INTERNATIONAL STANDARD

ISO/ IEC/IEEE 29119-1

Second edition
2022-01

Software and systems engineering — Software testing —

Part 1: General concepts

Ingénierie du logiciel et des systèmes — Essais du logiciel —

Partie 1: Concepts généraux

Reference number
ISO/IEC/IEEE 29119-1:2022(E)

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

Contents

Page

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 good reasons

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

1 Scope

This document specifies general concepts in software testing and presents key concepts for the ISO/IEC/IEEE 29119 series.

2 Normative references

There are no normative references in this document.

3 Terms and definitions

For the purposes of this document, the following terms and definitions 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.

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

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
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.

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)

3.23
decision
type of statement 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

3.30
equivalence partition
equivalence class
class of inputs or outputs that are expected to be treated similarly by the *test item* (3.107)

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 be gained from personal experience, or can 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
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 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

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)

3.43
load testing
type of *performance 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* (3.100) 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

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).

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).

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 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