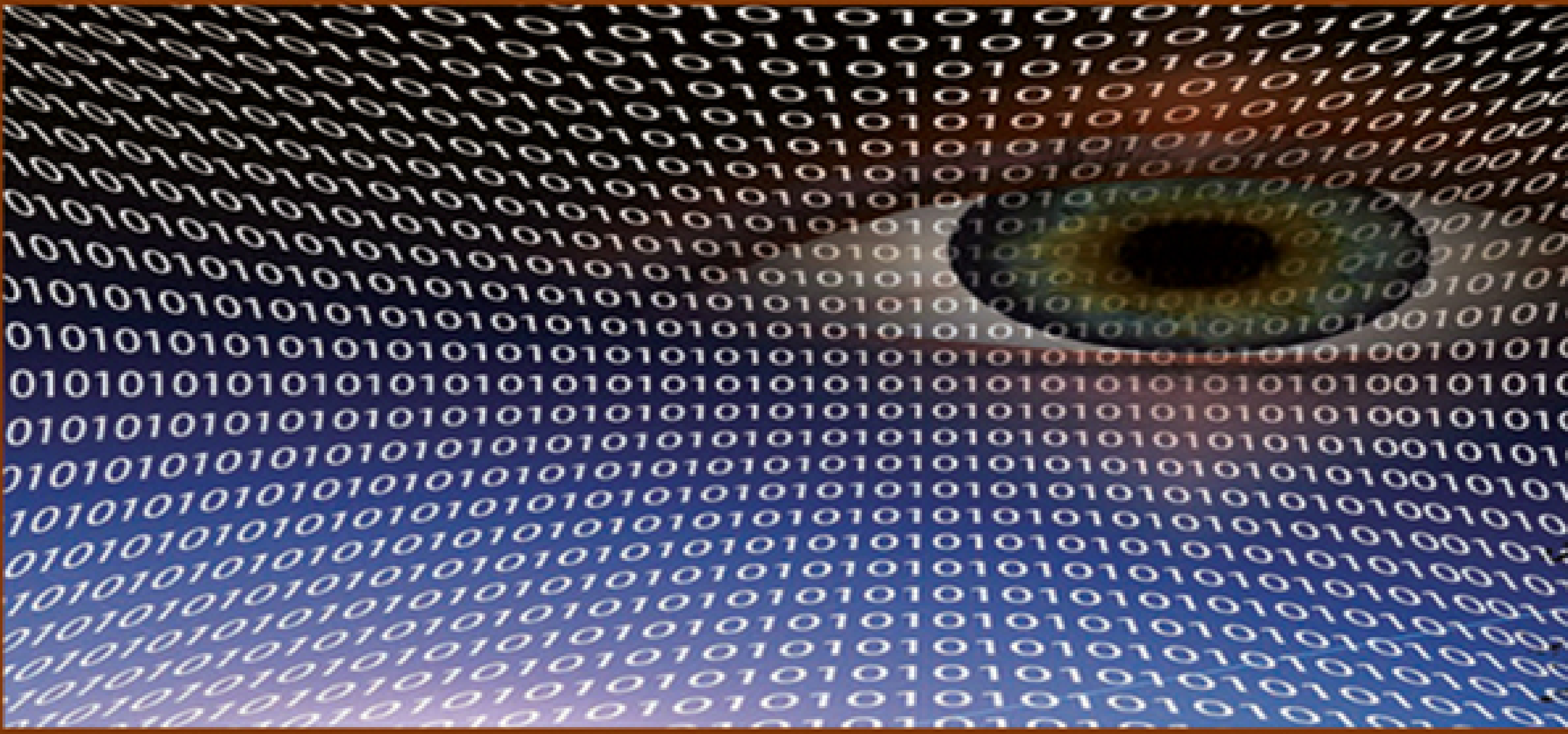

Fundamentals of Software Testing

Bernard Homès

Fundamentals of Software Testing

I would like to dedicate this book to a number of persons:

– to those who came before me and opened the way in the field of testing, many of them are listed in the bibliography;

– to those who will follow me, hoping that this book will provide you with a good start in this wonderful career;

– to my colleagues and dear friends who read the draft and proposed suggestions;

– to my spouse and children who have to suffer a husband and father who is too frequently away and quite demanding.

Fundamentals of Software Testing

Bernard Homès

iSTE WILEY

First published 2012 in Great Britain and the United States by ISTE Ltd and John Wiley & Sons, Inc.

ISTE Ltd
27-37 St George's Road
London SW19 4EU
UK

www.iste.co.uk

John Wiley & Sons, Inc.
111 River Street
Hoboken, NJ 07030
USA

www.wiley.com

Library of Congress Cataloging-in-Publication Data

Homès, Bernard.
Fundamentals of software testing / Bernard Homès.
p. cm.
Includes bibliographical references and index.
ISBN 978-1-84821-324-1
1. Computer software--Testing. I. Title.
QA76.76.T48H674 2012
005.1--dc23

2011040335

British Library Cataloguing-in-Publication Data
A CIP record for this book is available from the British Library
ISBN: 978-1-84821-324-1

Printed and bound in Great Britain by CPI Group (UK) Ltd., Croydon, Surrey CR0 4YY

MIX
Paper from responsible sources
FSC® C013604

Table of Contents

Preface

Why this book

Software testing is becoming more and more important in the industry, reflecting the increasing importance of software quality in today's world.

Due to the lack of formal and recognized training in software testing, a group of specialist consultants gathered together in 2002 and founded the International Software Testing Qualifications Board (ISTQB). They defined the minimal set of methodological and technical knowledge that testers should know depending on their experience. This was gathered into what is called a syllabus. The foundation level syllabus was reviewed in 2011 and is the basis of an international certification scheme, already obtained by more than 200,000 testers worldwide. For testers who wish to prepare for the ISTQB foundation level exam, this book can serve as reference material and a study guide. It references the 2011 version of the ISTQB Certified Tester Foundation Level syllabus.

This book follows the order and chapters of the syllabus, helping you to successfully complete the certification exam. It is a one-stop reference book offering you:

– more detailed explanations than those found in the ISTQB syllabus;

– definitions of the terms (i.e. the Glossary) used in the certification exams;

– practice questions similar to those encountered during the certification exam;

– a sample exam.

For testers who want to acquire a good understanding of software and system tests, this book provides the fundamental principles as described by the ISTQB and recognized experts.

This book provides answers and areas of discussion allowing test leaders and managers to:

– improve their understanding of testing;

– have an overview of process improvement linked to software testing;

– increase the efficiency of their software development and tests.

Throughout this book, you will find learning objectives (noted FLO-…) that represent the ISTQB foundation level syllabus learning objectives. These are the topics that certification candidates should know and that are examined in the ISTQB certification exams.

Prerequisite

Software testing does not require specific prerequisites. Although it is not mandatory, a common understanding of data processing and software allows you to have a better understanding of software testing.

The reader with software development knowledge, whatever the programming language, will understand certain aspects faster, but a simple practice as a user should be enough to understand this book.

ISTQB, CFTL (Comité Français des Tests Logiciels) and national boards

The ISTQB is a not-for-profit international association grouping national software testing boards covering approximately 50 countries. These national boards are made up of software testing specialists, consultants, and experts, and together they define the syllabi and examination directives for system and software testers. The CFTL represents France on the ISTQB and offers the ISTQB certification in France.

To define the syllabus content for all three software tester certification levels (i.e. foundation, advanced, expert), and the applicable exam directives, the ISTQB has created a number of working groups, each in charge of a specific subject (i.e. foundation level syllabus, advanced level syllabus, expert level syllabi, training provider accreditation, examination specification, etc.). These work groups are, as the national boards, made up of software testing experts and specialists, consultants, presenters at conferences, professors and national or international specialists in software testing and systems quality. Their combined expertise enables them to synthesize knowledge from numerous fields (aeronautics, space, medical, commercial, rail, telecoms, etc.) and various levels (technicians, analysts, project leaders, specialists, experts, researcher, managers, etc.).

A number of prominent authors of software testing books participate in the creation of the syllabi, ensuring that these reflect what a tester should know depending on his/her level of experience (foundation, advanced, expert) and on his/her objectives (test management, functional testing, and test techniques, specialization in software security or performance testing, etc.).

Glossary and syllabus

The ISTQB is aware of the broad diversity of terms used and the associated diversity of interpretation of these terms depending upon the customers, countries, and organizations. A common glossary of software testing terms has been set up and national boards provide translation of these terms in national languages to promote better understanding of the terms and the associated concepts. This becomes more and more important in a context of international cooperation and offshore sub-contracting.

The syllabi define the basis of the certification exams; they also help to define the scope of training and are applicable at three levels of experience: foundation level, advanced level and expert level. This book focuses on the foundation level.

The foundation level, made up of a single module, is detailed in the following chapters.

The advanced level is made up of three modules:

– test manager, which focuses on the test management and test process improvements aspects;

– test analyst, which focuses on the testing of characteristics of the software and systems, mostly without the use of tools; and

– technical test analyst, which focuses on the testing of non-functional characteristics of software and systems, mostly with the use of tools.

The expert level focuses on specific aspects, such as test management, test process improvement, or performances.

ISTQB certification

The ISTQB proposes software tester certifications, which are recognized as equivalent by all ISTQB member boards throughout the world. The level of difficulty of the questions and the exams are based on the same criteria (defined in the syllabi) and the same terms (defined in the Glossary).

The certification exams proposed by the CFTL and the national boards of the ISTQB enable the candidates to validate their knowledge, and assure employers or potential customers of a minimum level of knowledge from their testers, whatever their origin.

These certifications are recognized as equivalent throughout the whole world, enabling international cross-recognition. In France, more than 1,500 people have successfully passed the certification, and more than 200,000 have acquired such a certification worldwide. This shows the need from the industry to have an independent certification of the software testing activities.

Key for understanding the content

To be used efficiently, this book has the following characteristics:

FLO-xxx: text that starts with FLO-xxx is a reminder of the learning objectives present in the ISTQB foundation level syllabus for certified testers. Those objectives are expanded in the paragraphs following this tag.

The titles of the chapters correspond to those of the ISTQB foundation level syllabus, version 2011. This is often the case too for the section heads; the syllabus reference is provided in the form (FLx.y) where x.y stands for the chapter and section head of the ISTQB foundation level syllabus. The example below shows that the section of the book refers to the 2.4 section of the foundation syllabus:

2.4. Tests and maintenance (FL2.4)

A synopsis closes each of the chapters, summarizing the aspects covered and identifying the terms of the glossary to know for the certification exam. Sample exam questions are also provided at the end of each chapter. These questions were developed by applying the same criteria as for the creation of real exam questions.

The sample questions provided in Chapters 1, 2, 3, 4, 5 and 6 are reproduced with kind permission of © Bernard Homès 2011.

Glossary

The definitions hereafter are extracted from the International Software Testing Qualifications Board (ISTQB) *Standard Glossary of Terms used in Software Testing*. Only the terms used for the Foundation Level certification exams are mentioned, so as not to drown the reader in terms that are used at other levels or in other syllabi.

ACM: (Association for Computer Machinery) professional and scientific association for the development of information technology as science and profession.

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.

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 as a form of internal acceptance testing.

Attack: Directed and focused attempt to evaluate the quality, especially reliability, of a test object by attempting to force specific failures to occur.

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 in order to acquire feedback from the market.

Black-box technique: See *black-box testing.*

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

Boundary value analysis: A black-box test design technique in which test cases are designed based on boundary values.

Branch coverage: The percentage of branches that have been exercised by a test suite. One hundred percent branch coverage implies both 100% decision coverage and 100% statement coverage

Bug: see *defect*.

CFTL: *Comité Français des Tests Logiciels*, French association [LOI 01] for the development of testing in France and French-speaking countries.

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.

Commercial off-the-shelf software (COTS): See *off-the-shelf software*.

Compiler: A software tool that translates programs expressed in a high-order language into their machine language equivalents.

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*.

Component integration testing: Tests executed to identify defects in the interfaces and interactions between integrated components.

Component testing: The testing of individual software components.

Configuration control: An element of configuration management, consisting of the evaluation, co-ordination, approval or disapproval, and implementation of changes to configuration items after formal establishment of their configuration identification.

Configuration 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.

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.

Confirmation testing: See *re-testing*.

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

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

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 the test suite.

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 calling graph and a subroutine).

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. 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.

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 coverage: The percentage of decision outcomes that have been exercised by a test suite. One hundred percent decision coverage implies both 100% branch coverage and 100% statement coverage.

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.

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 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 management: The process of recognizing, investigating, taking action, and disposing of defects. It involves recording defects, classifying them, and identifying the impact.

Defect management tool: See *incident management tool*.

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.

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 highlight memory leaks.

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

Entry criteria: The set of generic and specific conditions for permitting a process to proceed with a defined task, e.g. test phase. The purpose of entry criteria is to prevent a task starting that would entail more (wasted) effort compared to the effort needed to remove the failed entry criteria.

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.

Error: A human action that produces an incorrect result [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.

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 by testing to report against and to plan when to stop testing.

Exhaustive testing: A test approach in which the test suite comprises all combinations of input values and preconditions.

Exploratory testing: Testing 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.

Failure: Actual deviation of the component or system from its expected delivery, service, or result (according to Fenton). The inability of a system or system component to perform a required function within specified limits. A failure may be produced when a fault is encountered [EUR 00].

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.

Fault attack: See *attack*.

Field testing: See *beta testing*.

Finite state testing: See *state transition testing*.

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

Functional requirement: A requirement that specifies a function that a component or system must perform.

Functional testing: Testing based on an analysis of the specification of the functionality of a component or system. See also *black-box testing*.

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).

IEEE: Institute for Electrical and Electronic Engineers, a professional, not for profit association for the advancement of technology, based on the electrical and electronic technologies. This association is active in the design of standards. There is a French chapter of this association providing publications useful for software testers.

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 during testing which requires investigation.

Incident report: A document reporting on any event that occurs during the testing which requires investigation.

Incident management tool: A tool that facilitates the recording and status tracking of incidents found during testing. 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*.

Incremental development model: A development life cycle where a project is broken into a series of increments, each of which delivers a portion of the functionality in the overall project requirements. The requirements are prioritized and delivered in priority order in the appropriate increment. In some (but not all) versions of this life cycle model, each sub-project follows a “mini V-model” with its own design, coding and testing phases.

Independence of testing: Separation of responsibilities, which encourages the accomplishment of objective testing.

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

Inspection: A type of 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. See also *peer review*.

Intake test: A special instance of a smoke test to decide whether 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*.

Interoperability testing: The process of testing to determine the interoperability of a software product. See also *functionality testing*.

ISTQB: International Software Testing Qualifications Board, a nonprofit association developing international certification for software testers.

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*.

Master test plan: See *project test plan*.

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

Metric: A measurement scale and the method used for measurement.

Mistake: See *error*.

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

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

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

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*.

Non-functional requirement: A requirement that does not relate to functionality, but to attributes of it such as reliability, efficiency, usability, maintainability, and portability.

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.

Oracle: See *test oracle*

Peer review: See *technical review*.

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

Performance testing tool: A tool to support performance testing and that usually has two main facilities: load generation and test transaction measurement. Load generation can simulate either multiple users or high volumes of input data. During execution, response time measurements are taken from selected transactions and these are logged. Performance testing tools normally provide reports based on test logs and graphs of load against response times.

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

Probe effect: The effect on the component or system when it 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.

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

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

Quality: The degree to which a component, system or process meets specified requirements and/or user/customer needs and expectations.

RAD: Rapid Application Development, a software development model.

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.

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

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.

Requirement management tool: A tool that supports the recording of requirements, attributes of requirements (e.g. priority, knowledge responsible), and annotation, and facilitates traceability through layers of requirements and requirement change management. Some requirement management tools also provide facilities for static analysis, such as consistency checking and violations to pre-defined requirement rules.

Re-testing: Testing that runs test cases that failed the last time they were run, in order to verify the success of corrective actions.

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 walk-through.

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 who 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-based testing: An approach to testing to reduce the level of product risks and inform stakeholders on their status, starting in the initial stages of a project. It involves the identification of product risks and their use in guiding the test process.

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

SBTM: Session-based test management, an ad hoc and exploratory test management technique, based on fixed length sessions (from 30 to 120 minutes) during which testers explore a part of the software application.

Scribe: The person who has to record each defect mentioned and any suggestions for improvement during a review meeting, on a logging form. The scribe has to ensure that the logging form is readable and understandable.

Scripting language: A programming language in which executable test scripts are written, used by a test execution tool (e.g. a capture/replay tool).

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

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.

SLA: Service level agreement, service agreement between a supplier and its client, defining the level of service a customer can expect from the provider.

Smoke test: A subset of all defined/planned test cases that cover the main functionality of a component or system, to ascertain 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.

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

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

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

Static testing: Testing of a component or system at specification or implementation level without execution of that software, e.g. reviews or static code analysis.

Stress testing: A type of performance testing conducted to evaluate a system or component at or beyond the limits of its anticipated or specified workloads, or with reduced availability of resources such as access to memory or servers. See also *performance testing*, *load testing*.

Stress testing tool: A tool that supports stress testing.

Structural testing: See *white-box testing*.

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.

System integration testing: Testing the integration of systems and packages; testing interfaces to external organizations (e.g. electronic data interchange, the internet).

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

Technical review: A peer group discussion activity that focuses on achieving consensus on the technical approach to be taken. A technical review is also known as a peer review.

Test: A set of one or more test cases.

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 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.

Test case: A set of input values, execution preconditions, expected results, and execution post conditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement.

Test case specification: A document specifying a set of test cases (objective, inputs, test actions, expected results, and execution preconditions) for a test item.

Test comparator: A test tool to perform automated test comparison.

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, 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 data: Data that exists (for example, in a database) before a test is executed, and that affects or is affected by the component or system under test.

Test data preparation tool: A type of test tool that enables data to be selected from existing databases or created, generated, manipulated and edited for use in testing.

Test design: The process of transforming general testing objectives into tangible test conditions and test cases. See *test design specification*.

Test design specification: A document specifying the test conditions (coverage items) for a test item, the detailed test approach, and identifying the associated high level test cases.

Test design technique: A method used to derive 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, or from specified test conditions held in the tool itself.

Test-driven development: Agile development method, where the tests are designed and automated before the code (from the requirements or specifications), then the minimal amount of code is written to successfully pass the test. This iterative method ensures that the code continues to fulfill requirements via test execution.

Test environment: An environment containing hardware, instrumentation, simulators, software tools, and other support elements needed to conduct a test.

Test execution: The process of running a test by the component or system under test, producing actual results.

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 tool: A type of test tool that is able to execute other software using an automated test script, e.g. capture/playback.

Test harness: A test environment comprised of stubs and drivers needed to conduct a test.

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.

Test log: A chronological record of relevant details about the execution of tests.

Test management: The planning, estimating, monitoring, and control of test activities, typically carried out by a test manager.

Test manager: The person responsible for testing and evaluating a test object. The individual, who directs, controls, administers, plans, and regulates the evaluation of a test object.

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 results with what was expected. See also *test management*.

Test objective: A reason or purpose for designing and executing a test.

Test oracle: A source to determine expected results to compare with the actual result of the software under test. An oracle may be the existing system (for a benchmark), a user manual, or an individual's specialized knowledge, but should not be the code.

Test plan: A document describing the scope, approach, resources, and schedule of intended test activities. Amongst others, it identifies test items, the features to be tested, the testing tasks, who will do each task, the degree of tester independence, the test environment, the test design techniques, and test measurement techniques to be used, and the rationale for their choice, and any risks requiring contingency planning. It is a record of the test planning process.

Test 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 the test script or manual test script.

Test report: See *test summary report*.

Test script: Commonly used to refer to a test procedure specification, especially an automated one.

Test strategy: A high-level document defining the test levels to be performed and the testing within those levels for a program (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.

Tester: A technically skilled professional who is involved in the testing of a component or system.

Testware: Artifacts produced during the test process required to plan, design, and execute tests, such as documentation, scripts, inputs, expected results, setup and clear-up procedures, files, databases, environment, and any additional software or utilities used in testing.

Thread testing: A version of component integration testing where the progressive integration of components follows the implementation of subsets of the requirements, as opposed to the integration of components by levels of a hierarchy.

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

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.

Use case testing: A black-box test design technique in which test cases are designed to execute user scenarios.

User acceptance testing: See *acceptance testing*.

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