, customers or other authorized entity to determine whether or not to

accept the system. [After IEEE 610]

**Accessibility testing:** Testing to determine the ease by which users with disabilities can use a

component or system.

**Accuracy:** The capability of the software product to provide the right or agreed results or effects with the needed degree of precision. [ISO 9126] See also *functionality testing*.

**Action word driven testing:** See *keyword driven testing*

**Actual outcome:** See *actual result*.

**Actual result:** The behavior produced/observed when a component or system is tested.

**Ad hoc review:** See *informal review*.

**Ad hoc testing:** Testing carried out informally; no formal test preparation takes place, no

recognized test design technique is used, there are no expectations for results and

arbitrariness guides the test execution activity.

**Adaptability:** The capability of the software product to be adapted for different specified

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

for the software considered. [ISO 9126] See also *portability*.

**Agile testing:** Testing practice for a project using agile methodologies, such as extreme

programming (XP), treating development as the customer of testing and emphasizing the

test-first design paradigm. See also *test driven development*.

**Algorithm test [TMap]:** See *branch testing*.

**Alpha testing:** Simulated or actual operational testing by potential users/customers or an

independent test team at the developers’ site, but outside the development organization.

Alpha testing is often employed for off-the-shelf software as a form of internal acceptance

testing.

**Analyzability:** The capability of the software product to be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified to be identified. [ISO 9126] See also *maintainability*.

**Analyzer:** See *static analyzer*.

**Anomaly:** Any condition that deviates from expectation based on requirements specifications, design documents, user documents, standards, etc. or from someone’s perception or experience. Anomalies may be found during, but not limited to, reviewing, testing, analysis, compilation, or use of software products or applicable documentation. [IEEE 1044] See also *bug, defect, deviation, error, fault, failure, incident, and problem*.

**Arc testing:** See *branch testing*.

**Attack:** Directed and focused attempt to evaluate the quality, especially reliability, of a test

object by attempting to force specific failures to occur.

**Attractiveness:** The capability of the software product to be attractive to the user. [ISO 9126] See also *usability*.

**Audit:** An independent evaluation of software products or processes to ascertain compliance to standards, guidelines, specifications, and/or procedures based on objective criteria, including documents that specify:

(1) The form or content of the products to be produced

(2) The process by which the products shall be produced

(3) How compliance to standards or guidelines shall be measured. [IEEE 1028]

**Audit trail:** A path by which the original input to a process (e.g. data) can be traced back

through the process, taking the process output as a starting point. This facilitates defect

analysis and allows a process audit to be carried out. [After TMap]

**Automated test ware:** Test ware used in automated testing, such as tool scripts.

**Availability:** The degree to which a component or system is operational and accessible when

required for use. Often expressed as a percentage. [IEEE 610]

**B**

**Back-to-back testing:** Testing in which two or more variants of a component or system are

executed with the same inputs, the outputs compared, and analyzed in cases of

discrepancies. [IEEE 610]

**Baseline:** A specification or software product that has been formally reviewed or agreed upon, that thereafter serves as the basis for further development, and that can be changed only through a formal change control process. [After IEEE 610]

**Basic block:** A sequence of one or more consecutive executable statements containing no

branches. Note: A node in a control flow graph represents a basic block.

**Basis test set:** A set of test cases derived from the internal structure of a component or

specification to ensure that 100% of a specified coverage criterion will be achieved.

**Behavior:** The response of a component or system to a set of input values and preconditions.

**Benchmark test:** (1) A standard against which measurements or comparisons can be made.

(2) A test that is be used to compare components or systems to each other or to a standard

as in (1). [After IEEE 610]

**Bespoke software:** Software developed specifically for a set of users or customers. The

opposite is off-the-shelf software.

**Best practice:** A superior method or innovative practice that contributes to the improved

performance of an organization under given context, usually recognized as ‘best’ by other

peer organizations.

**Beta testing:** Operational testing by potential and/or existing users/customers at an external

site not otherwise involved with the developers, to determine whether or not a component

or system satisfies the user/customer needs and fits within the business processes. Beta

testing is often employed as a form of external acceptance testing for off-the-shelf software

in order to acquire feedback from the market.

**Big-bang testing:** A type of integration testing in which software elements, hardware

elements, or both are combined all at once into a component or an overall system, rather

than in stages. [After IEEE 610] See also *integration testing*.

**Black-box technique:** See *black box test design technique*.

**Black-box testing:** Testing, either functional or non-functional, without reference to the

internal structure of the component or system.

**Black-box test design technique:** Procedure to derive and/or select test cases based on an

analysis of the specification, either functional or non-functional, of a component or system

without reference to its internal structure.

**Blocked test case:** A test case that cannot be executed because the preconditions for its

execution is not fulfilled.

**Bottom-up testing:** An incremental approach to integration testing where the lowest level

components are tested first, and then used to facilitate the testing of higher level

components. This process is repeated until the component at the top of the hierarchy is

tested. See also *integration testing*.

**Boundary value:** An input value or output value which is on the edge of equivalence

partition or at the smallest incremental distance on either side of an edge, for example the

minimum or maximum value of a range.

**Boundary value analysis:** A black box test design technique in which test cases are designed

based on boundary values. See also *boundary value.*

**Boundary value coverage:** The percentage of boundary values that have been exercised by a

test suite.

**Boundary value testing:** See *boundary value analysis*.

**Branch:** A basic block that can be selected for execution based on a program construct in

which one of two or more alternative program paths is available, e.g. case, jump, go to, if then-else.

**Branch condition:** See *condition*.

**Branch condition combination coverage:** See *multiple condition coverage*.

**Branch condition combination testing:** See *multiple condition testing*.

**Branch condition coverage:** See *condition coverage*.

**Branch coverage:** The percentage of branches that have been exercised by a test suite. 100%

branch coverage implies both 100% decision coverage and 100% statement coverage.

**Branch testing:** A white box test design technique in which test cases are designed to execute branches.

**Buffer:** A device or storage area used to store data temporarily for differences in rates of data flow, time or occurrence of events, or amounts of data that can be handled by the devices or processes involved in the transfer or use of the data. [IEEE 610]

**Buffer overflow:** A memory access defect due to the attempt by a process to store data

beyond the boundaries of a fixed length buffer, resulting in overwriting of adjacent

memory areas or the raising of an overflow exception. See also *buffer.*

**Bug:** See *defect*.

**Bug report:** See *defect report.*

**Bug taxonomy:** *See defect taxonomy.*

**Bug tracking tool:** See *defect management tool.*

**Business process-based testing:** An approach to testing in which test cases are designed

based on descriptions and/or knowledge of business processes.

**C**

**Capability Maturity Model (CMM):** A five level staged framework that describes the key

elements of an effective software process. The Capability Maturity Model covers best practices for planning, engineering and managing software development and maintenance.

[CMM] See also *Capability Maturity Model Integration (CMMI).*

**Capability Maturity Model Integration (CMMI):** A framework that describes the key

elements of an effective product development and maintenance process. The Capability

Maturity Model Integration covers best-practices for planning, engineering and managing

product development and maintenance. CMMI is the designated successor of the CMM.

[CMMI] See also *Capability Maturity Model (CMM).*

**Capture/playback tool:** A type of test execution tool where inputs are recorded during

manual testing in order to generate automated test scripts that can be executed later (i.e.

replayed). These tools are often used to support automated regression testing.

**Capture/replay tool:** See *capture/playback tool*.

**CASE:** Acronym for Computer Aided Software Engineering.

**CAST:** Acronym for Computer Aided Software Testing. See also *test automation*.

**Cause-effect graph:** A graphical representation of inputs and/or stimuli (causes) with their

associated outputs (effects), which can be used to design test cases.

**Cause-effect graphing:** A black box test design technique in which test cases are designed

from cause-effect graphs. [BS 7925/2]

**Cause-effect analysis:** See *cause-effect graphing*.

**Cause-effect decision table:** See *decision table*.

**Certification:** The process of confirming that a component, system or person complies with

its specified requirements, e.g. by passing an exam.

**Changeability:** The capability of the software product to enable specified modifications to be implemented. [ISO 9126] See also *maintainability*.

**Change control:** See *configuration control.*

**Change control board:** See *configuration control board*.

**Checker:** See *reviewer*.

**Chow's coverage metrics:** See *N-switch coverage*.

**Classification tree**: A tree showing equivalence partitions hierarchically ordered, which is

used to design test cases in the classification tree method. See also *classification tree*

*method.*

**Classification tree method:** A black box test design technique in which test cases, described

by means of a classification tree, are designed to execute combinations of representatives

of input and/or output domains.

**Code:** Computer instructions and data definitions expressed in a programming language or in a form output by an assembler, compiler or other translator. [IEEE 610]

**Code analyzer:** See *static code analyzer*.

**Code coverage:** An analysis method that determines which parts of the software have been

executed (covered) by the test suite and which parts have not been executed, e.g. statement coverage, decision coverage or condition coverage.

**Code-based testing:** See *white box testing*.

**Co-existence:** The capability of the software product to co-exist with other independent

software in a common environment sharing common resources. [ISO 9126] See also

*portability.*

**Commercial off-the-shelf software**: See *off-the-shelf software*.

**Comparator:** See *test comparator*.

**Compatibility testing:** See *interoperability testing*.

**Compiler:** A software tool that translates programs expressed in a high order language into

their machine language equivalents. [IEEE 610]

**Complete testing:** See *exhaustive testing*.

**Completion criteria:** See *exit criteria*.

**Complexity:** The degree to which a component or system has a design and/or internal

structure that is difficult to understand, maintain and verify. See also C*yclomatic*

*Complexity.*

**Compliance:** The capability of the software product to adhere to standards, conventions or

regulations in laws and similar prescriptions. [ISO 9126]

**Compliance testing:** The process of testing to determine the compliance of the component or system.

**Component:** A minimal software item that can be tested in isolation.

**Component integration testing:** Testing performed to expose defects in the interfaces and

interaction between integrated components.

**Component specification:** A description of a component’s function in terms of its output

values for specified input values under specified conditions, and required non-functional

behavior (e.g. resource-utilization).

**Component testing:** The testing of individual software components. [After IEEE 610]

**Compound condition:** Two or more single conditions joined by means of a logical operator

(AND, OR or XOR), e.g. ‘A>B AND C>1000’.

**Concrete test case:** See *low level test case*.

**Concurrency testing:** Testing to determine how the occurrence of two or more activities

within the same interval of time, achieved either by interleaving the activities or by

simultaneous execution is handled by the component or system. [After IEEE 610]

**Condition:** A logical expression that can be evaluated as True or False, e.g. A>B. See also *test condition.*

**Condition combination coverage:** See *multiple condition coverage*.

**Condition combination testing:** See *multiple condition testing*.

**Condition coverage:** The percentage of condition outcomes that have been exercised by a test suite. 100% condition coverage requires each single condition in every decision statement to be tested as True and False.

**Condition determination coverage:** The percentage of all single condition outcomes that

independently affect a decision outcome that has been exercised by a test case suite.

100% condition determination coverage implies 100% decision condition coverage.

**Condition determination testing:** A white box test design technique in which test cases are

designed to execute single condition outcomes that independently affect a decision

outcome.

**Condition testing:** A white box test design technique in which test cases are designed to

execute condition outcomes.

**Condition outcome:** The evaluation of a condition to True or False.

**Confidence test:** See *smoke test*.

**Configuration:** The composition of a component or system as defined by the number, nature and interconnections of its constituent parts.

**Configuration auditing:** The function to check on the contents of libraries of configuration

items, e.g. for standards compliance. [IEEE 610]

**Configuration control:** An element of configuration management, consisting of the

evaluation, co-ordination, approval or disapproval, and implementation of changes to

configuration items after formal establishment of their configuration identification.

**Configuration control board (CCB):** A group of people responsible for evaluating and

approving or disapproving proposed changes to configuration items, and for ensuring

implementation of approved changes. [IEEE 610]

**Configuration identification:** An element of configuration management, consisting of

selecting the configuration items for a system and recording their functional and physical

characteristics in technical documentation. [IEEE 610]

**Configuration item:** An aggregation of hardware, software or both, that is designated for

configuration management and treated as a single entity in the configuration management

process. [IEEE 610]

**Configuration management:** A discipline applying technical and administrative direction and

surveillance to: identify and document the functional and physical characteristics of a

configuration item, control changes to those characteristics, record and report change

processing and implementation status, and verify compliance with specified requirements.

**Configuration management tool:** A tool that provides support for the identification and

control of configuration items, their status over changes and versions, and the release of

baselines consisting of configuration items.

**Configuration testing:** See *portability testing*.

**Confirmation testing:** See *re-testing*.

**Conformance testing:** See *compliance testing*.

**Consistency:** The degree of uniformity, standardization, and freedom from contradiction

among the documents or parts of a component or system. [IEEE 610]

**Control flow:** A sequence of events (paths) in the execution through a component or system.

**Control flow analysis:** A form of static analysis based on a representation of sequences of

events (paths) in the execution through a component or system.

**Control flow graph:** An abstract representation of all possible sequences of events (paths) in the execution through a component or system.

**Control flow path:** See *path*.

**Continuous representation:** A capability maturity model structure wherein capability levels

provide a recommended order for approaching process improvement within specified

process areas. [CMMI]

**Conversion testing:** Testing of software used to convert data from existing systems for use in replacement systems.

**Cost of quality**: The total costs incurred on quality activities and issues and often split into

prevention costs, appraisal costs, internal failure costs and external failure costs.

**COTS:** Acronym for Commercial Off-The-Shelf software. See *off-the-shelf software.*

**Coverage:** The degree, expressed as a percentage, to which a specified coverage item has been exercised by a test suite.

**Coverage analysis:** Measurement of achieved coverage to a specified coverage item during

test execution referring to predetermined criteria to determine whether additional testing is

required and if so, which test cases are needed.

**Coverage measurement tool:** See *coverage tool.*

**Coverage item:** An entity or property used as a basis for test coverage, e.g. equivalence

partitions or code statements.

**Coverage tool:** A tool that provides objective measures of what structural elements, e.g.

statements, branches have been exercised by a test suite.

**Custom software:** See *bespoke software*.

**Cyclomatic complexity:** The number of independent paths through a program. Cyclomatic

complexity is defined as: L – N + 2P, where

- L = the number of edges/links in a graph

- N = the number of nodes in a graph

- P = the number of disconnected parts of the graph (e.g. a called graph and a subroutine)

[After McCabe]

**Cyclomatic number:** See *cyclomatic complexity*.

**D**

**Daily build:** a development activity where a complete system is compiled and linked every

day (usually overnight), so that a consistent system is available at any time including all

latest changes.

**Data definition:** An executable statement where a variable is assigned a value.

**Data driven testing:** A scripting technique that stores test input and expected results in a table or spreadsheet, so that a single control script can execute all of the tests in the table. Data driven testing is often used to support the application of test execution tools such as

Capture/playback tools. [Fewster and Graham] See also *keyword driven testing*.

**Data flow:** An abstract representation of the sequence and possible changes of the state of

data objects, where the state of an object is any of: creation, usage, or destruction.

**Data flow analysis:** A form of static analysis based on the definition and usage of variables.

**Data flow coverage:** The percentage of definition-use pairs that have been exercised by a test suite.

**Data flow testing:** A white box test design technique in which test cases are designed to

execute definition and use pairs of variables.

**Data integrity testing:** See *database integrity testing*.

**Database integrity testing:** Testing the methods and processes used to access and manage the data(base), to ensure access methods, processes and data rules function as expected and that during access to the database, data is not corrupted or unexpectedly deleted, updated or created.

**Dead code:** See *unreachable code*.

**Debugger:** See *debugging tool*.

**Debugging:** The process of finding, analyzing and removing the causes of failures in

software.

**Debugging tool:** A tool used by programmers to reproduce failures, investigate the state of

programs and find the corresponding defect. Debuggers enable programmers to execute

programs step by step, to halt a program at any program statement and to set and examine

program variables.

**Decision:** A program point at which the control flow has two or more alternative routes. A

node with two or more links to separate branches.

**Decision condition coverage:** The percentage of all condition outcomes and decision

outcomes that have been exercised by a test suite. 100% decision condition coverage

implies both 100% condition coverage and 100% decision coverage.

**Decision condition testing:** A white box test design technique in which test cases are

designed to execute condition outcomes and decision outcomes.

**Decision coverage:** The percentage of decision outcomes that have been exercised by a test

suite. 100% decision coverage implies both 100% branch coverage and 100% statement

coverage.

**Decision outcome:** The result of a decision (which therefore determines the branches to be

taken).

**Decision table:** A table showing combinations of inputs and/or stimuli (causes) with their

associated outputs and/or actions (effects), which can be used to design test cases.

**Decision table testing:** A black box test design technique in which test cases are designed to

execute the combinations of inputs and/or stimuli (causes) shown in a decision table.

**Decision testing:** A white box test design technique in which test cases are designed to

execute decision outcomes.

**Defect:** A flaw in a component or system that can cause the component or system to fail to

perform its required function, e.g. an incorrect statement or data definition. A defect, if

encountered during execution, may cause a failure of the component or system.

**Defect based technique:** *See* defect based test design technique.

**Defect based test design technique:** A procedure to derive and/or select test cases targeted at one or more defect categories, with tests being developed from what is known about the

specific defect category. See also *defect taxonomy.*

**Defect density:** The number of defects identified in a component or system divided by the

size of the component or system (expressed in standard measurement terms, e.g. lines-of code, number of classes or function points).

**Defect Detection Percentage (DDP):** The number of defects found by a test phase, divided

by the number found by that test phase and any other means afterwards.

**Defect management:** The process of recognizing, investigating, taking action and disposing

of defects. It involves recording defects, classifying them and identifying the impact.

[After IEEE 1044]

**Defect management tool:** A tool that facilitates the recording and status tracking of defects

and changes. They often have workflow-oriented facilities to track and control the

allocation, correction and re-testing of defects and provide reporting facilities. See also

*incident management tool*.

**Defect masking:** An occurrence in which one defect prevents the detection of another.

**Defect report:** A document reporting on any flaw in a component or system that can cause the component or system to fail to perform its required function. [After IEEE 829]

**Defect taxonomy:** A system of (hierarchical) categories designed to be a useful aid for

reproducibly classifying defects.

**Defect tracking tool:** See *defect management tool*.

**Definition-use pair:** The association of the definition of a variable with the use of that

variable. Variable uses include computational (e.g. multiplication) or to direct the

execution of a path (“predicate” use).

**Deliverable:** Any (work) product that must be delivered to someone other than the (work)

product’s author.

**Design-based testing:** An approach to testing in which test cases are designed based on the

architecture and/or detailed design of a component or system (e.g. tests of interfaces

between components or systems).

**Desk checking:** Testing of software or specification by manual simulation of its execution.

See also *static analysis.*

**Development testing:** Formal or informal testing conducted during the implementation of a

component or system, usually in the development environment by developers.

**Deviation:** See *incident*.

**Deviation report:** See *incident report*.

**Dirty testing:** See *negative testing*.

**Documentation testing:** Testing the quality of the documentation, e.g. user guide or

installation guide.

**Domain:** The set from which valid input and/or output values can be selected.

**Driver:** A software component or test tool that replaces a component that takes care of the

control and/or the calling of a component or system. [After TMap]

**Dynamic analysis:** The process of evaluating behavior, e.g. memory performance, CPU

usage, of a system or component during execution. [After IEEE 610]

**Dynamic analysis tool:** A tool that provides run-time information on the state of the software code. These tools are most commonly used to identify unassigned pointers, check pointer arithmetic and to monitor the allocation, use and de-allocation of memory and to flag memory leaks.

**Dynamic comparison:** Comparison of actual and expected results, performed while the

software is being executed, for example by a test execution tool.

**Dynamic testing:** Testing that involves the execution of the software of a component or

system.

**E**

**Efficiency:** The capability of the software product to provide appropriate performance,

relative to the amount of resources used under stated conditions. [ISO 9126]

**Efficiency testing:** The process of testing to determine the efficiency of a software product.

**Elementary comparison testing:** A black box test design technique in which test cases are

designed to execute combinations of inputs using the concept of condition determination

coverage. [TMap]

**Emulator:** A device, computer program, or system that accepts the same inputs and produces the same outputs as a given system. [IEEE 610] See also *simulator*.

**Entry criteria:** The set of generic and specific conditions for permitting a process to go

forward with a defined task, e.g. test phase. The purpose of entry criteria is to prevent a

task from starting which would entail more (wasted) effort compared to the effort needed

to remove the failed entry criteria. [Gilb and Graham]

**Entry point:** The first executable statement within a component.

**Equivalence class:** See *equivalence partition*.

**Equivalence partition:** A portion of an input or output domain for which the behavior of a

component or system is assumed to be the same, based on the specification.

**Equivalence partition coverage:** The percentage of equivalence partitions that have been

exercised by a test suite.

**Equivalence partitioning:** A black box test design technique in which test cases are designed

to execute representatives from equivalence partitions. In principle test cases are designed

to cover each partition at least once.

**Error:** A human action that produces an incorrect result. [After IEEE 610]

**Error guessing:** A test design technique where the experience of the tester is used to

anticipate what defects might be present in the component or system under test as a result

of errors made, and to design tests specifically to expose them.

**Error seeding:** See *fault seeding.*

**Error seeding tool:** See *fault seeding tool.*

**Error tolerance:** The ability of a system or component to continue normal operation despite

the presence of erroneous inputs. [After IEEE 610].

**Evaluation:** See *testing*.

**Exception handling:** Behavior of a component or system in response to erroneous input, from either a human user or from another component or system, or to an internal failure.

**Executable statement:** A statement which, when compiled, is translated into object code, and which will be executed procedurally when the program is running and may perform an

action on data.

**Exercised:** A program element is said to be exercised by a test case when the input value

causes the execution of that element, such as a statement, decision, or other structural

element.

**Exhaustive testing:** A test approach in which the test suite comprises all combinations of

input values and preconditions.

**Exit criteria:** The set of generic and specific conditions, agreed upon with the stakeholders,

for permitting a process to be officially completed. The purpose of exit criteria is to

prevent a task from being considered completed when there are still outstanding parts of

the task which have not been finished. Exit criteria are used to report against and to plan

when to stop testing. [After Gilb and Graham]

**Exit point:** The last executable statement within a component.

**Expected outcome:** See *expected result*.

**Expected result:** The behavior predicted by the specification, or another source, of the

component or system under specified conditions.

**Experienced-based technique:** See *experienced-based test design technique.*

**Experienced-based test design technique:** Procedure to derive and/or select test cases based on the tester’s experience, knowledge and intuition.

**Exploratory testing:** An informal test design technique where the tester actively controls the design of the tests as those tests are performed and uses information gained while testing to design new and better tests. [After Bach]

**F**

**Fail:** A test is deemed to fail if its actual result does not match its expected result.

**Failure:** Deviation of the component or system from its expected delivery, service or result.

**Failure mode:** The physical or functional manifestation of a failure. For example, a system in

failure mode may be characterized by slow operation, incorrect outputs, or complete

termination of execution. [IEEE 610]

**Failure Mode and Effect Analysis (FMEA):** A systematic approach to risk identification

and analysis of identifying possible modes of failure and attempting to prevent their

occurrence. See also *Failure Mode, Effect and Criticality Analysis (FMECA).*

**Failure Mode, Effect and Criticality Analysis (FMECA):** An extension of FMEA, as in

addition to the basic FMEA, it includes a criticality analysis, which is used to chart the

probability of failure modes against the severity of their consequences. The result

highlights failure modes with relatively high probability and severity of consequences,

allowing remedial effort to be directed where it will produce the greatest value. See also

*Failure Mode and Effect Analysis (FMEA).*

**Failure rate:** The ratio of the number of failures of a given category to a given unit of

measure, e.g. failures per unit of time, failures per number of transactions, failures per

number of computer runs. [IEEE 610]

**False-fail result**: A test result in which a defect is reported although no such defect actually

exists in the test object.

**False-pass result**: A test result which fails to identify the presence of a defect that is actually

present in the test object.

**False-positive result**: See *false-fail result*.

**False-negative result**: See *false-pass result*.

**Fault:** See *defect*.

**Fault attack:** See *attack.*

**Fault density:** See *defect density*.

**Fault Detection Percentage (FDP):** See *Defect Detection Percentage (DDP).*

**Fault masking:** See *defect masking*.

**Fault seeding:** The process of intentionally adding known defects to those already in the

component or system for the purpose of monitoring the rate of detection and removal, and

estimating the number of remaining defects. [IEEE 610]

**Fault seeding tool:** A tool for seeding (i.e. intentionally inserting) faults in a component or

system.

**Fault tolerance:** The capability of the software product to maintain a specified level of

performance in cases of software faults (defects) or of infringement of its specified

interface. [ISO 9126] See also *reliability, robustness*.

**Fault Tree Analysis (FTA):** A technique used to analyze the causes of faults (defects). The

technique visually models how logical relationships between failures, human errors, and

external events can combine to cause specific faults to disclose.

**Feasible path:** A path for which a set of input values and preconditions exists which causes it

to be executed.

**Feature:** An attribute of a component or system specified or implied by requirements

documentation (for example reliability, usability or design constraints). [After IEEE 1008]

**Field testing:** See *beta testing*.

**Finite state machine:** A computational model consisting of a finite number of states and

transitions between those states, possibly with accompanying actions. [IEEE 610]

**Finite state testing:** See *state transition testing*.

**Formal review:** A review characterized by documented procedures and requirements, e.g.

inspection.

**Frozen test basis:** A test basis document that can only be amended by a formal change control process. See also *baseline*.

**Function Point Analysis (FPA):** Method aiming to measure the size of the functionality of

an information system. The measurement is independent of the technology. This

measurement may be used as a basis for the measurement of productivity, the estimation of

the needed resources, and project control.

**Functional integration:** An integration approach that combines the components or systems

for the purpose of getting a basic functionality working early. See also *integration testing*.

**Functional requirement:** A requirement that specifies a function that a component or system must perform. [IEEE 610]

**Functional test design technique:** Procedure to derive and/or select test cases based on an

analysis of the specification of the functionality of a component or system without

reference to its internal structure. See also *black box test design technique*.

**Functional testing:** Testing based on an analysis of the specification of the functionality of a

component or system. See also *black box testing*.

**Functionality:** The capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions. [ISO 9126]

**Functionality testing:** The process of testing to determine the functionality of a software

product.

**G**

**Glass box testing:** See *white box testing*.

**H**

**Hazard analysis**: A technique used to characterize the elements of risk. The result of a hazard analysis will drive the methods used for development and testing of a system. See also *risk analysis.*

**Heuristic evaluation:** A static usability test technique to determine the compliance of a user

interface with recognized usability principles (the so-called “heuristics”).

**High level test case:** A test case without concrete (implementation level) values for input data and expected results. Logical operators are used; instances of the actual values are not yet defined and/or available. See also *low level test case*.

**Horizontal traceability:** The tracing of requirements for a test level through the layers of test

documentation (e.g. test plan, test design specification, test case specification and test

procedure specification or test script).

**Hyperlink**: A pointer within a web page that leads to other web pages.

**Hyperlink tool**: A tool used to check that no broken hyperlinks are present on a web site.

**I**

**Impact analysis:** The assessment of change to the layers of development documentation, test documentation and components, in order to implement a given change to specified

requirements.

**Incident:** Any event occurring that requires investigation. [After IEEE 1008]

**Incident logging:** Recording the details of any incident that occurred, e.g. during testing.

**Incident management:** The process of recognizing, investigating, taking action and disposing of incidents. It involves logging incidents, classifying them and identifying the impact. [After IEEE 1044]

**Incident management tool:** A tool that facilitates the recording and status tracking of

incidents. They often have workflow-oriented facilities to track and control the allocation,

correction and re-testing of incidents and provide reporting facilities. See also *defect*

*management tool.*

**Incident report:** A document reporting on any event that occurred, e.g. during the testing,

which requires investigation? [After IEEE 829]

**Incremental development model:** A development life cycle where a project is broken into a

series of increments, each of which delivers a portion of the functionality in the overall

project requirements. The requirements are prioritized and delivered in priority order in the

appropriate increment. In some (but not all) versions of this life cycle model, each

subproject follows a ‘mini V-model’ with its own design, coding and testing phases.

**Incremental testing:** Testing where components or systems are integrated and tested one or some at a time, until all the components or systems are integrated and tested.

**Independence of testing:** Separation of responsibilities, which encourages the

accomplishment of objective testing. [After DO-178b]

**Infeasible path:** A path that cannot be exercised by any set of possible input values.

**Informal review:** A review not based on a formal (documented) procedure.

**Input:** A variable (whether stored within a component or outside) that is read by a

component.

**Input domain:** The set from which valid input values can be selected. See also *domain*.

**Input value:** An instance of an input. See also *input*.

**Inspection:** A type of peer review that relies on visual examination of documents to detect

defects, e.g. violations of development standards and non-conformance to higher level

documentation. The most formal review technique and therefore always based on a

documented procedure. [After IEEE 610, IEEE 1028] See also *peer review*.

**Inspection leader:** See *moderator*.

**Inspector:** See *reviewer*.

**Install ability:** The capability of the software product to be installed in a specified

environment [ISO 9126]. See also *portability*.

**Install ability testing:** The process of testing the install ability of a software product. See also

*portability testing*.

**Installation guide:** Supplied instructions on any suitable media, which guides the installer

through the installation process. This may be a manual guide, step-by-step procedure,

installation wizard, or any other similar process description.

**Installation wizard:** Supplied software on any suitable media, which leads the installer

through the installation process. It normally runs the installation process, provides

feedback on installation results, and prompts for options.

**Instrumentation:** The insertion of additional code into the program in order to collect

information about program behavior during execution, e.g. for measuring code coverage.

**Instrumenter:** A software tool used to carry out instrumentation.

**Intake test:** A special instance of a smoke test to decide if the component or system is ready

for detailed and further testing. An intake test is typically carried out at the start of the test

execution phase. See also *smoke test.*

**Integration:** The process of combining components or systems into larger assemblies.

**Integration testing:** Testing performed to expose defects in the interfaces and in the

interactions between integrated components or systems. See also *component integration*

*testing, system integration testing*.

**Integration testing in the large:** See *system integration testing*.

**Integration testing in the small:** See *component integration testing*.

**Interface testing:** An integration test type that is concerned with testing the interfaces

between components or systems.

**Interoperability:** The capability of the software product to interact with one or more

specified components or systems. [After ISO 9126] See also *functionality*.

**Interoperability testing:** The process of testing to determine the interoperability of a

software product. See also *functionality testing*.

**Invalid testing:** Testing using input values that should be rejected by the component or

system. See also *error tolerance*.

**Isolation testing:** Testing of individual components in isolation from surrounding

components, with surrounding components being simulated by stubs and drivers, if needed.

**Iterative development model:** A development life cycle where a project is broken into a

usually large number of iterations. An iteration is a complete development loop resulting in

a release (internal or external) of an executable product, a subset of the final product under

development, which grows from iteration to iteration to become the final product.

**K**

**Keyword driven testing:** A scripting technique that uses data files to contain not only test

data and expected results, but also keywords related to the application being tested. The

keywords are interpreted by special supporting scripts that are called by the control script

for the test. See also *data driven testing*.

**L**

**LCSAJ:** A Linear Code Sequence And Jump, consisting of the following three items

(conventionally identified by line numbers in a source code listing): the start of the linear

sequence of executable statements, the end of the linear sequence, and the target line to

which control flow is transferred at the end of the linear sequence.

**LCSAJ coverage:** The percentage of LCSAJs of a component that have been exercised by a

test suite. 100% LCSAJ coverage implies 100% decision coverage.

**LCSAJ testing:** A white box test design technique in which test cases are designed to execute LCSAJs.

**Learnability:** The capability of the software product to enable the user to learn its application. [ISO 9126] See also *usability*.

**Level test plan**: A test plan that typically addresses one test level. See also *test plan.*

**Link testing:** See *component integration testing*.

**Load profile:** A specification of the activity which a component or system being tested may

experience in production. A load profile consists of a designated number of virtual users

who process a defined set of transactions in a specified time period and according to a

predefined operational profile. See also *operational profile.*

**Load testing:** A type of performance testing conducted to evaluate the behavior of a

component or system with increasing load, e.g. numbers of parallel users and/or numbers

of transactions, to determine what load can be handled by the component or system. See

also *performance testing, stress testing*.

**Logic-coverage testing:** See *white box testing*.

**Logic-driven testing:** See *white box testing*.

**Logical test case:** See *high level test case*.

**Low level test case:** A test case with concrete (implementation level) values for input data and expected results. Logical operators from high level test cases are replaced by actual values that correspond to the objectives of the logical operators. See also *high level test case*.

**M**

**Maintenance:** Modification of a software product after delivery to correct defects, to improve performance or other attributes, or to adapt the product to a modified environment. [IEEE 1219]

**Maintenance testing:** Testing the changes to an operational system or the impact of a

changed environment to an operational system.

**Maintainability:** The ease with which a software product can be modified to correct defects,

modified to meet new requirements, modified to make future maintenance easier, or

adapted to a changed environment. [ISO 9126]

**Maintainability testing:** The process of testing to determine the maintainability of a software product.

**Management review:** A systematic evaluation of software acquisition, supply, development,

operation, or maintenance process, performed by or on behalf of management that

monitors progress, determines the status of plans and schedules, confirms requirements and their system allocation, or evaluates the effectiveness of management approaches to

achieve fitness for purpose. [After IEEE 610, IEEE 1028]

**Master test plan:** A test plan that typically addresses multiple test levels. See also *test plan*.

**Maturity:** (1) The capability of an organization with respect to the effectiveness and

efficiency of its processes and work practices. See also *Capability Maturity Model*, *Test*

*Maturity Model*. (2) The capability of the software product to avoid failure as a result of

defects in the software. [ISO 9126] See also *reliability*.

**Measure:** The number or category assigned to an attribute of an entity by making a

measurement. [ISO 14598]

**Measurement:** The process of assigning a number or category to an entity to describe an

attribute of that entity. [ISO 14598]

**Measurement scale:** A scale that constrains the type of data analysis that can be performed

on it. [ISO 14598]

**Memory leak:** A defect in a program's dynamic store allocation logic that causes it to fail to

reclaim memory after it has finished using it, eventually causing the program to fail due to

lack of memory.

**Metric:** A measurement scale and the method used for measurement. [ISO 14598]

**Migration testing:** See *conversion testing*.

**Milestone:** A point in time in a project at which defined (intermediate) deliverables and

results should be ready.

**Modeling tool:** A tool that supports the validation of models of the software or system

[Graham].

**Moderator:** The leader and main person responsible for an inspection or other review

process.

**Modified condition decision coverage:** See *condition determination coverage*.

**Modified condition decision testing:** See *condition determination testing*.

**Modified multiple condition coverage:** See *condition determination coverage*.

**Modified multiple condition testing:** See *condition determination testing*.

**Module:** See *component*.

**Module testing:** See *component testing*.

**Monitor:** A software tool or hardware device that runs concurrently with the component or

system under test and supervises, records and/or analyses the behavior of the component or

system. [After IEEE 610]

**Monitoring tool:** See *monitor.*

**Monkey testing:** Testing by means of a random selection from a large range of inputs and by

randomly pushing buttons, ignorant on how the product is being used.

**Multiple condition:** See *compound condition*.

**Multiple condition coverage:** The percentage of combinations of all single condition

outcomes within one statement that have been exercised by a test suite. 100% multiple

condition coverage implies 100% condition determination coverage.

**Multiple condition testing:** A white box test design technique in which test cases are

designed to execute combinations of single condition outcomes (within one statement).

**Mutation analysis:** A method to determine test suite thoroughness by measuring the extent to which a test suite can discriminate the program from slight variants (mutants) of the

program.

**Mutation testing:** See *back-to-back testing*.

**N**

**N-switch coverage:** The percentage of sequences of N+1 transitions that have been exercised by a test suite. [Chow]

**N-switch testing:** A form of state transition testing in which test cases are designed to execute all valid sequences of N+1 transitions. [Chow] See also *state transition testing*.

**Negative testing:** Tests aimed at showing that a component or system does not work.

Negative testing is related to the testers’ attitude rather than a specific test approach or test

design technique, e.g. testing with invalid input values or exceptions.

**Non-conformity:** Non fulfillment of a specified requirement. [ISO 9000]

**Non-functional requirement:** A requirement that does not relate to functionality, but to

attributes such as reliability, efficiency, usability, maintainability and portability.

**Non-functional testing:** Testing the attributes of a component or system that do not relate to functionality, e.g. reliability, efficiency, usability, maintainability and portability.

**Non-functional test design techniques:** Procedure to derive and/or select test cases for nonfunctional testing based on an analysis of the specification of a component or system

without reference to its internal structure. See also *black box test design technique*.

**O**

**Off-the-shelf software:** A software product that is developed for the general market, i.e. for a large number of customers, and that is delivered to many customers in identical format.

**Operability:** The capability of the software product to enable the user to operate and control it. [ISO 9126] See also *usability*.

**Operational acceptance testing:** Operational testing in the acceptance test phase, typically

performed in a simulated real-life operational environment by operator and/or

administrator focusing on operational aspects, e.g. recoverability, resource-behavior,

installability and technical compliance. See also *operational testing.*

**Operational environment:** Hardware and software products installed at users’ or customers’

sites where the component or system under test will be used. The software may include

operating systems, database management systems, and other applications.

**Operational profile**: The representation of a distinct set of tasks performed by the component or system, possibly based on user behavior when interacting with the component or system, and their probabilities of occurrence. A task is logical rather that physical and can be executed over several machines or be executed in non-contiguous time segments.

**Operational profile testing:** Statistical testing using a model of system operations (short

duration tasks) and their probability of typical use.

**Operational testing:** Testing conducted to evaluate a component or system in its operational environment. [IEEE 610]

**Orthogonal array**: A 2-dimensional array constructed with special mathematical properties,

such that choosing any two columns in the array provides every pair combination of each

number in the array.

**Orthogonal array testing:** A systematic way of testing all-pair combinations of variables

using orthogonal arrays. It significantly reduces the number of all combinations of

variables to test all pair combinations. See also *pairwise testing.*

**Output:** A variable (whether stored within a component or outside) that is written by a

component.

**Output domain:** The set from which valid output values can be selected. See also *domain*.

**Output value:** An instance of an output. See also *output*.

**P**

**Pair programming:** A software development approach whereby lines of code (production

and/or test) of a component are written by two programmers sitting at a single computer.

This implicitly means ongoing real-time code reviews are performed.

**Pair testing:** Two persons, e.g. two testers, a developer and a tester, or an end-user and a

tester, working together to find defects. Typically, they share one computer and trade

control of it while testing.

**Pair wise testing:** A black box test design technique in which test cases are designed to

execute all possible discrete combinations of each pair of input parameters. See also

*orthogonal array testing.*

**Partition testing:** See *equivalence partitioning*.

**Pass:** A test is deemed to pass if its actual result matches its expected result.

**Pass/fail criteria:** Decision rules used to determine whether a test item (function) or feature

has passed or failed a test. [IEEE 829]

**Path:** A sequence of events, e.g. executable statements, of a component or system from an

entry point to an exit point.

**Path coverage:** The percentage of paths that have been exercised by a test suite. 100% path

coverage implies 100% LCSAJ coverage.

**Path sensitizing:** Choosing a set of input values to force the execution of a given path.

**Path testing:** A white box test design technique in which test cases are designed to execute

paths.

**Peer review:** A review of a software work product by colleagues of the producer of the

product for the purpose of identifying defects and improvements. Examples are inspection,

technical review and walkthrough.

**Performance:** The degree to which a system or component accomplishes its designated

functions within given constraints regarding processing time and throughput rate. [After

IEEE 610] See also *efficiency*.

**Performance indicator:** A high level metric of effectiveness and/or efficiency used to guide

and control progressive development, e.g. lead-time slip for software development.

[CMMI]

**Performance profiling:** Definition of user profiles in performance, load and/or stress testing.

Profiles should reflect anticipated or actual usage based on an operational profile of a

component or system, and hence the expected workload. See also *load profile, operational*

*profile.*

**Performance testing:** The process of testing to determine the performance of a software

product. See also *efficiency testing*.

**Performance testing tool:** A tool to support performance testing and that usually has two

main facilities: load generation and test transaction measurement. Load generation can

simulate either multiple users or high volumes of input data. During execution, response

time measurements are taken from selected transactions and these are logged. Performance testing tools normally provide reports based on test logs and graphs of load against response times.

**Phase test plan:** A test plan that typically addresses one test phase. See also *test plan.*

**Pointer:** A data item that specifies the location of another data item; for example, a data item that specifies the address of the next employee record to be processed. [IEEE 610]

**Portability:** The ease with which the software product can be transferred from one hardware or software environment to another. [ISO 9126]

**Portability testing:** The process of testing to determine the portability of a software product.

**Post condition:** Environmental and state conditions that must be fulfilled after the execution

of a test or test procedure.

**Post-execution comparison:** Comparison of actual and expected results, performed after the software has finished running.

**Precondition:** Environmental and state conditions that must be fulfilled before the component or system can be executed with a particular test or test procedure.

**Predicted outcome:** See *expected result*.

**Pretest:** See *intake test*.

**Priority:** The level of (business) importance assigned to an item, e.g. defect.

**Procedure testing:** Testing aimed at ensuring that the component or system can operate in

conjunction with new or existing users’ business procedures or operational procedures.

**Probe effect:** The effect on the component or system by the measurement instrument when

the component or system is being measured, e.g. by a performance testing tool or monitor.

For example performance may be slightly worse when performance testing tools are being

used.

**Problem:** See *defect*.

**Problem management:** See *defect management*.

**Problem report:** See *defect report*.

**Process:** A set of interrelated activities, which transform inputs into outputs. [ISO 12207]

**Process cycle test:** A black box test design technique in which test cases are designed to

execute business procedures and processes. [TMap] See also *procedure testing.*

**Process improvement:** A program of activities designed to improve the performance and

maturity of the organization’s processes, and the result of such a program. [CMMI]

**Production acceptance testing:** See *operational acceptance testing.*

**Product risk:** A risk directly related to the test object. See also *risk*.

**Project:** A project is a unique set of coordinated and controlled activities with start and finish

dates undertaken to achieve an objective conforming to specific requirements, including

the constraints of time, cost and resources. [ISO 9000]

**Project risk:** A risk related to management and control of the (test) project, e.g. lack of

staffing, strict deadlines, changing requirements, etc. See also *risk*.

**Program Instrumenter:** See *instrumenter*.

**Program testing:** See *component testing*.

**Project test plan:** See *master test plan*.

**Pseudo-random:** A series which appears to be random but is in fact generated according to

some prearranged sequence.

**Q**

**Qualification:** The process of demonstrating the ability to fulfill specified requirements. Note

the term ‘qualified’ is used to designate the corresponding status. [ISO 9000]

**Quality:** The degree to which a component, system or process meets specified requirements

and/or user/customer needs and expectations. [After IEEE 610]

**Quality assurance:** Part of quality management focused on providing confidence that quality

requirements will be fulfilled. [ISO 9000]

**Quality attribute:** A feature or characteristic that affects an item’s quality. [IEEE 610]

**Quality characteristic:** See *quality attribute*.

**Quality management:** Coordinated activities to direct and control an organization with regard to quality. Direction and control with regard to quality generally includes the establishment of the quality policy and quality objectives, quality planning, quality control, quality assurance and quality improvement. [ISO 9000]

**R**

**Random testing:** A black box test design technique where test cases are selected, possibly

using a pseudo-random generation algorithm, to match an operational profile. This

technique can be used for testing non-functional attributes such as reliability and

performance.

**Recorder:** See *scribe*.

**Record/playback tool:** See *capture/playback tool*.

**Recoverability:** The capability of the software product to re-establish a specified level of

performance and recover the data directly affected in case of failure. [ISO 9126] See also

*reliability*.

**Recoverability testing:** The process of testing to determine the recoverability of a software

product. See also *reliability testing*.

**Recovery testing:** See *recoverability testing*.

**Regression testing:** Testing of a previously tested program following modification to ensure

those defects have not been introduced or uncovered in unchanged areas of the software, as a result of the changes made. It is performed when the software or its environment is

changed.

**Regulation testing:** See *compliance testing.*

**Release note:** A document identifying test items, their configuration, current status and other delivery information delivered by development to testing, and possibly other stakeholders, at the start of a test execution phase. [After IEEE 829]

**Reliability:** The ability of the software product to perform its required functions under stated conditions for a specified period of time, or for a specified number of operations.

**Reliability growth model:** A model that shows the growth in reliability over time during

continuous testing of a component or system as a result of the removal of defects that result in reliability failures.

**Reliability testing:** The process of testing to determine the reliability of a software product.

**Replaceability:** The capability of the software product to be used in place of another specified software product for the same purpose in the same environment. [ISO 9126] See also *portability*.

**Requirement:** A condition or capability needed by a user to solve a problem or achieve an

objective that must be met or possessed by a system or system component to satisfy a

contract, standard, specification, or other formally imposed document. [After IEEE 610]

**Requirements-based testing:** An approach to testing in which test cases are designed based

on test objectives and test conditions derived from requirements, e.g. tests that exercise

specific functions or probe non-functional attributes such as reliability or usability.

**Requirements management tool:** A tool that supports the recording of requirements,

requirements attributes (e.g. priority, knowledge responsible) and annotation, and

facilitates traceability through layers of requirements and requirements change

management. Some requirements management tools also provide facilities for static

analysis, such as consistency checking and violations to pre-defined requirements rules.

**Requirements phase:** The period of time in the software life cycle during which the

requirements for a software product are defined and documented. [IEEE 610]

**Resource utilization:** The capability of the software product to use appropriate amounts and

types of resources, for example the amounts of main and secondary memory used by the

program and the sizes of required temporary or overflow files, when the software performs

its function under stated conditions. [After ISO 9126] See also *efficiency*.

**Resource utilization testing:** The process of testing to determine the resource-utilization of a software product. See also *efficiency testing*.

**Result:** The consequence/outcome of the execution of a test. It includes outputs to screens,

changes to data, reports, and communication messages sent out. See also *actual result,*

*expected result.*

**Resumption criteria:** The testing activities that must be repeated when testing is re-started

after a suspension. [After IEEE 829]

**Re-testing:** Testing that runs test cases that failed the last time they were run, in order to

verify the success of corrective actions.

**Retrospective meeting:** A meeting at the end of a project during which the project team

members evaluate the project and learn lessons that can be applied to the next project.

**Review:** An evaluation of a product or project status to ascertain discrepancies from planned

results and to recommend improvements. Examples include management review, informal

review, technical review, inspection, and walkthrough. [After IEEE 1028]

**Reviewer:** The person involved in the review that identifies and describes anomalies in the

product or project under review. Reviewers can be chosen to represent different viewpoints

and roles in the review process.

**Review tool:** A tool that provides support to the review process. Typical features include

review planning and tracking support, communication support, collaborative reviews and a

repository for collecting and reporting of metrics.

**Risk:** A factor that could result in future negative consequences; usually expressed as impact

and likelihood.

**Risk analysis:** The process of assessing identified risks to estimate their impact and

probability of occurrence (likelihood).

**Risk-based testing:** An approach to testing to reduce the level of product risks and inform

stakeholders on their status, starting in the initial stages of a project. It involves the

identification of product risks and their use in guiding the test process.

**Risk control:** The process through which decisions are reached and protective measures are

implemented for reducing risks to, or maintaining risks within, specified levels.

**Risk identification:** The process of identifying risks using techniques such as brainstorming,

checklists and failure history.

**Risk level:** The importance of a risk as defined by its characteristics impact and likelihood.

The level of risk can be used to determine the intensity of testing to be performed. A risk

level can be expressed either qualitatively (e.g. high, medium, low) or quantitatively.

**Risk management:** Systematic application of procedures and practices to the tasks of

identifying, analyzing, prioritizing, and controlling risk.

**Risk mitigation:** See *risk control*.

**Risk type:** A specific category of risk related to the type of testing that can mitigate (control)

that category. For example the risk of user-interactions being misunderstood can be

mitigated by usability testing.

**Robustness:** The degree to which a component or system can function correctly in the

presence of invalid inputs or stressful environmental conditions. [IEEE 610] See also

*error-tolerance, fault-tolerance.*

**Robustness testing:** Testing to determine the robustness of the software product.

**Root cause:** A source of a defect such that if it is removed, the occurrence of the defect type is decreased or removed. [CMMI]

**Root cause analysis:** An analysis technique aimed at identifying the root causes of defects. By directing corrective measures at root causes, it is hoped that the likelihood of defect

recurrence will be minimized.

**S**

**Safety:** The capability of the software product to achieve acceptable levels of risk of harm to

people, business, software, property or the environment in a specified context of use.

**Safety critical system:** A system whose failure or malfunction may result in death or serious

injury to people, or loss or severe damage to equipment, or environmental harm.

**Safety testing:** Testing to determine the safety of a software product.

**Sanity test:** See *smoke test*.

**Scalability:** The capability of the software product to be upgraded to accommodate increased loads.

**Scalability testing:** Testing to determine the scalability of the software product.

**Scenario testing:** See *use case testing*.

**Scribe:** The person who records each defect mentioned and any suggestions for process

improvement during a review meeting, on a logging form. The scribe has to ensure that the

logging form is readable and understandable.

**Scripted testing:** Test execution carried out by following a previously documented sequence

of tests.

**Scripting language:** A programming language in which executable test scripts are written,

used by a test execution tool (e.g. a capture/playback tool).

**Security:** Attributes of software products that bear on its ability to prevent unauthorized

access, whether accidental or deliberate, to programs and data. [ISO 9126] See also

*functionality.*

**Security testing:** Testing to determine the security of the software product. See also

*functionality testing.*

**Security testing tool:** A tool that provides support for testing security characteristics and

vulnerabilities.

**Security tool:** A tool that supports operational security.

**Serviceability testing:** See *maintainability testing*.

**Severity:** The degree of impact that a defect has on the development or operation of a

component or system.

**Simulation:** The representation of selected behavioral characteristics of one physical or

abstract system by another system. [ISO 2382/1]

**Simulator:** A device, computer program or system used during testing, which behaves or

operates like a given system when provided with a set of controlled inputs. [After IEEE

610, DO178b] See also *emulator*.

**Site acceptance testing:** Acceptance testing by users/customers at their site, to determine

whether or not a component or system satisfies the user/customer needs and fits within the

business processes, normally including hardware as well as software.

**Smoke test:** A subset of all defined/planned test cases that cover the main functionality of a

component or system, to ascertaining that the most crucial functions of a program work,

but not bothering with finer details. A daily build and smoke test is among industry best

practices. See also *intake test*.

**Software:** Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system. [IEEE 610]

**Software attack:** *See attack.*

**Software Failure Mode and Effect Analysis (SFMEA):** See *Failure Mode and Effect*

*Analysis (FMEA)*.

**Software Failure Mode Effect and Criticality Analysis (SFMECA):** See *Failure Mode*

*and Effect, and Criticality Analysis (FMECA)*.

**Software Fault Tree Analysis (SFTA):** See *Fault Tree Analysis (FTA).*

**Software feature:** See *feature*.

**Software life cycle:** The period of time that begins when a software product is conceived and ends when the software is no longer available for use. The software life cycle typically

includes a concept phase, requirements phase, design phase, implementation phase, test

phase, installation and checkout phase, operation and maintenance phase, and sometimes,

retirement phase. Note these phases may overlap or be performed iteratively.

**Software product characteristic:** See *quality attribute.*

**Software quality:** The totality of functionality and features of a software product that bear on its ability to satisfy stated or implied needs. [After ISO 9126]

**Software quality characteristic:** See *quality attribute*.

**Software test incident:** See *incident*.

**Software test incident report:** See *incident report*.

**Software Usability Measurement Inventory (SUMI):** A questionnaire based usability test

technique to evaluate the usability, e.g. user-satisfaction, of a component or system.

**Source statement:** See *statement*.

**Specification:** A document that specifies, ideally in a complete, precise and verifiable manner, the requirements, design, behavior, or other characteristics of a component or system, and, often, the procedures for determining whether these provisions have been satisfied. [After IEEE 610]

**Specification-based testing:** See *black box testing*.

**Specification-based technique**: See *black box test design technique.*

**Specification-based test design technique:** See *black box test design technique*.

**Specified input:** An input for which the specification predicts a result.

**Stability:** The capability of the software product to avoid unexpected effects from modifications in the software. [ISO 9126] See also *maintainability*.

**Staged representation:** A model structure wherein attaining the goals of a set of process areas establishes a maturity level; each level builds a foundation for subsequent levels.

**Standard software:** See *off-the-shelf software*.

**Standards testing:** See *compliance testing*.

**State diagram:** A diagram that depicts the states that a component or system can assume, and shows the events or circumstances that cause and/or result from a change from one state to another. [IEEE 610]

**State table:** A grid showing the resulting transitions for each state combined with each

possible event, showing both valid and invalid transitions.

**State transition:** A transition between two states of a component or system.

**State transition testing:** A black box test design technique in which test cases are designed to execute valid and invalid state transitions. See also *N-switch testing*.

**Statement:** An entity in a programming language, which is typically the smallest indivisible

unit of execution.

**Statement coverage:** The percentage of executable statements that have been exercised by a test suite.

**Statement testing:** A white box test design technique in which test cases are designed to

execute statements.

**Static analysis:** Analysis of software artifacts, e.g. requirements or code, carried out without

execution of these software artifacts.

**Static analysis tool:** See *static analyzer*.

**Static analyzer:** A tool that carries out static analysis.

**Static code analysis:** Analysis of source code carried out without execution of that software.

**Static code analyzer:** A tool that carries out static code analysis. The tool checks source code, for certain properties such as conformance to coding standards, quality metrics or data flow anomalies.

**Static testing:** Testing of a component or system at specification or implementation level

without execution of that software, e.g. reviews or static code analysis.

**Statistical testing:** A test design technique in which a model of the statistical distribution of

the input is used to construct representative test cases. See also *operational profile testing*.

**Status accounting:** An element of configuration management, consisting of the recording and reporting of information needed to manage a configuration effectively. This information

includes a listing of the approved configuration identification, the status of proposed

changes to the configuration, and the implementation status of the approved changes.

**Storage:** See *resource utilization.*

**Storage testing:** See *resource utilization testing*.

**Stress testing:** A type of performance testing conducted to evaluate a system or component at or beyond the limits of its anticipated or specified work loads, or with reduced availability

of resources such as access to memory or servers. [After IEEE 610] See also *performance*

*testing, load testing*.

**Stress testing tool:** A tool that supports stress testing.

**Structure based testing:** See *white-box testing.*

**Structure-based technique:** See *white box test design technique*.

**Structural coverage:** Coverage measures based on the internal structure of a component or

system.

**Structural test design technique:** See *white box test design technique*.

**Structural testing:** See *white box testing*.

**Structured walkthrough:** See *walkthrough*.

**Stub:** A skeletal or special-purpose implementation of a software component, used to develop or test a component that calls or is otherwise dependent on it. It replaces a called

component. [After IEEE 610]

**Sub path:** A sequence of executable statements within a component.

**Suitability:** The capability of the software product to provide an appropriate set of functions

for specified tasks and user objectives. [ISO 9126] See also *functionality*.

**Suspension criteria:** The criteria used to (temporarily) stop all or a portion of the testing

activities on the test items. [After IEEE 829]

**Syntax testing:** A black box test design technique in which test cases are designed based upon the definition of the input domain and/or output domain.

**System:** A collection of components organized to accomplish a specific function or set of

functions. [IEEE 610]

**System of systems:** Multiple heterogeneous, distributed systems that are embedded in

networks at multiple levels and in multiple domains interconnected addressing large-scale

inter-disciplinary common problems and purposes.

**System integration testing:** Testing the integration of systems and packages; testing

interfaces to external organizations (e.g. Electronic Data Interchange, Internet).

**System testing:** The process of testing an integrated system to verify that it meets specified

requirements.

**T**

**Technical review:** A peer group discussion activity that focuses on achieving consensus on

the technical approach to be taken. [Gilb and Graham, IEEE 1028] See also *peer review*.

**Test:** A set of one or more test cases. [IEEE 829]

**Test approach:** The implementation of the test strategy for a specific project. It typically

includes the decisions made that follow based on the (test) project’s goal and the risk

assessment carried out, starting points regarding the test process, the test design techniques to be applied, exit criteria and test types to be performed.

**Test automation:** The use of software to perform or support test activities, e.g. test

management, test design, test execution and results checking.

**Test basis:** All documents from which the requirements of a component or system can be

inferred. The documentation on which the test cases are based. If a document can be

amended only by way of formal amendment procedure, then the test basis is called a frozen

test basis. [After TMap]

**Test bed:** See *test environment*.

**Test case:** A set of input values, execution preconditions, expected results and execution

post conditions, developed for a particular objective or test condition, such as to exercise a

particular program path or to verify compliance with a specific requirement.

**Test case design technique:** See *test design technique*.

**Test case specification:** A document specifying a set of test cases (objective, inputs, test

actions, expected results, and execution preconditions) for a test item. [After IEEE 829]

**Test case suite:** See *test suite*.

**Test charter:** A statement of test objectives, and possibly test ideas about how to test. Test

charters are used in exploratory testing. See also *exploratory testing*.

**Test closure:** During the test closure phase of a test process data is collected from completed activities to consolidate experience, test ware, facts and numbers. The test closure phase consists of finalizing and archiving the test ware and evaluating the test process, including preparation of a test evaluation report. See also *test process.*

**Test comparator:** A test tool to perform automated test comparison of actual results with

expected results.

**Test comparison:** The process of identifying differences between the actual results produced by the component or system under test and the expected results for a test. Test comparison can be performed during test execution (dynamic comparison) or after test execution.

**Test completion criteria:** See *exit criteria*.

**Test condition:** An item or event of a component or system that could be verified by one or

more test cases, e.g. a function, transaction, feature, quality attribute, or structural element.

**Test control:** A test management task that deals with developing and applying a set of

corrective actions to get a test project on track when monitoring shows a deviation from

what was planned. See also *test management*.

**Test coverage:** See *coverage*.

**Test cycle:** Execution of the test process against a single identifiable release of the test object.

**Test data:** Data that exists (for example, in a database) before a test is executed, and that

affects or is affected by the component or system under test.

**Test data preparation tool:** A type of test tool that enables data to be selected from existing

databases or created, generated, manipulated and edited for use in testing.

**Test design:** (1) See *test design specification*.

(2) The process of transforming general testing objectives into tangible test conditions and

test cases.

**Test design specification:** A document specifying the test conditions (coverage items) for a

test item, the detailed test approach and identifying the associated high level test cases.

**Test design technique:** Procedure used to derive and/or select test cases.

**Test design tool:** A tool that supports the test design activity by generating test inputs from a specification that may be held in a CASE tool repository, e.g. requirements management

tool, from specified test conditions held in the tool itself, or from code.

**Test driven development:** A way of developing software where the test cases are developed, and often automated, before the software is developed to run those test cases.

**Test driver:** See *driver*.

**Test environment:** An environment containing hardware, instrumentation, simulators,

software tools, and other support elements needed to conduct a test. [After IEEE 610]

**Test estimation:** The calculated approximation of a result (e.g. effort spent, completion date, costs involved, number of test cases, etc.) which is usable even if input data may be

incomplete, uncertain, or noisy.

**Test evaluation report:** A document produced at the end of the test process summarizing all

testing activities and results. It also contains an evaluation of the test process and lessons

learned.

**Test execution:** The process of running a test on the component or system under test,

producing actual result(s).

**Test execution automation:** The use of software, e.g. capture/playback tools, to control the

execution of tests, the comparison of actual results to expected results, the setting up of test preconditions, and other test control and reporting functions.

**Test execution phase:** The period of time in a software development life cycle during which

the components of a software product are executed, and the software product is evaluated

to determine whether or not requirements have been satisfied. [IEEE 610]

**Test execution schedule:** A scheme for the execution of test procedures. The test procedures are included in the test execution schedule in their context and in the order in which they are to be executed.

**Test execution technique:** The method used to perform the actual test execution, either

manual or automated.

**Test execution tool:** A type of test tool that is able to execute other software using an

automated test script, e.g. capture/playback.

**Test fail:** See *fail*.

**Test generator:** See *test data preparation tool*.

**Test harness:** A test environment comprised of stubs and drivers needed to execute a test.

**Test incident:** See *incident*.

**Test incident report:** See *incident report*.

**Test implementation:** The process of developing and prioritizing test procedures, creating test data and, optionally, preparing test harnesses and writing automated test scripts.

**Test infrastructure:** The organizational artifacts needed to perform testing, consisting of test environments, test tools, office environment and procedures.

**Test input:** The data received from an external source by the test object during test execution. The external source can be hardware, software or human.

**Test item:** The individual element to be tested. There usually is one test object and many test

items. See also *test object*.

**Test item transmittal report:** See *release note*.

**Test leader:** See *test manager.*

**Test level:** A group of test activities that are organized and managed together. A test level is

linked to the responsibilities in a project. Examples of test levels are component test,

integration test, system test and acceptance test. [After TMap]

**Test log:** A chronological record of relevant details about the execution of tests. [IEEE 829]

**Test logging:** The process of recording information about tests executed into a test log.

**Test manager:** The person responsible for project management of testing activities and

resources, and evaluation of a test object. The individual, who directs, controls, administers,

plans and regulates the evaluation of a test object.

**Test management:** The planning, estimating, monitoring and control of test activities,

typically carried out by a test manager.

**Test management tool:** A tool that provides support to the test management and control part of a test process. It often has several capabilities, such as testware management, scheduling of tests, and the logging of results, progress tracking, incident management and test reporting.

**Test Maturity Model (TMM):** A five level staged framework for test process improvement,

related to the Capability Maturity Model (CMM), that describes the key elements of an

effective test process.

**Test Maturity Model Integrated (TMMi):** A five level staged framework for test process

improvement, related to the Capability Maturity Model Integration (CMMI), that describes

the key elements of an effective test process.

**Test monitoring:** A test management task that deals with the activities related to periodically checking the status of a test project. Reports are prepared that compare the actual to that which was planned. See also *test management*.

**Test object:** The component or system to be tested. See also *test item*.

**Test objective:** A reason or purpose for designing and executing a test.

**Test oracle:** A source to determine expected results to compare with the actual result of the

software under test. An oracle may be the existing system (for a benchmark), a user

manual, or an individual’s specialized knowledge, but should not be the code.

**Test outcome:** See *result*.

**Test pass:** See *pass*.

**Test performance indicator:** A high level metric of effectiveness and/or efficiency used to

guide and control progressive test development, e.g. Defect Detection Percentage (DDP).

**Test phase:** A distinct set of test activities collected into a manageable phase of a project, e.g. the execution activities of a test level.

**Test plan:** A document describing the scope, approach, resources and schedule of intended

test activities. It identifies amongst others test items, the features to be tested, the testing

tasks, who will do each task, degree of tester independence, the test environment, the test

design techniques and entry and exit criteria to be used, and the rationale for their choice,

and any risks requiring contingency planning. It is a record of the test planning process.

**Test planning:** The activity of establishing or updating a test plan.

**Test policy:** A high level document describing the principles, approach and major objectives

of the organization regarding testing.

**Test Point Analysis (TPA):** A formula based test estimation method based on function point

analysis.

**Test procedure:** See *test procedure specification*.

**Test procedure specification:** A document specifying a sequence of actions for the execution of a test. Also known as test script or manual test script. [After IEEE 829]

**Test process:** The fundamental test process comprises test planning and control, test analysis and design, test implementation and execution, evaluating exit criteria and reporting, and test closure activities.

**Test Process Improvement (TPI):** A continuous framework for test process improvement

that describes the key elements of an effective test process, especially targeted at system

testing and acceptance testing.

**Test progress report**: A document summarizing testing activities and results, produced at

regular intervals, to report progress of testing activities against a baseline (such as the

original test plan) and to communicate risks and alternatives requiring a decision to

management.

**Test record:** See *test log*.

**Test recording:** See *test logging*.

**Test reproducibility:** An attribute of a test indicating whether the same results are produced each time the test is executed.

**Test report:** See *test summary report*.

**Test requirement:** See *test condition*.

**Test rig:** *See test environment.*

**Test run:** Execution of a test on a specific version of the test object.

**Test run log:** See *test log*.

**Test result:** See *result*.

**Test scenario:** See *test procedure specification.*

**Test schedule:** A list of activities, tasks or events of the test process, identifying their intended start and finish dates and/or times, and interdependencies.

**Test script:** Commonly used to refer to a test procedure specification, especially an automated one.

**Test session:** An uninterrupted period of time spent in executing tests. In exploratory testing, each test session is focused on a charter, but testers can also explore new opportunities or issues during a session. The tester creates and executes test cases on the fly and records their progress. See also *exploratory testing*.

**Test set:** See *test suite*.

**Test situation:** See *test condition.*

**Test specification:** A document that consists of a test design specification, test case

specification and/or test procedure specification.

**Test specification technique:** See *test design technique.*

**Test stage:** See *test level*.

**Test strategy:** A high-level description of the test levels to be performed and the testing within those levels for an organization or programme (one or more projects).

**Test suite:** A set of several test cases for a component or system under test, where the post

condition of one test is often used as the precondition for the next one.

**Test summary report:** A document summarizing testing activities and results. It also contains an evaluation of the corresponding test items against exit criteria. [After IEEE 829]

**Test target:** A set of exit criteria.

**Test technique:** See *test design technique.*

**Test tool:** A software product that supports one or more test activities, such as planning and

control, specification, building initial files and data, test execution and test analysis.

**Test type:** A group of test activities aimed at testing a component or system focused on a

specific test objective, i.e. functional test, usability test, regression test etc. A test type may

take place on one or more test levels or test phases. [After TMap]

**Testability:** The capability of the software product to enable modified software to be tested.

[ISO 9126] See also *maintainability*.

**Testability review:** A detailed check of the test basis to determine whether the test basis is at an adequate quality level to act as an input document for the test process.

**Testable requirements:** The degree to which a requirement is stated in terms that permits

establishment of test designs (and subsequently test cases) and execution of tests to

determine whether the requirements have been met. [After IEEE 610]

**Tester:** A skilled professional who is involved in the testing of a component or system.

**Testing:** The process consisting of all life cycle activities, both static and dynamic, concerned

with planning, preparation and evaluation of software products and related work products

to determine that they satisfy specified requirements, to demonstrate that they are fit for

purpose and to detect defects.

**Test ware:** Artifacts produced during the test process required to plan, design, and execute

tests, such as documentation, scripts, inputs, expected results, set-up and clear-up

procedures, files, databases, environment, and any additional software or utilities used in

testing.

**Thread testing:** A version of component integration testing where the progressive integration of components follows the implementation of subsets of the requirements, as opposed to the integration of components by levels of a hierarchy.

**Time behavior:** See *performance*.

**Top-down testing:** An incremental approach to integration testing where the component at the top of the component hierarchy is tested first, with lower level components being simulated by stubs. Tested components are then used to test lower level components. The process is repeated until the lowest level components have been tested. See also *integration testing.*

**Traceability:** The ability to identify related items in documentation and software, such as

requirements with associated tests. See also horizontal traceability, vertical traceability.

**U**

**Understandability:** The capability of the software product to enable the user to understand

whether the software is suitable, and how it can be used for particular tasks and conditions of use. [ISO 9126] See also *usability*.

**Unit:** See *component*.

**Unit testing:** See *component testing*.

**Unreachable code:** Code that cannot be reached and therefore is impossible to execute.

**Usability:** The capability of the software to be understood learned, used and attractive to the user when used under specified conditions. [ISO 9126]

**Usability testing:** Testing to determine the extent to which the software product is

understood, easy to learn, easy to operate and attractive to the users under specified

conditions. [After ISO 9126]

**Use case:** A sequence of transactions in a dialogue between a user and the system with a

tangible result.

**Use case testing:** A black box test design technique in which test cases are designed to

execute user scenarios.

**User acceptance testing:** See *acceptance testing*.

**User scenario testing:** See *use case testing*.

**User test:** A test whereby real-life users are involved to evaluate the usability of a component or system.

**Unit test framework:** A tool that provides an environment for unit or component testing in

which a component can be tested in isolation or with suitable stubs and drivers. It also

provides other support for the developer, such as debugging capabilities. [Graham]

**V**

**V-model:** A framework to describe the software development life cycle activities from

requirements specification to maintenance. The V-model illustrates how testing activities

can be integrated into each phase of the software development life cycle.

**Validation:** Confirmation by examination and through provision of objective evidence that

the requirements for a specific intended use or application have been fulfilled. [ISO 9000]

**Variable:** An element of storage in a computer that is accessible by a software program by

referring to it by a name.

**Verification:** Confirmation by examination and through provision of objective evidence that

specified requirements have been fulfilled. [ISO 9000]

**Vertical traceability:** The tracing of requirements through the layers of development

documentation to components.

**Version control:** See *configuration control.*

**Volume testing:** Testing where the system is subjected to large volumes of data. See also

*resource-utilization testing*.

**W**

**Walkthrough:** A step-by-step presentation by the author of a document in order to gather

information and to establish a common understanding of its content. See also *peer review*.

**White-box techniques:** See *white-box test design techniques.*

**White-box test design technique:** Procedure to derive and/or select test cases based on an

analysis of the internal structure of a component or system.

**White-box testing:** Testing based on an analysis of the internal structure of the component or system.

**Wide Band Delphi:** An expert based test estimation technique that aims at making an

accurate estimation using the collective wisdom of the team members.

**Wild pointer:** A pointer that references a location that is out of scope for that pointer or that

does not exist. See also *pointer.*