Standard glossary of terms used in Software Testing

Version 2.1 (dd. April 1st, 2010)

Produced by the 'Glossary Working Party'
International Software Testing Qualifications Board

Editor : Erik van Veenendaal (The Netherlands)

Contributors

Rex Black (USA)
Enst Düring (Norway)
Sigrid Eldh (Sweden)
Isabel Evans (UK)
Simon Frankish (UK)
David Fuller (Australia)
Annu George (India)
Dorothy Graham (UK)
Mats Grindal (Sweden)
Matthias Hamburg (Germany)
Julian Harty (UK)
David Hayman (UK)
Bernard Homes (France)
Ian Howles (UK)
Juha Itkonen (Finland)
Paul Jorgensen (US)
Vipul Kocher (India)
Fernando Lamas de Oliveira (Portugal)
Tilo Linz (Germany)
Gustavo Marquez Sosa (Spain)
Don Mills (UK)
Peter Morgan (UK)
Thomas Müller (Switzerland)
Avi Ofer (Israel)

Dale Perry (USA)
Horst Pohlmann (Germany)
Meile Posthuma (The Netherlands)
Erkki Pöyhönen (Finland)
Maaret Pyhäjärvi (Finland)
Andy Redwood (UK)
Stuart Reid (UK)
Piet de Roo (The Netherlands)
Steve Sampson (UK)
Shane Saunders (UK)
Hans Schaefer (Norway)
Jurriën Seubers (The Netherlands)
Dave Sherratt (UK)
Mike Smith (UK)
Andreas Spillner (Germany)
Lucjan Stapp (Poland)
Richard Taylor (UK)
Geoff Thompson (UK)
Stephanie Ulrich (Germany)
Matti Vuori (Finland)
Gearrel Welvaart (The Netherlands)
Paul Weymouth (UK)
Pete Williams (UK)

Change History

Version 1.3 d.d. May, 31st 2007	
New terms added: - action word driven testing - bug tracking tool - coverage measurement tool - modeling tool - monkey testing - scripted testing - specification-based technique - stress testing tool - structure-based technique - unit test framework - white box technique	*Terms changed:* - basic block - control flow graph - defect management tool - independence of testing - project risk - risk-based testing - test comparator - test process
Version 2.0 d.d. December, 2nd 2007	
New terms added: - attack - buffer - buffer overflow - bug taxonomy - classification tree - control flow analysis - continuous representation - cost of quality - defect based technique - defect based test design technique - defect taxonomy - error seeding tool - Failure Mode, Effect and Criticality Analysis (FMECA) - false-fail result - false-pass result - false-negative result - false-positive result - fault attack - fault seeding - fault seeding tool - hazard analysis - hyperlink - hyperlink tool - load profile - operational acceptance testing - operational profile - orthogonal array - orthogonal array testing - pairwise testing - performance profiling - pointer	*Terms changed:* - bebugging - error seeding - Failure Mode and Effect Analysis (FMEA) - Fault Tree Analysis (FTA) - load testing - horizontal traceability - modified multiple condition testing - process cycle test - product risk - root cause - specification-based technique - stress testing - test charter

- procedure testing - process improvement - production acceptance testing - qualification - reliability growth model - retrospective meeting - risk level - risk type - root cause analysis - safety critical system - software attack - Software Failure Mode and Effect Analysis (SFMEA) - Software Failure Mode Effect and Criticality Analysis (SFMECA) - Software Fault Tree Analysis (SFTA) - software lifecycle - staged representation - system of systems - test design - test estimation - test implementation - Test Maturity Model Integration (TMMi) - test progress report - test rig - test schedule - test session - wild pointer	
Version 2.1 d.d. April 1st, 2010	
New terms added: - accuracy testing - acting (IDEAL) - agile manifesto - agile software development - assessment report - assessor - balanced scorecard - call graph - causal analysis - cause-effect diagram - change management - charter - checklist-based testing - clear box testing - codependent behavior - content-based model - corporate dashboard - critical success factor - critical testing processes	Terms changed: - attack - buffer overflow - control flow analysis - cyclomatic complexity - desk checking - entry point - exit point - Failure Mode, Effects, and Criticality Analysis (FMECA) - invalid testing - memory leak - modeling tool - monkey testing - operational acceptance testing - performance testing tool - risk-based testing - risk type - scribe - Software Failure Mode, Effects, and

- CTP
- dashboard
- dd-path
- Deming cycle
- diagnosing (IDEAL)
- EFQM excellence model
- emotional intelligence
- establishing (IDEAL)
- extreme programming
- fishbone diagram
- Goal Question Metric
- GQM
- hyperlink test tool
- IDEAL
- indicator
- initiating (IDEAL)
- Ishikawa diagram
- lead assessor
- learning (IDEAL)
- lifecycle model
- load testing tool
- manufacturing-based quality
- maturity level
- maturity model
- Mean Time Between Failures
- Mean Time To Repair
- mind-map
- MTBF
- MTTR
- Pareto analysis
- post-project meeting
- process assessment
- process model
- product-based quality
- project retrospective
- quality gate
- Rational Unified Process
- risk category
- RUP
- scorecard
- SCRUM
- session-based test management
- session-based testing
- Software Process Improvement
- SPI
- standard
- STEP
- structure-based test design technique
- suitability testing
- SUMI

Criticality Analysis (SFMECA)
- static analysis
- static testing
- system of systems
- test estimation
- use case
- use case testing

- Systematic Test and Evaluation Process - test deliverable - test improvement plan - Test Process Group - test process improvement manifesto - test process improver - Total Quality Management - TPG - TQM - transactional analysis - transcendent-based quality - user-based quality - value-based quality - WBS - Work Breakdown Structure	

Table of Contents

Foreword

In compiling this glossary the working party has sought the views and comments of as broad a spectrum of opinion as possible in industry, commerce and government bodies and organizations, with the aim of producing an international testing standard which would gain acceptance in as wide a field as possible. Total agreement will rarely, if ever, be achieved in compiling a document of this nature. Contributions to this glossary have been received from the testing communities in Australia, Belgium, Finland, France, Germany, India, Israel, The Netherlands, Norway, Portugal, Spain, Sweden, Switzerland, United Kingdom, and USA.

Many (software) testers have used BS 7925-1 since its original publication in 1998. It has served also as a major reference for the Information Systems Examination Board (ISEB) qualification at both Foundation and Practitioner level. The standard was initially developed with a bias towards component testing, but, since its publication, many comments and proposals for new definitions have been submitted to both improve and expand the standard to cover a wider range of software testing. The ISTQB testing glossary has incorporated many of these suggested updates. It is used as a reference document for the International Software Testing Qualification Board (ISTQB) software testing qualification scheme.

1. Introduction

Much time and effort is wasted both within and between industry, commerce, government and professional and academic institutions when ambiguities arise as a result of the inability to differentiate adequately between such terms as 'statement coverage' and 'decision coverage'; 'test suite', 'test specification' and 'test plan' and similar terms which form an interface between various sectors of society. Moreover, the professional or technical use of these terms is often at variance, with different meanings attributed to them.

2. Scope

This document presents concepts, terms and definitions designed to aid communication in (software) testing and related disciplines.

3. Arrangement

The glossary has been arranged in a single section of definitions ordered alphabetically. Some terms are preferred to other synonymous ones, in which case, the definition of the preferred term appears, with the synonymous ones referring to that. For example *structural testing* refers to *white box testing*. For synonyms, the "See" indicator is used

"See also" cross-references are also used. They assist the user to quickly navigate to the right index term. "See also" cross-references are constructed for relationships such as broader term to a narrower term, and overlapping meaning between two terms.

4. Normative references

At the time of publication, the edition indicated was valid. All standards are subject to revision, and parties to agreements based upon this Standard are encouraged to investigate the possibility of applying the most recent edition of the standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

- BS 7925-2:1998. Software Component Testing.
- DO-178B:1992. Software Considerations in Airborne Systems and Equipment Certification, Requirements and Technical Concepts for Aviation (RTCA SC167).
- IEEE 610.12:1990. Standard Glossary of Software Engineering Terminology.
- IEEE 829:1998. Standard for Software Test Documentation.
- IEEE 1008:1993. Standard for Software Unit Testing.
- IEEE 1012:2004 Standard for Verification and Validation Plans
- IEEE 1028:1997. Standard for Software Reviews and Audits.
- IEEE 1044:1993. Standard Classification for Software Anomalies.
- IEEE 1219:1998. Software Maintenance.
- ISO/IEC 2382-1:1993. Data processing - Vocabulary - Part 1: Fundamental terms.
- ISO 9000:2005. Quality Management Systems – Fundamentals and Vocabulary.
- ISO/IEC 9126-1:2001. Software Engineering – Software Product Quality – Part 1: Quality characteristics and sub-characteristics.
- ISO/IEC 12207:1995. Information Technology – Software Lifecycle Processes.
- ISO/IEC 14598-1:1999. Information Technology – Software Product Evaluation - Part 1: General Overview.
- ISO 15504-9: 1998. Information Technology – Software Process Assessment – Part 9: Vocabulary

5. Trademarks

In this document the following trademarks are used:
- CMM, CMMI and IDEAL are registered trademarks of Carnegie Mellon University
- EFQM is a registered trademark of the EFQM Foundation
- Rational Unified Process is a registered trademark of Rational Software Corporation
- STEP is a registered trademark of Software Quality Engineering
- TMap, TPA and TPI are registered trademarks of Sogeti Nederland BV
- TMM is a registered service mark of Illinois Institute of Technology
- TMMi is a registered trademark of the TMMi Foundation

6. 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. [Gerrard]

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*.

accuracy testing: The process of testing to determine the accuracy of a software product

acting (IDEAL): The phase within the IDEAL model where the improvements are developed, put into practice, and deployed across the organization. The acting phase consists of the activities: create solution, pilot/test solution, refine solution and implement solution. See also *IDEAL*.

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 manifesto: A statement on the values that underpin agile software development. The values are:
- individuals and interactions over processes and tools
- working software over comprehensive documentation
- customer collaboration over contract negotiation
- responding to change over following a plan.

agile software development: A group of software development methodologies based on iterative incremental development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams.

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, problem*.

arc testing: See *branch testing*.

assessment report: A document summarizing the assessment results, e.g. conclusions, recommendations and findings. See also *process assessment*.

assessor: A person who conducts an assessment; any member of an assessment team.

attack: Directed and focused attempt to evaluate the quality, especially reliability, of a test object by attempting to force specific failures to occur. See also *negative testing*.

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 testware: Testware 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]

balanced scorecard: A strategic performance management tool for measuring whether the operational activities of a company are aligned with its objectives in terms of business vision and strategy. See also *corporate dashboard, scorecard*.

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.

bebugging: [Abbott] See *fault seeding*.

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 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.

black box testing: Testing, either functional or non-functional, without reference to the internal structure of the component or system.

blocked test case: A test case that cannot be executed because the preconditions for its execution are 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 an 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 failure 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

call graph: An abstract representation of calling relationships between subroutines in a program.

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*.

causal analysis: The analysis of defects to determine their root cause. [CMMI]

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

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

cause-effect diagram: A graphical representation used to organize and display the interrelationships of various possible root causes of a problem. Possible causes of a real or

potential defect or failure are organized in categories and subcategories in a horizontal tree-structure, with the (potential) defect or failure as the root node. [After Juran]

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]

certification: The process of confirming that a component, system or person complies with its specified requirements, e.g. by passing an exam.

change control: See *configuration control.*

change control board: See *configuration control board.***change management:** (1) A structured approach to transitioning individuals, teams, and organizations from a current state to a desired future state. (2) Controlled way to effect a change, or a proposed change, to a product or service. See also *configuration management.*

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

charter: See *test charter.*

checker: See *reviewer.*

checklist-based testing: An experience-based test design technique whereby the experienced tester uses a high-level list of items to be noted, checked, or remembered, or a set of rules or criteria against which a product has to be verified. See also *experience-based testing.*

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

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, describedby means of a classification tree, are designed to execute combinations of representatives of input and/or output domains. [Grochtmann]

clear-box testing: See *white-box testing*.

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*.

codependent behavior: Excessive emotional or psychological dependence on another person, specifically in trying to change that person's current (undesirable) behavior while supporting them in continuing that behavior. For example, in software testing, complaining about late delivery to test and yet enjoying the necessary "heroism" working additional hours to make up time when delivery is running late, therefore reinforcing the lateness.

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

condition testing: A white box test design technique in which test cases are designed to execute condition outcomes.

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. [IEEE 610]

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. [IEEE 610]

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]

content-based model: A process model providing a detailed description of good engineering practices, e.g. test practices.

continuous representation: A capability maturity model structure wherein capability levels provide a recommended order for approaching process improvement within specified process areas. [CMMI]

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 unique paths (sequences of events) in the execution through a component or system. Control flow analysis evaluates the integrity of control flow structures, looking for possible control flow anomalies such as closed loops or logically unreachable process steps.

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*.

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

corporate dashboard: A dashboard-style representation of the status of corporate performance data. See also *balanced scorecard, dashboard*.

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 item: An entity or property used as a basis for test coverage, e.g. equivalence partitions or code statements.

coverage measurement tool: See *coverage tool*.

coverage tool: A tool that provides objective measures of what structural elements, e.g. statements, branches have been exercised by a test suite.

critical success factor: An element which is necessary for an organization or project to achieve its mission. They are the critical factors or activities required for ensuring the success. See also *content-based model*.

Critical Testing Processes: A content-based model for test process improvement built around twelve critical processes. These include highly visible processes, by which peers and management judge competence and mission-critical processes in which performance affects the company's profits and reputation.

CTP: See *Critical Testing Processes*.

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 or 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.

dashboard: A representation of dynamic measurements of operational performance for some organization or activity, using metrics represented via metaphores such as visual "dials", "counters", and other devices resembling those on the dashboard of an automobile, so that the effects of events or activities can be easily understood and related to operational goals. See also *corporate dashboard, scorecard.*

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. [Beizer]

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.

dd-path: A path of execution (usually through a graph representing a program, such as a flow-chart) that does not include any conditional nodes such as the path of execution between two decisions.

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. [Veenendaal04] See also *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. [After IEEE 610]

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.

Deming cycle: An iterative four-step problem-solving process, (plan-do-check-act), typically used in process improvement. [After Deming]

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 a specification by manual simulation of its execution. See also *static testing.*

development testing: Formal or informal testing conducted during the implementation of a component or system, usually in the development environment by developers. [After IEEE 610]

deviation: See *incident.*

deviation report: See *incident report.*

diagnosing (IDEAL): The phase within the IDEAL model where it is determined where one is, relative to where one wants to be. The diagnosing phase consists of the activities: characterize current and desired states and develop recommendations. See also *IDEAL.*

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.

EFQM (European Foundation for Quality Management) excellence model: A non-prescriptive framework for an organisation's quality management system, defined and owned by the European Foundation for Quality Management, based on five 'Enabling' criteria (covering what an organisation does), and four 'Results' criteria (covering what an organisation achieves).

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]

emotional intelligence: The ability, capacity, and skill to identify, assess, and manage the emotions of one's self, of others, and of groups.

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: An executable statement or process step which defines a point at which a given process is intended to begin..

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].

establishing (IDEAL): The phase within the IDEAL model where the specifics of how an organization will reach its destination are planned. The establishing phase consists of the activities: set priorities, develop approach and plan actions. See also *IDEAL*.

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: An executable statement or process step which defines a point at which a given process is intended to cease..

expected outcome: See *expected result*.

expected result: The behavior predicted by the specification, or another source, of the component or system under specified conditions.

experience-based technique: See *experience-based test design technique.*

experience-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]

extreme programming: A software engineering methodology used within agile software development whereby core practices are programming in pairs, doing extensive code review, unit testing of all code, and simplicity and clarity in code. See also *agile software development.*

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. [After Fenton]

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, Effects, 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*.

fishbone diagram: See *cause-effect diagram.*

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*.

Goal Question Metric: An approach to software measurement using a three-level model: conceptual level (goal), operational level (question) and quantitative level (metric).

GQM: See *Goal Question Metric*.

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

I

IDEAL: An organizational improvement model that serves as a roadmap for initiating, planning, and implementing improvement actions. The IDEAL model is named for the five phases it describes: initiating, diagnosing, establishing, acting, and learning **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 lifecycle 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 lifecycle 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]

indicator: A measure that can be used to estimate or predict another measure. [ISO 14598]

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. **initiating (IDEAL):** The phase within the IDEAL model where the groundwork is laid for a successful improvement effort. The initiating phase consists of the activities: set context, build sponsorship and charter infrastructure. See also *IDEAL.*

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.*

installability: The capability of the software product to be installed in a specified environment [ISO 9126]. See also *portability.*

installability testing: The process of testing the installability 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, negative testing*.

Ishikawa diagram: See *cause-effect diagram.*

isolation testing: Testing of individual components in isolation from surrounding components, with surrounding components being simulated by stubs and drivers, if needed.

item transmittal report: See *release note*.

iterative development model: A development lifecycle 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

key performance indicator: See *performance indicator*.

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, consists 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.

lead assessor: The person who leads an assessment. In some cases, for instance CMMi and TMMi when formal assessments are conducted, the lead-assessor must be accredited and formally trained.

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

learning (IDEAL): The phase within the IDEAL model where one learns from experiences and improves one's ability to adopt new processes and technologies in the future. The learning phase consists of the activities: analyze and validate, and propose future actions. See also *IDEAL*.

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

lifecycle model: A partitioning of the life of a product or project into phases. [CMMI] See also *software lifecycle.*

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.*

load testing tool: See *performance testing tool.*

logic-coverage testing: See *white box testing*. [Myers]

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

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.

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.

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]

manufacturing-based quality: A view of quality, whereby quality is measured by the degree to which a product or service conforms to its intended design and requirements. Quality arises from the process(es) used. [After Garvin] See also *product-based quality, transcendent-based quality, user-based quality, value-based quality.*

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.*

maturity level: Degree of process improvement across a predefined set of process areas in which all goals in the set are attained. [TMMi]

maturity model: A structured collection of elements that describe certain aspects of maturity in an organization, and aid in the definition and understanding of an organization's processes. A maturity model often provides a common language, shared vision and framework for prioritizing improvement actions.

Mean Time Between Failures: The arithmetic mean (average) time between failures of a system. The MTBF is typically part of a reliability growth model that assumes the failed system is immediately repaired, as a part of a defect fixing process. See also *reliability growth model.*

Mean Time To Repair: The arithmetic mean (average) time a system will take to recover from any failure. This typically includes testing to insure that the defect has been resolved.

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 memory access failure due to a defect in a program's dynamic store allocation logic that causes it to fail to release memory after it has finished using it, eventually causing the program and/or other concurrent processes 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.

mind-map: A diagram used to represent words, ideas, tasks, or other items linked to and arranged around a central key word or idea. Mind maps are used to generate, visualize, structure, and classify ideas, and as an aid in study, organization, problem solving, decision making, and writing.

mistake: See *error*.

modeling tool: A tool that supports the creation, amendment and verification 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 of how the product is being used.

MTBF: See *Mean Time Between Failures*.

MTTR: See *Mean Time To Repair*.

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. [After Beizer].

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 test design technique: Procedure to derive and/or select test cases for non-functional 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*.

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.

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) operational environment by operations and/or systems administration staff 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 occurence. 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. [Musa]

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

oracle: See *test oracle*.

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*.

outcome: See *result*.

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.

pairwise 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.*

Pareto analysis: A statistical technique in decision making that is used for selection of a limited number of factors that produce significant overall effect. In terms of quality improvement, a large majority of problems (80%) are produced by a few key causes (20%).

partition testing: See *equivalence partitioning.* [Beizer]

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 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.

postcondition: 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.

post-project meeting: See *retrospective meeting.*

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.

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.*

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.

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

process assessment: A disciplined evaluation of an organization's software processes against a reference model. [after ISO 15504]

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]

process model: A framework wherein processes of the same nature are classified into a overall model, e.g. a test improvement model.

product-based quality: A view of quality, wherein quality is based on a well-defined set of quality attributes. These attributes must be measured in an objective and quantitative way. Differences in the quality of products of the same type can be traced back to the way the specific quality attributes have been implemented. [After Garvin] See also *manufacturing-*

based quality, quality attribute, transcendent-based quality, user-based quality, value-based quality.

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

production acceptance testing: See *operational acceptance testing.*

program instrumenter: See *instrumenter.*

program testing: See *component testing.*

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 retrospective: A structured way to capture lessons learned and to create specific action plans for improving on the next project or next project phase.

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.*

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 gate: A special milestone in a project. Quality gates are located between those phases of a project strongly depending on the outcome of a previous phase. A quality gate includes a formal check of the documents of the previous phase.

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.

Rational Unified Process: A proprietary adaptable iterative software development process framework consisting of four project lifecycle phases: inception, elaboration, construction and transition.

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 that 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. [ISO 9126]

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 lifecycle 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]

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.

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.

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 of their status, starting in the initial stages of a project. It involves the identification of product risks and the use of risk levels to guide the test process.

risk category: See risk type.

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 set of risks grouped by one or more common factors such as a quality attribute, cause, location, or potential effect of risk;. A specific set of product risk types is related to the type of testing that can mitigate (control) that risk type. 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 occurence 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.

RUP: See *Rational Unified Process.*

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. [ISO 9126]

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. [After Gerrard]

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

scenario testing: See *use case testing*.

scorecard: A representation of summarized performance measurements representing progress towards the implementation of long-term goals. A scorecard provides static measurements of performance over or at the end of a defined interval. See also *balanced scorecard, dashboard.*

scribe: The person who records each defect mentioned and any suggestions for process improvement during a review meeting, on a logging form. The scribe should 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).

SCRUM: An iterative incremental framework for managing projects commonly used with agile software development. See also *agile software development.*

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*.

session-based test management: A method for measuring and managing session-based testing, e.g. exploratory testing.

session-based testing: An approach to testing in which test activities are planned as uninterrupted sessions of test design and execution, often used in conjunction with exploratory testing.

severity: The degree of impact that a defect has on the development or operation of a component or system. [After IEEE 610]

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, Effects, and Criticality Analysis (SFMECA): See *Failure Mode,Effects, and Criticality Analysis (FMECA)*.

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

software feature: See *feature*.

software lifecycle: 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 lifecycle 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 Process Improvement: A program of activities designed to improve the performance and maturity of the organization's software processes and the results of such a program. [After CMMI]

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 for measuring software quality from the end user's point of view. [Veenendaal04]

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.

SPI: See *Sofware Process Improvement.*

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. [CMMI]

standard: Formal, possibly mandatory, set of requirements developed and used to prescribe consistent approaches to the way of working or to provide guidelines (e.g., ISO/IEC standards, IEEE standards, and organizational standards). [After CMMI]

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 development artifacts. Static analysis is usually carried out by means of a supporting tool.

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 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. [IEEE 610]

STEP: See *Systematic Test and Evaluation Process.*

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.

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*.

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

structure-based 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]

subpath: 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*.

suitability testing: The process of testing to determine the suitability of a software product

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 integration testing: Testing the integration of systems and packages; testing interfaces to external organizations (e.g. Electronic Data Interchange, Internet).

system of systems: Multiple heterogeneous, distributed systems that are embedded in networks at multiple levels and in multiple interconnected domains, addressing large-scale inter-disciplinary common problems and purposes, usually without a common management structure.

system testing: The process of testing an integrated system to verify that it meets specified requirements. [Hetzel]

Systematic Test and Evaluation Process: A structured testing methodology, also used as a content-based model for improving the testing process. Systematic Test and Evaluation Process (STEP) does not require that improvements occur in a specific order. See also *content-based model.*

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 postconditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement. [After IEEE 610]

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, testware, facts and numbers. The test closure phase consists of finalizing and archiving the testware 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 deliverable: Any test (work) product that must be delivered to someone other than the test (work) product's author. See also *deliverable.*

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. [After IEEE 829]

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 related to various aspects of testing (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 lifecycle 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. [Fewster and Graham]

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 implementation: The process of developing and prioritizing test procedures, creating test data and, optionally, preparing test harnesses and writing automated test scripts.

test improvement plan: A plan for achieving organizational test process improvement objectives based on a thorough understanding of the current strengths and weaknesses of the organization's test processes and test process assets. [After CMMI]

test incident: See *incident*.

test incident report: See *incident report*.

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 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, the logging of results, progress tracking, incident management and test reporting.

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 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 actuals 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), other software, a user manual, or an individual's specialized knowledge, but should not be the code. [After Adrion]

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. [After Gerrard]

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. [After IEEE 829]

test planning: The activity of establishing or updating a test plan.

Test Point Analysis (TPA): A formula based test estimation method based on function point analysis. [TMap]

test policy: A high level document describing the principles, approach and major objectives of the organization regarding testing.

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 Group: A collection of (test) specialists who facilitate the definition, maintenance, and improvement of the test processes used by an organization. [After CMMI]

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 process improvement manifesto: A statement that echoes the agile manifesto, and defines values for improving the testing process. The values are:
- flexibility over detailed processes
- best Practices over templates
- deployment orientation over process orientation
- peer reviews over quality assurance (departments)
- business driven over model driven. [Veenendaal08]

test process improver: A person implementing improvements in the test process based on a test improvement plan.

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 report: See *test summary report* and *test progress report*.

test reproducibility: An attribute of a test indicating whether the same results are produced each time the test is executed.

test requirement: See *test condition*.

test result: See *result*.

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 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. [TMap] See also *CAST*.

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. [After TMap]

testable requirements: The degree to which a requirement is stated in terms that permit 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 lifecycle 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.

testware: 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. [After Fewster and Graham]

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.*

Total Quality Management: An organization-wide management approach centered on quality, based on the participation of all its members and aiming at long-term success through customer satisfaction, and benefits to all members of the organization and to society. Total Quality Management consists of planning, organizing, directing, control, and assurance. [After ISO 8402]

TPG: See *Test Process Group.*

TQM: See *Total Quality Management.*

traceability: The ability to identify related items in documentation and software, such as requirements with associated tests. *See also* horizontal traceability, vertical traceability.

transactional analysis: The analysis of transactions between people and within people's minds; a transaction is defined as a stimulus plus a response. Transactions take place between people and between the ego states (personality segments) within one person's mind.

transcendent-based quality: A view of quality, wherein quality cannot be precisely defined, but we know it when we see it, or are aware of its absence when it is missing. Quality depends on the perception and affective feelings of an individual or group of individuals towards a product. [After Garvin] See also *manufacturing-based quality, product-based quality, user-based quality, value-based quality.*

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 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]

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 an actor and a component or system with a tangible result, where an actor can be a user or anything that can exchange information with the system.

use case testing: A black box test design technique in which test cases are designed to execute scenarios of use cases.

user acceptance testing: See *acceptance testing.*

user-based quality: A view of quality, wherein quality is the capacity to satisfy needs, wants and desires of the user(s). A product or service that does not fulfill user needs is unlikely to find any users. This is a context dependent, contingent approach to quality since different business characteristics require different qualities of a product. [after Garvin] See also *manufacturing-based quality, product-based quality, transcendent-based quality, value-based quality.*

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.

V

V-model: A framework to describe the software development lifecycle activities from requirements specification to maintenance. The V-model illustrates how testing activities can be integrated into each phase of the software development lifecycle.

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]

value-based quality: A view of quality, wherein quality is defined by price. A quality product or service is one that provides desired performance at an acceptable cost. Quality is determined by means of a decision process with stakeholders on trade-offs between time, effort and cost aspects. [After Garvin] See also *manufacturing-based quality, product-based quality, transcendent-based quality, user-based quality.*

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]

version control: See *configuration control.*

vertical traceability: The tracing of requirements through the layers of development documentation to components.

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. [Freedman and Weinberg, IEEE 1028] See also *peer review*.

WBS: See *Work Breakdown Structure.*

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

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*.

Work Breakdown Structure: An arrangement of work elements and their relationship to each other and to the end product. [CMMI]

Annex A (Informative)

Index of sources; the following non-normative sources were used in constructing this glossary:

[Abbott] J. Abbot (1986), *Software Testing Techniques*, NCC Publications.

[Adrion] W. Adrion, M. Branstad and J. Cherniabsky (1982), Validation, Verification and Testing of Computer Software, in: *Computing Surveys*, Vol. 14, No 2, June 1982.

[Bach] J. Bach (2004), Exploratory Testing, in: E. van Veenendaal, *The Testing Practitioner – 2nd edition*, UTN Publishing, ISBN 90-72194-65-9.

[Beizer] B. Beizer (1990), *Software Testing Techniques*, van Nostrand Reinhold, ISBN 0-442-20672-0

[Chow] T. Chow (1978), Testing Software Design Modelled by Finite-Sate Machines, in: *IEEE Transactions on Software Engineering*, Vol. 4, No 3, May 1978.

[CMM] M. Paulk, C. Weber, B. Curtis and M.B. Chrissis (1995), *The Capability Maturity Model, Guidelines for Improving the Software Process*, Addison-Wesley, ISBN 0-201-54664-7

[CMMI] M.B. Chrissis, M. Konrad and S. Shrum (2004), *CMMI, Guidelines for Process Integration and Product Improvement*, Addison Wesley, ISBN 0-321-15496-7

[Deming] D. W. Edwards (1986), *Out of the Crisis*, MIT Center for Advanced Engineering Study, ISBN 0-911379-01-0

[Fenton] N. Fenton (1991), *Software Metrics: a Rigorous Approach*, Chapman & Hall, ISBN 0-53249-425-1

[Fewster and Graham] M. Fewster and D. Graham (1999), *Software Test Automation, Effective use of test execution tools*, Addison-Wesley, ISBN 0-201-33140-3.

[Freedman and Weinberg] D. Freedman and G. Weinberg (1990), *Walkthroughs, Inspections, and Technical Reviews,* Dorset House Publishing, ISBN 0-932633-19-6.

[Garvin] D.A. Garvin (1984), What does product quality really mean?, in: *Sloan Management Review*, Vol. 26, nr. 1 1984

[Gerrard] P. Gerrard and N. Thompson (2002), *Risk-Based E-Business Testing,* Artech House Publishers, ISBN 1-58053-314-0.

[Gilb and Graham] T. Gilb and D. Graham (1993), *Software Inspection*, Addison-Wesley, ISBN 0-201-63181-4.

[Graham] D. Graham, E. van Veenendaal, I. Evans and R. Black (2007), *Foundations of Software Testing, Thomson Learning*, ISBN 978-1-84480-355-2

[Grochtmann] M. Grochtmann (1994), Test Case Design Using Classification Trees, in: *Conference Proceedings STAR* 1994.

[Hetzel] W. Hetzel (1988), *The complete guide to software testing – 2nd edition*, QED Information Sciences, ISBN 0-89435-242-3.

[Juran] J.M. Juran (1979), *Quality Control Handbook,* McGraw-Hill

[McCabe] T. McCabe (1976), A complexity measure, in: *IEEE Transactions on Software Engineering,* Vol. 2, pp. 308-320.

[Musa] J. Musa (1998), *Software Reliability Engineering Testing*, McGraw-Hill Education, ISBN 0-07913-271-5.

[Myers] G. Myers (1979), *The Art of Software Testing*, Wiley, ISBN 0-471-04328-1.

[TMap] M. Pol, R. Teunissen, E. van Veenendaal (2002), *Software Testing, A guide to the TMap Approach,* Addison Wesley, ISBN 0-201-745712.

[Veenendaal04] E. van Veenendaal (2004), *The Testing Practitioner – 2nd edition,* UTN Publishing, ISBN 90-72194-65-9.

[Veenendaal08] E. van Veendaal (2008), Test Improvement Manifesto, in: *Testing Experience*, Issue 04/08, December 2008

Annex B (Method of commenting on this glossary)

Comments are invited on this document so that the glossary can be further improved to satisfy the needs of the testing community.

When making a comment, be sure to include the following information:
- Your name and contact details;
- The version number of the glossary (currently 2.1);
- Exact part of the glossary;
- For new glossary entries, also include a reference to the ISTQB syllabus that uses the term;
- Supporting information, such as the reason for a proposed change, or a reference to the use of a term.

You can submit comments in a variety of ways, which in order of preference are as follows:
1. By E-mail to eve@improveqs.nl;
2. By post to Improve Quality Services BV, attn. Mr. E. van Veenendaal,
   Laan van Diepenvoorde 1, 5551 TC, Waalre, The Netherlands;
3. By FAX to +31 40 20 21450, marked for the attention of Mr. E. van Veenendaal.