
Software and systems engineering — Software testing —

Part 1: General concepts

*Ingénierie du logiciel et des systèmes — Essais du logiciel —
Partie 1: Concepts généraux*





COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2022

© IEEE 2022

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO or IEEE at the respective address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Institute of Electrical and Electronics Engineers, Inc
3 Park Avenue, New York
NY 10016-5997, USA

Email: stds.ipr@ieee.org
Website: www.ieee.org



Published in Switzerland

Authorized licensed use limited to: Tsinghua University. Downloaded on March 06, 2026 at 22:53:15 UTC from IEEE Xplore. Restrictions apply.

THIS IS A REDLINE DOCUMENT PRODUCED BY TECHSTREET, AN ASME COMPANY.

This document is intended to provide users with an indication of changes from one edition to the next. It includes a full-text version of the new document, plus an indication of changes from the previous version.

Redlines are designed to save time and improve efficiencies by using the latest software technology to find and highlight document changes. More professionals are using valuable new technologies like redlines, to help improve outcomes in a fast paced global business world.

Because it may not be technically possible to capture all changes accurately, it is recommended that users consult previous editions as appropriate. In all cases, only the current base version of this publication is to be considered the official document.

This Redline is produced solely by Techstreet. IEEE has not prepared this Redline and is not responsible for its accuracy.

Redline Processing Notes:

1. ~~Red Text~~ - Red strikethrough text denotes deletions.
2. Blue Text - Blue underlined text denotes modifications and additions.

Contents

Page

Foreword	v
Introduction	vii
1 Scope	1
2 Normative references	1
3 Terms and definitions	1
4 Software testing concepts	16
4.1 Introduction to software testing	16
4.1.1 Overview	16
4.1.2 Relationship to quality management	16
4.1.3 Verification and validation	16
4.1.4 Test item	17
4.1.5 Static and dynamic testing	17
4.1.6 Exhaustive testing and sampling	18
4.1.7 Testing as a heuristic	18
4.1.8 Purpose of testing	18
4.1.9 Test basis	19
4.1.10 Test oracle	19
4.1.11 Test independence	19
4.2 Test plans and test strategies	19
4.2.1 General	19
4.2.2 Risks and risk management	20
4.2.3 Risks and requirements as the basis of a test strategy	20
4.2.4 Test approaches	21
4.2.5 Testing in development and maintenance life cycles	22
4.2.6 Domains and system characteristics	23
4.2.7 Test strategy contents	23
4.3 Test frameworks	24
4.3.1 Test processes	24
4.3.2 Test documentation	28
4.3.3 Documentation requirements	30
4.3.4 Configuration management and testing	30
4.3.5 Tool support	30
4.3.6 Process improvement and testing	30
4.3.7 Test metrics	31
4.4 Test design and execution	31
4.4.1 Test model	31
4.4.2 Model-based testing	32
4.4.3 Scripted and exploratory testing	33
4.4.4 Test design techniques	33
4.4.5 Experience-based testing	34
4.4.6 Retesting and regression testing	35
4.4.7 Manual and automated testing	35
4.4.8 Continuous testing	35
4.4.9 Back-to-back testing	35
4.4.10 A/B testing	36
4.4.11 Mathematical-based and fuzz testing	36
4.4.12 Test environments	36
4.4.13 Test data management	37
4.5 Project management and testing	37
4.6 Communication and reporting	37
4.7 Defects and incident management	39

Annex A (informative) System characteristics and testing – examples


Documents

Document Sharing Hub

Annex B (informative) Testing roles.....46

Bibliography.....47

IEEE Notices and Abstract.....48

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO/IEC documents should be noted. This document was drafted in accordance with the rules given in the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see <https://patents.iec.ch>).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

ISO/IEC/IEEE 29119-1 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 7, *Software and systems engineering*, in cooperation with the Systems and Software Engineering Standards Committee of the IEEE Computer Society, under the Partner Standards Development Organization cooperation agreement between ISO and IEEE.

This second edition cancels and replaces the first edition (ISO/IEC/IEEE 29119-1:2013), which has been technically revised.

The main changes are as follows:

- Testing terms and their definitions that are not covered within this document have been removed. This has led to this document being renamed from 'Concepts and definitions' to 'General concepts'.
- The coverage of test concepts has been made more concise and re-ordered.
- The concept of test sub-processes has been removed due to its complexity and replaced with additional coverage of the instantiation of test processes.
- The expected content of a test strategy has been clarified.

- A simplified test design process is described, with the derivation of test cases now based on test models rather than on test conditions.
- The coverage of metrics and measures has been moved from an annex into the body of the document.
- The annex explaining how testing fits into different life cycle models has been removed.
- A new annex providing examples of how systems from different domains are associated with certain characteristics and test approaches has been added.

A list of all parts in the ISO/IEC/IEEE 29119 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

The purpose of the ISO/IEC/IEEE 29119 series is to define an internationally agreed set of standards for software testing that can be used by any organization when performing any form of software testing and using any life cycle.

It is recognized that there are many different types of software, software organizations, and methodologies. Software domains include information technology (IT), personal computers (PC), embedded, mobile, scientific and many other classifications. Software organizations range from small to large, co-located to world-wide, and commercial to those providing a public service. Software development methodologies include object-oriented, traditional, agile and DevOps. These and other factors influence software testing. The ISO/IEC/IEEE 29119 series can support testing in many different contexts.

This document facilitates the use of other parts in the ISO/IEC/IEEE 29119 series by introducing the general concepts on which the ISO/IEC/IEEE 29119 series is built.

A general introduction to software testing is provided. The role of software testing in quality management and as part of verification and validation is described; and its implementation in the form of both static and dynamic testing is defined. The impracticality of exhaustive testing and the need for sampling are explained; and the importance of the test basis and test oracle are described. The benefits of test independence are introduced.

Test plans and test strategies are described in the context of risk-based testing, which is the recommended approach to strategizing and managing testing that underlies the ISO/IEC/IEEE 29119 series and provides the basis for test prioritization and focus. Test levels, test types and test design techniques (and corresponding measures) are described in the context of their inclusion as part of the test strategy.

Various test frameworks are presented, including test processes (and test process improvement), test metrics, test documentation, configuration management and tool support.

The performance of test design and execution based on the use of a test model is described. Several of the most important test design and execution choices are considered, including scripted and exploratory testing approaches, the importance of test design techniques for the creation of test cases, test patterns, retesting and regression testing, manual and automated testing, back-to-back and A/B testing.

Several activities that directly support test design and executions are introduced, including test environments, test data management, communications and reporting and defect and incident management.

[Annex A](#) briefly describes a number of system characteristics and suggested associated test approaches. If a tester can identify which of the system characteristics apply to the system they are testing, then they should consider whether the specialized testing listed for the characteristic is appropriate for inclusion in their test strategy.

[Annex B](#) introduces several generic testing roles and briefly describes their responsibilities.

The test process model that the ISO/IEC/IEEE 29119 series is based on is defined in detail in ISO/IEC/IEEE 29119-2. ISO/IEC/IEEE 29119-2 covers the software testing processes at the organizational level, test management level and for dynamic test levels. Testing is the primary approach to risk treatment in software development. This document defines a risk-based approach to testing.

Templates and examples of test documentation that are produced during the testing process are defined in ISO/IEC/IEEE 29119-3. Software testing techniques that can be used during testing are defined in ISO/IEC/IEEE 29119-4.

While this document is informative, ISO/IEC/IEEE 29119-2, ISO/IEC/IEEE 29119-3, and ISO/IEC/IEEE 29119-4 are normative, meaning that they include requirements for anyone wanting to claim conformance to these standards. Users who want to use the standards but have not purchased the full document details ISO/IEC/IEEE 29119-1:2021

for not following every requirement (e.g. for someone following an agile approach to development and testing) can claim tailored conformance as long as the level of tailoring and its rationale are described and agreed. Specific details of conformance are provided in the relevant conformance clause in each of the standards.

The ISO/IEC/IEEE 29119 series can be used in isolation or can be used as part of a larger set of standards that cover other aspects of the software life cycle. For instance, some users use ISO/IEC/IEEE 12207 to define software system life cycle models appropriate to their products and services (and some may use the corresponding systems engineering standard, ISO/IEC/IEEE 15288), and reference the ISO/IEC/IEEE 29119 series for their software testing needs.

Together, the ISO/IEC/IEEE 29119 series aims to provide stakeholders with the ability to manage and perform software testing in any organization.

Software and systems engineering — Software testing —

Part 1:

General concepts ~~and definitions~~

1 Scope

This document ~~part of ISO/IEC/IEEE 29119~~ specifies general ~~definitions and~~ concepts in software testing. ~~It provides definitions of testing terms and presents discussion of concepts~~ key concepts for ~~to the understanding of the~~ ISO/IEC/IEEE 29119 series ~~of software testing international standards.~~

~~2 Conformance~~

~~ISO/IEC/IEEE 29119-1 is informative and no conformance with it is required.~~

~~The ISO/IEC/IEEE 29119 software testing series of standards contain three standards where conformance may be claimed:~~

~~— test processes;~~

~~— test documentation;~~

~~— test techniques.~~

~~Conformance is addressed in ISO/IEC/IEEE 29119-2, ISO/IEC/IEEE 29119-3 and ISO/IEC/IEEE 29119-4.~~

2 Normative references

There are no normative references in this document.

~~This document does not require the use of any normative references. Standards useful for the implementation and interpretation of this part of ISO/IEC/IEEE 29119 are listed in the Bibliography.~~

3 Terms and definitions

For the purposes of this document, the following terms and definitions ~~given in ISO/IEC/IEEE 24765 and the following~~ apply.

ISO, IEC and IEEE maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at <https://www.iso.org/obp>

— IEC Electropedia: available at <https://www.electropedia.org/>

— IEEE Standards Dictionary Online: available at <https://dictionary.ieee.org>

NOTE For additional terms and definitions in the field of systems and software engineering, see ISO/IEC/IEEE 24765, which is published periodically as a “snapshot” of the SEVOCAB (Systems and software Engineering Vocabulary) and is publicly accessible at <https://www.computer.org/sevocab>.

~~NOTE The following terms and definitions are provided to assist with the understanding and readability of parts 1, 2, 3 and 4 of the ISO/IEC/IEEE 29119 Software Testing standards so some terms defined in ISO/IEC/IEEE 29119-1 will not be used in ISO/IEC/IEEE 29119-1 and will only be used in another standard in the ISO/IEC/IEEE 29119 series. Only terms critical to the understanding of these standards are included; this clause is not intended to provide a complete list of testing terms. The systems and software engineering Vocabulary ISO/IEC/IEEE 24765 should be referenced for terms not defined in this clause. This source is available at the following web site: <http://www.computer.org/sevocab>.~~

3.1

A/B testing

split-run testing

statistical *testing* (3.131) approach that allows testers to determine which of two systems or components performs better

3.2

accessibility testing

type of usability *testing* [\(3.131\)](#) used to measure the degree to which a *test item* [\(3.107\)](#) can be operated by users with the widest possible range of characteristics and ~~abilities~~*capabilities*

3.3

activation function

transfer function

<artificial intelligence (3.7)> formula associated with a node in a neural network (3.50) that determines the output of the node (activation value (3.4)) from the inputs to the neuron

3.4

activation value

<artificial intelligence (3.7)> output of an activation function (3.3) of a node in a neural network (3.50)

3.5

actual result ~~results~~

set of behaviours or conditions of a *test item* [\(3.107\)](#), or set of conditions of associated data or the *test environment* [\(3.95\)](#), observed as a result of *test execution* [\(3.99\)](#)

EXAMPLE Outputs to screen, outputs to hardware, changes to data, reports, and communication messages sent.

3.6

AI-based system

system including one or more components implementing AI (3.7)

3.7

artificial intelligence

AI

capability of an engineered system to acquire, process and apply knowledge and skills

3.8

autonomous system

<artificial intelligence (3.7)> system capable of working without human intervention for sustained periods

3.9

autonomy

<artificial intelligence (3.7)> ability of a system to work for sustained periods without human intervention

3.10

back-to-back testing

differential testing

approach to testing (3.131) whereby an alternative version of the system is used to generate *expected results* (3.35) for comparison from the same test inputs

EXAMPLE The alternative version can be a system that already exists, a system developed by an independent team or a system implemented using a different programming language.

3.11

base choice

base value

input parameter value that is normally selected based on being a representative or typical value for the parameter

3.12

boundary value analysis

specification-based *test design technique* (3.94) based on exercising the boundaries of *equivalence partitions* (3.30)

3.13

branch testing

structure-based *test case* (3.85) design technique based on exercising branches in the *control flow* (3.22) of the *test item* (3.107)

3.14

cause-effect graph

graphical representation of *decision rules* (3.25) between causes (inputs described as Boolean *conditions* (3.21)) and effects (outputs described as Boolean expressions)

3.15

cause-effect graphing

specification-based *test design technique* (3.94) based on exercising *decision rules* (3.25) in a *cause-effect graph* (3.14)

3.16

classification

<artificial intelligence (3.7)> machine learning (3.44) function that predicts the output class for a given input

3.17

classification tree

hierarchical tree model of the input data to a program in which the inputs are represented by distinct classifications (relevant test aspects) and classes (input values)

3.18

combinatorial testing

combinatorial test design techniques

class of specification-based test design techniques (3.94) based on exercising combinations of P - V pairs (3.56)

EXAMPLE Pairwise testing (3.57), base choice (3.11) testing.

4.3

~~backup and recovery testing~~

~~type of reliability testing that measures the degree to which system state can be restored from backup within specified parameters of time, cost, completeness, and accuracy in the event of failure~~

4.4

~~black box testing~~

~~see specification-based testing (4.39)~~

4.5

capacity testing

~~type of performance efficiency testing conducted to evaluate the level at which increasing load (of users, transactions, data storage, etc.) compromises a test item's ability to sustain required performance~~

3.19

compatibility testing

type of *testing* (3.131) that measures the degree to which a *test item* (3.107) can function satisfactorily alongside other independent products in a shared environment (co-existence), and where necessary, exchanges information with other systems or components (interoperability)

3.20

completion criteria

conditions under which the *testing* (3.131) activities are considered complete

3.21

condition

Boolean expression containing no Boolean operators

EXAMPLE "A < B" is a condition but "A and B" is not.

3.22

control flow

sequence in which operations are performed during the execution of a *test item* (3.107)

4.7

~~coverage item~~

~~see test coverage item (4.54)~~

3.23

decision

Type ~~types~~ of statement ~~statements~~ in which a choice between two or more possible outcomes controls which set of actions will result

Note 1 to entry: Typical decisions are simple selections (e.g. if-then-else), to decide when to exit loops (e.g. while-loop), and in case (switch) statements (e.g. case-1-2-3-... -N).

3.24

decision outcome

result of a *decision* (3.23) that determines the branch to be executed

3.25

decision rule

combination of *conditions* (3.21) (also known as causes) and actions (also known as effects) that produce a specific outcome in *decision table testing* (3.27) and *cause-effect graphing* (3.15)

3.26

decision table

tabular representation of *decision rules* (3.25) between causes (inputs described as Boolean *conditions* (3.21)) and effects (outputs described as Boolean expressions)

3.27

decision table testing

specification-based test design technique (3.94) based on exercising decision rules (3.25) in a decision table (3.26)

3.28

decision testing

structure-based test case (3.85) design technique based on exercising decision outcomes (3.24) in the control flow (3.22) of the test item (3.107)

3.29

dynamic testing

testing (3.131) in which a test item (3.107) is evaluated by executing it

~~testing that requires the execution of the test item~~

4.10

endurance testing

~~type of performance efficiency testing conducted to evaluate whether a test item can sustain a required load continuously for a specified period of time~~

3.30

equivalence partition

equivalence class

class subset of inputs the range of values of a variable, or outputs that are set of variables, within a test item or at its interfaces such that all the values in the partition can reasonably be expected to be treated similarly by the test item (3.107) i.e. they may be considered "equivalent") by the test item

4.12

equivalence partition coverage

~~proportion of identified equivalence partitions of a test item that are covered by a test set~~

~~Note 1 to entry: In many cases, the identification of equivalence partitions is subjective (especially in the sub-partitioning of "invalid" partitions), so a definitive count of the number of equivalence partitions in a test item could be impossible.~~

3.31

equivalence partitioning

test design technique [\(3.94\)](#) in which *test cases* [\(3.85\)](#) are designed to exercise *equivalence partitions* [\(3.30\)](#) by using one or more representative members of each partition

3.32

error guessing

test design technique [\(3.94\)](#) in which *test cases* [\(3.85\)](#) are derived on the basis of the tester's knowledge of past failures, or general knowledge of failure modes

Note 1 to entry: The relevant knowledge [can](#) ~~could~~ be gained from personal experience, or [can](#)~~might~~ be encapsulated in, for example, a defects database or a "bug taxonomy".

[3.33](#)

executable statement

statement which, when compiled, is translated into object code, which will be executed procedurally when the *test item* [\(3.107\)](#) is running and may perform an action on program data

3.34

exhaustive testing

test approach (3.83) in which all combinations of input values and preconditions are tested

Note 1 to entry: In nearly all non-trivial situations, exhaustive testing is impossible, due to the large number of possible tests.

3.35

expected result ~~results~~

observable predicted behaviour of the *test item* (3.107) under specified conditions based on its specification or another source

3.36

experience-based testing

class of *test case* (3.85) design techniques based on using the experience of testers to generate test cases

EXAMPLE *Error guessing* (3.32).

Note 1 to entry: Experience-based testing can include concepts such as test attacks, tours, and error taxonomies which target potential problems such as security, performance, and other quality areas.

3.37

exploratory testing

experience-based testing (3.36) in which the tester spontaneously designs and executes tests based on the tester's existing relevant knowledge, prior exploration of the *test item* (3.107) (including the results of previous tests), and heuristic “rules of thumb” regarding common software behaviours and types of failure

Note 1 to entry: Exploratory testing hunts for hidden properties (including hidden behaviours) that, while quite possibly benign by themselves, can ~~could~~ interfere with other properties of the software under test, and so constitute a risk that the software will fail.

3.38

fuzz testing

<artificial intelligence (3.7)> software testing (3.131) approach in which high volumes of random (or near random) data, called fuzz, are used to generate inputs to the test item (3.107)

3.39

incident

anomalous or unexpected event, set of events, condition, or situation at any time during the life cycle of a project, product, service, or system

~~4.17~~

~~feature set~~

~~collection of items which contain the test conditions of the test item to be tested which can be collected from risks, requirements, functions, models, etc.~~

~~Note 1 to entry: This could be the set of all features for the item (its full feature set), or a subset identified for a specific purpose (the functional feature set etc.).~~

3.40

incident report

documentation of the occurrence, nature, and status of an *incident* (3.39)

Note 1 to entry: Incident reports are also known as anomaly reports, bug reports, defect reports, error reports, issues, problem reports and trouble reports, amongst other terms.

3.41

keyword

<keyword-driven testing (3.42)> one or more words used as a reference to a specific set of actions intended to be performed during the execution of one or more *test cases* (3.85)

Note 1 to entry: The actions include interactions with the User Interface during the test, verification, and specific actions to set up a *test scenario* (3.123).

Note 2 to entry: Keywords are named using at least one verb.

Note 3 to entry: Composite keywords can be constructed based on other keywords.

3.42

keyword-driven testing

testing (3.131) using *test cases* (3.85) composed from *keywords* (3.41)

4.19

installability testing

~~type of portability testing conducted to evaluate whether a test item or set of test items can be installed as required in all specified environments~~

3.43

load testing

type of *performance* ~~efficiency~~ *testing* (3.58) conducted to evaluate the behaviour of a *test item* (3.107) under anticipated conditions of varying load, usually between anticipated conditions of low, typical, and peak usage

3.44

machine learning

ML

process using computational techniques to enable systems to learn from data or experience

3.45

maintainability testing

test type [\(3.130\)](#) conducted to evaluate the degree of effectiveness and efficiency with which a *test item* [\(3.107\)](#) may be modified

3.46

manual testing

humans performing tests by entering information into a *test item* [\(3.107\)](#) and verifying the results

Note 1 to entry: Automated *testing* [\(3.131\)](#) uses tools, *robots* [\(3.70\)](#), and other *test execution engines* [\(3.10 0\)](#) to perform tests. Manual testing does not use these items.

3.47

MC/DC testing

modified condition decision testing

structure-based *test case* [\(3.85\)](#) design technique based on demonstrating that a single Boolean *condition* [\(3.21\)](#) within a *decision* [\(3.23\)](#) can independently affect the outcome of the decision

3.48

metamorphic relation

description of how changes to the test inputs for a *test case* (3.85) affect the expected outputs based on the required behaviour of a *test item* (3.107)

3.49

metamorphic testing

specification-based *test case* (3.85) design technique based on generating test cases based on existing test cases and *metamorphic relations* (3.48)

3.50

neural network

artificial neural network

<*artificial intelligence* (3.7)> network of primitive processing elements connected by weighted links with adjustable weights, in which each element produces a value by applying a nonlinear function to its input values, and transmits it to other elements or presents it as an output value

Note 1 to entry: Whereas some neural networks are intended to simulate the functioning of neurons in the nervous system, most neural networks are used in artificial intelligence as realizations of the connectionist model.

Note 2 to entry: Examples of nonlinear functions are a threshold function, a sigmoid function, and a polynomial function.

[SOURCE: ISO/IEC 2382:2015, 2120625, modified — The admitted term “neural net” has been removed; notes 3 to 5 to entry have been removed.]

3.51

neuron coverage

<artificial intelligence (3.7)> proportion of activated neurons divided by the total number of neurons in the neural network (3.50) (normally expressed as a percentage) for a set of tests

Note 1 to entry: A neuron is considered to be activated if its *activation value* (3.4) exceeds zero.

3.52

non-deterministic system

system which, given a particular set of inputs and starting state, will not always produce the same set of outputs and final state

3.53

organizational test practices

documentation that expresses the recommended approaches or methods for the *testing* (3.131) to be performed within an organization, providing detail on how the testing is to be performed

Note 1 to entry: The organizational test practices are aligned with the *organizational test policy* (3.118).

Note 2 to entry: An organization can have more than one organizational test practices document to cover markedly different contexts, such one for mobile apps and one for *safety* (3.71) critical systems.

Note 3 to entry: The organizational test practices can incorporate the context of the test policy where no separate test policy is available

4.22

Organizational Test Policy

~~an executive level document that describes the purpose, goals, and overall scope of the testing within an organization, and which expresses why testing is performed and what it is expected to achieve~~

~~Note 1 to entry: It is generally preferable to keep the Organizational Test Policy as short as possible in a given context.~~

3.54

organizational test process

test process [\(3.121\)](#) for developing and managing *organizational test specifications* [\(3.55\)](#)

3.55

organizational test specification

document that provides information about *testing* [\(3.131\)](#) for an organization, i.e. information that is not project specific

EXAMPLE The most common examples of organizational test specifications are [the](#) *organizational test policy* [\(3.118\)](#) and [the](#) *organizational test practices* [\(3.53\)](#) ~~Strategy~~.

[3.56](#)

[P-V pair](#)

[parameter-value pair](#)

[combination of a test item \(3.107\) parameter with a value assigned to that parameter, used as a test coverage item \(3.90\)](#)

3.57

pairwise testing

black-box test design technique (3.94) in which test cases (3.85) are designed to execute all possible discrete combinations of each pair of input parameters

Note 1 to entry: Pairwise testing is the most popular form of combinatorial testing (3.18).

4.25

~~Organizational Test Strategy~~

~~document that expresses the generic requirements for the testing to be performed on all the projects run within the organization, providing detail on how the testing is to be performed~~

~~Note 1 to entry: The Organizational Test Strategy is aligned with the Organizational Test Policy.~~

~~Note 2 to entry: An organisation could have more than one Organizational Test Strategy to cover markedly different project contexts.~~

4.26

~~pass/fail criteria~~

~~decision rules used to determine whether a test item, or feature of a test item, has passed or failed after testing~~

3.58

performance testing

type of *testing* (3.131) conducted to evaluate the degree to which a *test item* (3.107) accomplishes its designated functions within given constraints of time and other resources

3.59

portability testing

type of *testing* [\(3.131\)](#) conducted to evaluate the ease with which a *test item* [\(3.107\)](#) can be transferred from one hardware or software environment to another, including the level of modification needed for it to be executed in various types of environment

3.60

procedure testing

type of functional suitability *testing* [\(3.131\)](#) conducted to evaluate whether procedural instructions for interacting with a *test item* [\(3.107\)](#) or using its outputs meet user requirements and support the purpose of their use

3.61

product risk

risk that a product [can](#) ~~may~~ be defective in some specific aspect of its function, quality, or structure

3.62

project risk

risk related to the management of a project

EXAMPLE Lack of staffing, strict deadlines, changing requirements.

[3.63](#)

[random testing](#)

[specification-based *test design technique* \(3.94\) based on generating *test cases* \(3.85\) to exercise randomly selected *test item* \(3.107\) inputs](#)

3.64

regression testing

testing (3.131) performed following modifications to a *test item* (3.107) or to its operational environment, to identify whether ~~regression~~ failures in unmodified parts of the test item occur

Note 1 to entry: Regression testing differs from *retesting* (3.68) in that it does not test that the modification works correctly, but that other parts of the system have not been accidentally affected by the change.

Note 2 + to entry: The adequacy ~~sufficiency~~ of a set of regression *test cases* (3.85) depends on the item under test and on the modifications to that item or its operational environment.

3.65

regulatory standard

standard promulgated by a regulatory agency

3.66

reliability testing

type of *testing* (3.131) conducted to evaluate the ability of a *test item* (3.107) to perform its required functions, including evaluating the frequency with which failures occur, when it is used under stated conditions for a specified period of time

3.67

requirements-based testing

specification-based *test case* (3.85) design technique based on exercising atomic requirements

EXAMPLE An atomic requirement can be ‘The system shall collect and store the date of birth of all registered users.’

3.68

retesting

confirmation testing

testing (3.131) performed to check that modifications made to correct a fault have successfully removed the fault

Note 1 to entry: When retesting is performed often it is also complemented by *regression testing* (3.64), to help ensure that other unmodified parts of the *test item* (3.107) have not been accidentally adversely affected by the modifications.

~~re-execution of test cases that previously returned a "fail" result, to evaluate the effectiveness of intervening corrective actions~~

~~Note 1 to entry: Also known as confirmation testing.~~

3.69

risk-based testing

testing (3.131) in which the management, selection, prioritization ~~prioritisation~~, and use of testing activities and resources are consciously based on corresponding types and levels of analysed ~~analyzed~~ risk

3.70

robot

<artificial intelligence (3.7)> programmed actuated mechanism with a degree of *autonomy* (3.9), moving within its environment, to perform intended tasks

Note 1 to entry: A robot includes the control system and interface of the control system.

Note 2 to entry: The classification of robot into industrial robot or service robot is done according to its intended application.

3.71

safety

expectation that a system does not, under defined conditions, lead to a state in which human life, health, property, or the environment is endangered

[SOURCE: ISO/IEC/IEEE 12207:2017, 3.1.48]

3.72

scenario testing

specification-based *test case* (3.85) design technique based on exercising sequences of interactions between the *test item* (3.107) and other systems

Note 1 to entry: Users are considered to be other systems in this context.

~~class of test design technique in which tests are designed to execute individual scenarios~~

~~Note 1 to entry: A scenario can be a user story, use case, operational concept, or sequence of events the software may encounter etc.~~

3.73

scripted testing

testing (3.131) performed based on a documented *test script* (3.124)

~~dynamic testing in which the tester's actions are prescribed by written instructions in a test case~~

Note 1 to entry: This term normally applies to manually executed testing, rather than the execution of an automated script.

3.74

security testing

test type (3.130) ~~of testing~~ conducted to evaluate the degree to which a *test item* (3.107), and associated data and information, are protected so that unauthorized persons or systems cannot use, read, or modify them, and authorized persons or systems are not denied access to them

3.75

specification-based testing

black-box testing

closed box testing

testing (3.131) in which the principal *test basis* (3.84) is the external inputs and outputs of the *test item* (3.107), commonly based on a specification, rather than its implementation in source code or executable software

~~Note 1 to entry: Synonyms for specification-based testing include black box testing and closed box testing.~~

3.76

state transition testing

specification-based *test case* (3.85) design technique based on exercising transitions in a state model

EXAMPLE Example state models are state transition diagram and state table.

4.40

~~statement coverage~~

~~percentage of the set of all executable statements of a test item that are covered by a test set~~

3.77

statement testing

~~test design technique in which test cases are constructed to force execution of individual statements in a test item~~

structure-based *test case* (3.85) design technique based on exercising *executable statements* (3.33) in the source code of the *test item* (3.107)

3.78

static testing

testing [\(3.131\)](#) in which a *test item* [\(3.107\)](#) is examined against a set of quality or other criteria without [the test item](#) ~~code~~ being executed

EXAMPLE Reviews, static analysis.

3.79

stress testing

type of performance efficiency *testing* [\(3.131\)](#) conducted to evaluate a *test item's* [\(3.107\)](#) behaviour under conditions of loading above anticipated or specified capacity requirements, or of resource availability below minimum specified requirements

4.44

~~structural testing~~

~~see structure-based testing (4.45)~~

3.80

structure-based testing

[white box testing](#)

[glass-box testing](#)

[structural testing](#)

dynamic testing [\(3.29\)](#) in which the tests are derived from an examination of the structure of the *test item* [\(3.107\)](#)

Note 1 to entry: Structure-based testing is not restricted to use at component level and can be used at all levels, e.g. menu item coverage as part of a system test.

Note 2 to entry: Techniques include *branch testing* [\(3.13\)](#), *decision testing* [\(3.28\)](#), and *statement testing* [\(3.77\)](#).

~~Note 3 to entry: Synonyms for structure-based testing are structural testing, glass box testing, and white box testing.~~

3.81

suspension criteria

criteria used to (temporarily) stop all or a portion of the *testing* [\(3.131\)](#) activities

[3.82](#)

[test](#)

[activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component](#)

3.83

test approach

high-level test implementation choice, typically made as part of the *test strategy* (3.127) design activity

EXAMPLE 1 The use of model-based *testing* (3.131) for the functional system testing.

EXAMPLE 2 Typical choices made as test approaches are *test level* (3.10 8), *test type* (3.130), *test technique* (3.94), *test practice* (3.119) and the form of *static testing* (3.78) to be used.

3.84

test basis

information ~~body of knowledge~~ used as the basis for designing ~~the design of tests~~ and implementing *test cases* (3.85)

Note 1 to entry: The test basis may take the form of documentation, such as a requirements specification, design specification, or module specification, but may also be an undocumented understanding of the required behaviour.

3.85

test case

set of ~~test case~~ preconditions, inputs ~~(including actions, where applicable)~~, and *expected results* (3.35), developed to drive the execution of a *test item* (3.107) to meet *test objectives* (3.114), ~~including correct implementation, error identification, checking quality, and other valued information~~

Note 1 to entry: A test case is the lowest level of test implementation documentation ~~input~~ (i.e. test cases are not made up of test cases) for the *test level* (3.10 8) or *test type* (3.130) ~~sub-process~~ for which it is intended.

Note 2 to entry: Test case preconditions include [the required state of the test environment \(3.95\)](#), ~~existing~~ data (e.g. databases), ~~used by the software under~~ test [item, and the test item itself, hardware, etc.](#)

Note 3 to entry: Inputs are the data information [and actions, where applicable](#), used to drive *test execution* [\(3.9.9\)](#).

~~Note 4 to entry: Expected results include success criteria, failures to check for, etc.~~

4.49

Test Case Specification

~~documentation of a set of one or more test cases~~

3.86

test completion process

test management process [\(3.110\)](#) that [aims to ensure](#) ~~for ensuring~~ that useful test assets are made available for later use, *test environments* [\(3.95\)](#) are left in a satisfactory condition, and the results of *testing* [\(3.131\)](#) are recorded and communicated to relevant stakeholders

3.87

test completion report

[test summary report](#)

report that provides a summary of the *testing* [\(3.131\)](#) that was performed

~~Note 1 to entry: Also known as a Test Summary Report.~~

3.88

test condition

testable aspect of a component or system, such as a function, transaction, feature, quality attribute, or structural element identified as a basis for *testing* (3.131)

Note 1 to entry: The ISO/IEC/IEEE 29119 series does not use the concept of test conditions, but instead uses the concept of a *test model* (3.111) for test design~~can be used to derive coverage items, or can themselves constitute coverage items.~~

3.89

test coverage

degree, expressed as a percentage, to which specified *test coverage items* (3.90) have been exercised by a *test case* (3.85) or test cases

3.90

test coverage item

coverage item

measurable attribute of a *test item* (3.107) that is the focus of *testing* (3.131)

EXAMPLE *Equivalence partitions* (3.30), *transitions between states*, *executable statements* (3.33).

~~attribute or combination of attributes that is derived from one or more test conditions by using a test design technique that enables the measurement of the thoroughness of the test execution~~

3.91

test data

data created or selected to satisfy the input requirements for executing one or more *test cases* (3.85);
~~which may be defined in the Test Plan, test case or test procedure~~

Note 1 to entry: Test data ~~can~~ ~~could~~ be stored within the ~~product under test item~~ (3.107) (e.g. in arrays ~~or~~ flat files, ~~or a database~~), or ~~can come~~ ~~could be available~~ from ~~or supplied by~~ external sources, such as other systems, ~~other system components~~, hardware devices, or human operators.

3.92

test data readiness report

document describing the status of each *test data* (3.91) requirement

3.93

test design and implementation process

test process (3.121) for deriving and specifying *test cases* (3.85) and *test procedures* (3.120)

4.58

Test Design Specification

~~document specifying the features to be tested and their corresponding test conditions~~

3.94

test design technique

test technique

procedure used to create or select a *test model* (3.111), identify *test coverage items* (3.90), and derive corresponding *test cases* (3.85)

EXAMPLE *Equivalence partitioning* (3.31), *boundary value analysis* (3.12), *decision table testing* (3.27), *branch testing* (3.13).

Note 1 to entry: The test design technique is typically used to achieve a required level of *test coverage* (3.89).

Note 2 to entry: Some *test practices* (3.119), such as *exploratory testing* (3.37) or *model-based testing* (3.131) are sometimes referred to as “test techniques”. Following the definition in the ISO/IEC/IEEE 29119 series, they are not test design techniques as they are not themselves providing a way to create test cases, but instead use test design techniques to achieve that.

~~activities, concepts, processes, and patterns used to construct a test model that is used to identify test conditions for a test item, derive corresponding test coverage items, and subsequently derive or select test cases~~

3.95

test environment

environment containing facilities, hardware, software, firmware, procedures, needed to conduct a test ~~and documentation intended for or used to perform testing of software~~

Note 1 to entry: A test environment can ~~could~~ contain multiple environments to accommodate specific *test level* (3.10 8) or *test types* (3.130) ~~sub-processes~~ (e.g. a unit test environment, a performance test environment ~~etc.~~).

Note 2 to entry: A test environment can comprise several interconnected systems or virtual environments.

3.96

test environment and data management process

test process (3.121) for establishing and maintaining a required *test environment* (3.95) and corresponding test data (3.91)

3.97

test environment readiness report

document that describes the status of ~~the each~~ *test environment* (3.95) ~~requirement~~

Note 1 to entry: This can list the status of each of the *test environment requirements* (3.98).

3.98

test environment requirements

description of the necessary properties of the *test environment* (3.95)

Note 1 to entry: All or parts of the test environment requirements can ~~could~~ reference where the information can be found, e.g. in the ~~appropriate~~ *organizational test practices* (3.53) document, *test plan* (3.117), ~~Strategy, Test Plan,~~ and ~~/or~~ *test specification* (3.125).

4.63

~~Test Environment Set-up Process~~

~~dynamic test process for establishing and maintaining a required test environment~~

3.99

test execution

process of running a *test* (3.82) on the *test item* (3.107), producing *actual result(s)* (3.5)

3.100

test execution engine

<keyword-driven testing (3.42)> tool implemented in software and sometimes in hardware that can manipulate the *test item* (3.107) to execute *test cases* (3.85)

Note 1 to entry: A typical test execution engine includes unit test tool frameworks, stimulation-command systems, capture and playback tools or hardware *robots* (3.70) along with the software to control them.

3.101

test execution log

record ~~document that records details~~ of the execution of one or more *test procedures* [\(3.120\)](#)

3.102

test execution process

dynamic *test process* [\(3.121\)](#) for executing *test procedures* [\(3.120\)](#) created in the *test design and implementation process* [\(3.93\)](#) in the prepared *test environment* [\(3.95\)](#), and recording the results

3.103

test framework

structured set of principles, guidelines, and practices used for organizing, selecting and communicating *testing* [\(3.131\)](#)

3.104

test incident

event occurring during the execution of a *test* [\(3.82\)](#) that requires investigation

3.105

test incident reporting process

dynamic *test process* [\(3.121\)](#) for reporting *incidents* [\(3.39\)](#) ~~to the relevant stakeholders issues~~ requiring further action that were identified during the *test execution process* [\(3.102\)](#) to the relevant stakeholders

3.106

test independence

degree to which those performing *testing* (3.131) have separate responsibilities from those developing the *test item* (3.107)

3.107

test item

test object

work product to be tested ~~that is an object of testing~~

EXAMPLE Software component, ~~A~~ system, ~~a software item~~, ~~a~~ requirements document, ~~a~~ design specification, ~~a~~ user guide.

3.108

test level

one of a sequence of test stages each of which is typically associated with the achievement of particular objectives and used to treat particular risks

~~specific instantiation of a test sub-process~~

EXAMPLE The following are common test levels, listed sequentially: unit/component *testing* (3.131), ~~that can be instantiated as test sub-processes: component test level/sub-process,~~ integration *testing*, system *testing* ~~test level/sub-process,~~ system integration *testing* ~~test level/sub-process,~~ acceptance *testing* ~~test level/sub-process.~~

Note 1 to entry: It is not always necessary for a *test item* (3.107) to be tested at all test levels, but the sequence of test levels generally stays the same ~~Test levels are synonymous with test phases.~~

Note 2 to entry: Typical objectives can include consideration of basic functionality for unit/component testing, interaction between integrated components for integration testing, acceptability to end users for acceptance testing.

3.109

test management

planning, scheduling, estimating, monitoring, reporting, control, and completion of test activities

3.110

test management process

process used to coordinate, monitor, and control testing (3.131)

~~test process containing the sub-processes that are required for the management of a test project~~

Note 1 to entry: See test strategy and planning process (3.128), test monitoring and control process (3.113), test completion process (3.86).

3.111

test model

representation of the test item (3.107), which allows the testing (3.131) to be focused on particular characteristics or qualities

EXAMPLE Requirements statements, of equivalence partitions (3.30), state transition diagram, use case description, decision table (3.26), input syntax description, source code, control flow (3.22) graph, parameters and values, classification tree (3.17), natural language.

Note 1 to entry: The test model and the required test coverage (3.89) are used to identify test coverage items (3.90).

Note 2 to entry: A separate test model can be required for each different type of required test coverage included in the test *completion criteria* (3.20).

Note 3 to entry: A test model can include one or more *test conditions* (3.88).

Note 4 to entry: Test models are commonly used to support test design (e.g. they are used to support test design in ISO/IEC/IEEE 29119-4, and they are used in model-based testing). Other types of models exist to support other aspects of testing, such as *test environment* (3.95) models, test maturity models and test architecture models.

3.112

test model specification

document specifying the *test model* (3.111)

3.113

test monitoring and control process

test management process (3.110) that aims to ensure ~~for ensuring~~ that *testing* (3.131) is performed in line with a *test plan* (3.117) and with *organizational test specifications* (3.55)

3.114

test objective

reason for performing *testing* (3.131)

EXAMPLE Checking for correct implementation, identification of defects, measuring quality.

3.115

test oracle

source of information for determining whether a test has passed or failed

Note 1 to entry: The test oracle is often a specification used to generate *expected results* (3.35) for individual *test cases* (3.85), but other sources may be used, such as comparing *actual results* (3.5) with those of another similar program or system or asking a human expert.

4.73

test object

~~see test item (4.68)~~

4.74

test phase

~~specific instantiation of test sub-process~~

~~Note 1 to entry: Test phases are synonymous with test levels, therefore examples of test phases are the same as for test levels (e.g. system test phase/sub-process).~~

3.116

test oracle problem

challenge of determining whether a *test* (3.82) has passed or failed for a given set of test inputs and state

3.117

test plan

detailed description of *test objectives* (3.114) to be achieved and the means and schedule for achieving them, organized ~~organised~~ to coordinate *testing* (3.131) activities for some *test item* (3.107) or set of test items

Note 1 to entry: A project can have more than one test plan, for example there ~~can~~ ~~could~~ be a project test plan (also known as a master test plan) that encompasses all testing activities on the project; further detail of particular test activities ~~can~~ ~~could~~ be defined in one or more *test level (3.108)/test type (3.130) sub-process* plans (~~i.e.g.~~ a system test plan or a performance test plan).

Note 2 to entry: ~~Typically~~ A test plan is typically a written document, although ~~though~~ other ~~plan~~ formats ~~can~~ ~~could~~ be possible as defined locally within an organization or project.

Note 3 to entry: Test plans ~~can~~ ~~could~~ also be written for non-project activities, for example a maintenance test plan.

3.118

test policy

organizational test policy

executive-level document that describes the purpose, goals, principles, and scope of testing (3.131) within an organization

Note 1 to entry: The test policy defines what testing is performed and what it is expected to achieve but does not detail how testing is to be performed.

Note 2 to entry: The test policy can provide a framework for establishing, reviewing, and continually improving the organization's testing.

4.76

Test Planning Process

~~Test Management Process used to complete test planning and develop Test Plans~~

3.119

test practice

conceptual framework that can be applied to the *organizational test process* (3.54), the *test management process* (3.110), and/or the *dynamic test process* (3.121) to facilitate *testing* (3.131)

~~Note 1 to entry: Test Practices are sometimes referred to as test approaches.~~

3.120

test procedure

sequence of *test cases* (3.85) in execution order, ~~with~~ any associated actions ~~that may be~~ required to set up ~~the initial~~ preconditions and perform ~~any~~ wrap up activities post execution

~~Note 1 to entry: Test procedures include detailed instructions for how to run a set of one or more test cases selected to be run consecutively, including set up of common preconditions, and providing input and evaluating the actual result for each included test case.~~

4.79

Test Procedure Specification

~~document specifying one or more test procedures, which are collections of test cases to be executed for a particular objective~~

~~Note 1 to entry: The test cases in a test set are listed in their required order in the test procedure.~~

~~Note 2 to entry: Also known as a manual test script. A test procedure specification for an automated test run is usually called a test script.~~

3.121

test process

set of *testing* (3.131) activities performed to achieve a *test objective* (3.114)

~~provides information on the quality of a software product, often comprised of a number of activities, grouped into one or more test sub-processes~~

Note 1 to entry: ~~EXAMPLE~~ The test process for a particular project can ~~may well~~ consist of multiple *test levels* (3.10 8) and *test types* (3.130) ~~sub-processes, e.g. a system test sub-process, a Test Planning sub-process (a part of the larger Test Management Process) or a static testing sub-process.~~

4.81

test requirement

~~see test condition (4.52)~~

3.122

test result

indication of whether or not a specific *test case* (3.85) has passed or failed, i.e. if the *actual results* (3.5) ~~result observed as test item output~~ corresponds to the *expected results* (3.35) ~~result~~ or if deviations were observed

3.123

test scenario

situation or setting for a *test item* (3.107) used as the basis for generating *test cases* (3.85)

3.124

test script

test procedure specification [document specifying one or more test procedures \(3.120\)](#) ~~for manual or automated testing~~

4.84

~~test set~~

~~set of one or more test cases with a common constraint on their execution~~

~~EXAMPLE A specific test environment, specialized domain knowledge or specific purpose.~~

3.125

test specification

complete documentation of the test design, *test cases* [\(3.85\)](#) and *test procedures* [\(3.120\)](#) for a specific *test item* [\(3.107\)](#)

Note 1 to entry: A test specification [can](#) ~~could~~ be detailed in one document, in a set of documents, or in other ways, for example in a mixture of documents and database entries.

3.126

test status report

report that provides information about the status of the *testing* [\(3.131\)](#) that is being performed in a specified reporting period

3.127

test strategy

part of the *test plan* [\(3.117\)](#) that describes the approach to *testing* [\(3.131\)](#) for a specific ~~test~~ project, *test level* [\(3.108\)](#) or *test type* [\(3.130\)](#) ~~sub-process or sub-processes~~