INTERNATIONAL STANDARD

ISO/ IEC/IEEE 29119-4

Second edition
2021-10

Software and systems engineering — Software testing —

Part 4: Test techniques

Ingénierie du logiciel et des systèmes — Essais du logiciel —

Partie 4: Techniques d'essai

Reference number
ISO/IEC/IEEE 29119-4:2021(E)

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 29119-4 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 29119-4:2015), which has been technically revised.

The main changes compared to the previous edition are as follows:

— The test techniques in this document are defined using a new form of the test design and implementation process from ISO/IEC/IEEE 29119-2. In the first version, this process was based on the use of test conditions. Feedback on use of the previous edition highlighted a problem with users' understanding of test conditions and their use for deriving test cases. This second edition has replaced the use of test conditions with test models. Annex H provides more detail on this change and the Introduction describes the new process.

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 this document is to provide an International Standard that defines software test design techniques (also known as test case design techniques or test methods) that can be used within the test design and implementation process that is defined in ISO/IEC/IEEE 29119-2. This document does not describe a process for test design and implementation. The intent is to describe a series of techniques that have wide acceptance in the software testing industry. This document is originally based on the British standard, BS 7925-2. Annex G provides a mapping from the requirements of BS 7925-2 to the clauses and subclauses of this document.

The test design techniques presented in this document can be used to derive test cases that, when executed, generate evidence that test item requirements have been met or that defects are present in a test item (i.e. that requirements have not been met). Risk-based testing can be used to determine the set of techniques that are applicable in specific situations (risk-based testing is covered in ISO/IEC/IEEE 29119-1 and ISO/IEC/IEEE 29119-2).

NOTE A "test item" is a work product that is being tested (see ISO/IEC/IEEE 29119-1).

EXAMPLE 1 "Test items" include systems, software items, objects, classes, requirements documents, design specifications, and user guides.

Each technique follows the test design and implementation process that is defined in ISO/IEC/IEEE 29119-2 and shown in Figure 1.

Of the activities in this process, this document provides guidance on how to implement the following activities in detail for each technique that is described:

— create test model (TD1);

— identify test coverage items (TD2);

— derive test cases (TD3).

A test model represents testable aspects of a test item, such as a function, transaction, feature, quality attribute, or structural element identified as a basis for testing. The test model reflects the required test completion criterion in the test strategy.

EXAMPLE 2 If a test completion criterion for state transition testing was identified that required coverage of all states then the test model would show the states the test item can be in.

Test coverage items are attributes of the test model that can be covered during testing. A single test model will typically be the basis for several test coverage items.

A test case is a set of preconditions, inputs (including actions, where applicable), and expected results, developed to determine whether or not the covered part of the test item has been implemented correctly.

Specific (normative) guidance on how to implement the create test procedures activity (TD4) in the test design and implementation process of ISO/IEC/IEEE 29119-2 is not included in Clauses 5 or 6 because the process is the same for all techniques.

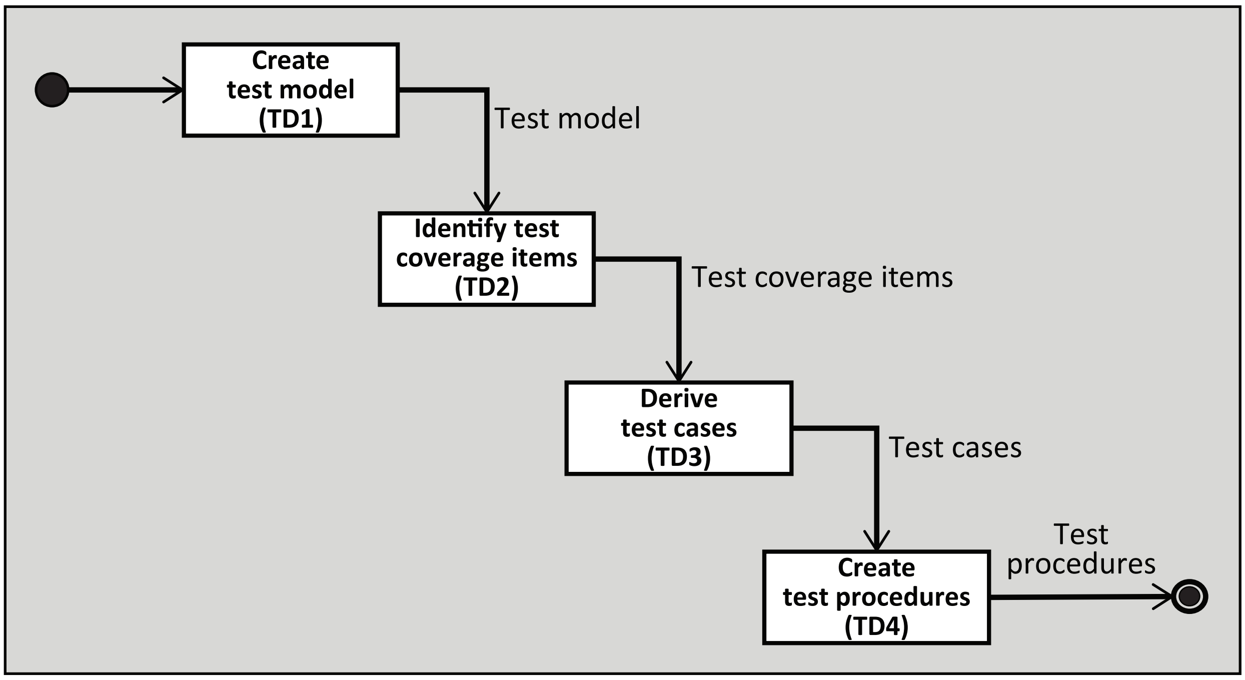

Figure 1 — ISO/IEC/IEEE 29119-2 test design and implementation process

ISO/IEC 25010 defines eight quality characteristics (including functional suitability) that can be used to identify types of testing that may be applicable for testing a specific test item. Annex A provides example mappings of test design techniques that apply to testing quality characteristics defined in ISO/IEC 25010.

Experience-based testing practices like exploratory testing and other test practices such as model-based testing are not defined in this document because this document only describes techniques for designing test cases. Test practices such as exploratory testing, which can use the test techniques defined in this document, are described in ISO/IEC/IEEE 29119-1.

Templates and examples of test documentation that are produced during the testing process are defined in ISO/IEC/IEEE 29119-3. The test techniques in this document do not describe how to document test cases (e.g. they do not include information or guidance on assigning unique identifiers, test case descriptions, priorities, traceability or pre-conditions to test cases). Information on how to document test cases can be found in ISO/IEC/IEEE 29119-3.

This document aims to provide stakeholders with the ability to design test cases for software testing in any organization.

Software and systems engineering — Software testing —

Part 4: Test techniques

1 Scope

This document defines test design techniques that can be used during the test design and implementation process that is defined in ISO/IEC/IEEE 29119-2.

Each technique follows the test design and implementation process that is defined in ISO/IEC/IEEE 29119-2 and shown in Figure 1. This document is intended for, but not limited to, testers, test managers, and developers, particularly those responsible for managing and implementing software testing.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC/IEEE 29119-2, *Software and systems engineering — Software testing — Part 2: Test processes*

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://ieeexplore.ieee.org/xpls/dictionary.jsp

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) database and is publicly accessible at https://www.computer.org/sevocab.

3.1
Backus-Naur Form
formal meta-language used for defining the syntax of a language in a textual format

3.2
base choice
base value
input parameter value used in ‘base choice testing’ that is normally selected based on being a representative or typical value for the parameter

3.3
boundary value analysis
specification-based *test case* (3.49) design technique based on exercising the boundaries of *equivalence partitions* (3.28)

3.4
branch condition combination testing
structure-based *test case* (3.49) design technique based on exercising combinations of Boolean values of *conditions* (3.13) within a decision

3.5
branch condition testing
structure-based *test case* (3.49) design technique based on exercising Boolean values of the *conditions* (3.13) within decisions and the *decision outcomes* (3.21)

3.6
branch testing
structure-based *test case* (3.49) design technique based on exercising branches in the *control flow* (3.14) of the *test item* (3.52)

3.7
c-use
computation data use
use of the value of a variable in any type of statement

3.8
cause-effect graph
graphical representation of *decision rules* (3.22) between causes (inputs described as Boolean conditions) and effects (outputs described as Boolean expressions)

3.9
cause-effect graphing
specification-based *test case* (3.49) design technique based on exercising *decision rules* (3.22) in a *cause-effect graph* (3.8)

3.10
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)

3.11
classification tree method
specification-based *test case* (3.49) design technique based on exercising classes in a *classification tree* (3.10)

3.12
combinatorial test design techniques
class of specification-based *test case* (3.49) design techniques based on exercising combinations of *P-V pairs* (3.38)

EXAMPLE All combinations *testing* (3.57), pair-wise testing, each choice testing, base choice testing

3.13
condition
Boolean expression containing no Boolean operators

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

3.14
control flow
sequence in which operations are performed during the execution of a *test item* (3.52)

3.15
control flow sub-path
sequence of *executable statements* (3.30) within a test *item* (3.52)

3.16
data definition
variable definition
statement where a variable is assigned a value

3.17
data definition c-use pair
data definition (3.16) and subsequent *computation data use* (3.7), where the *data use* (3.20) uses the value defined in the data definition

3.18
data definition p-use pair
data definition (3.16) and subsequent *predicate data use* (3.37), where the *data use* (3.20) uses the value defined in the data definition

3.19
data flow testing
class of structure-based *test case* (3.49) design techniques based on exercising *definition-use pairs* (3.26)

EXAMPLE All-definitions *testing* (3.57), all-c-uses testing, all-p-uses testing, all-uses testing, all-du-paths testing.

3.20
data use
executable statement (3.30) where the value of a variable is accessed

3.21
decision outcome
result of a decision that determines the branch to be executed

3.22
decision rule
combination of *conditions* (3.13) (also known as causes) and actions (also known as effects) that produce a specific outcome in *decision table testing* (3.24) and *cause-effect graphing* (3.9)

3.23
decision table
tabular representation of *decision rules* (3.22) between causes (inputs described as Boolean *conditions* (3.13)) and effects (outputs described as Boolean expressions)

3.24
decision table testing
specification-based *test case* (3.49) design technique based on exercising *decision rules* (3.22) in a *decision table* (3.23)

3.25
decision testing
structure-based *test case* (3.49) design technique based on exercising *decision outcomes* (3.21) in the *control flow* (3.14) of the *test item* (3.52)

3.26
definition-use pair
data definition-use pair
data definition (3.16) and subsequent *predicate* (3.40) or computational *data use* (3.20), where the data use uses the value defined in the data definition

3.27
entry point
point in a *test item* (3.52) at which execution of the test item can begin

Note 1 to entry: An entry point is an *executable statement* (3.30) within a test item that can be selected by an external process as the starting point for one or more *paths* (3.39) through the test item. It is most commonly the first executable statement within the test item.

3.28
equivalence partition
equivalence class
class of inputs or outputs that are expected to be treated similarly by the *test item* (3.52)

3.29
equivalence partitioning
specification-based *test case* (3.49) design technique based on exercising *equivalence partitions* (3.28)

3.30
executable statement
statement which, when compiled, is translated into object code, which will be executed procedurally when the *test item* (3.52) is running and may perform an action on program data

3.31
exit point
last *executable statement* (3.30) within a *test item* (3.52)

Note 1 to entry: An exit point is a terminal point of a *path* (3.39) through a test item, being an executable statement within the test item which either terminates the test item or returns control to an external process. This is most commonly the last executable statement within the test item.

3.32
expected result
observable predicted behaviour of the *test item* (3.52) under specified *conditions* (3.13) based on its specification or another source

3.33
experience-based testing
class of *test case* (3.49) design techniques based on using the experience of testers to generate test cases

EXAMPLE Error guessing.

Note 1 to entry: Experience based *testing* (3.57) can include concepts such as test attacks, tours, and error taxonomies which target potential problems such as security, performance, and other quality areas.

3.34
metamorphic relation
description of how changes to the test inputs for a *test case* (3.49) affect the expected outputs based on the required behaviour of a *test item* (3.52)

3.35
metamorphic testing
specification-based *test case* (3.49) design technique based on generating test cases on the basis of existing test cases and *metamorphic relations* (3.34)

3.36
modified condition/decision coverage testing
MCDC testing
structure-based *test case* (3.49) design technique based on demonstrating that a single Boolean *condition* (3.13) within a decision can independently affect the outcome of the decision

3.37
p-use
predicate data use
data use (3.20) associated with the *decision outcome* (3.21) of the *predicate* (3.40) portion of a decision statement

3.38
P-V pair
parameter-value pair
combination of a *test item* (3.52) parameter with a value assigned to that parameter, used as a test coverage item in *combinatorial test design techniques* (3.12)

3.39
path
sequence of *executable statements* (3.30) of a test *item* (3.52)

3.40
predicate
logical expression which evaluates to TRUE or FALSE to direct the execution *path* (3.39) in code

3.41
random testing
specification-based *test case* (3.49) design technique based on generating test cases to exercise randomly selected *test item* (3.52) inputs

3.42
requirements-based testing
specification-based *test case* (3.49) 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.'

3.43
scenario testing
specification-based *test case* (3.49) design technique based on exercising sequences of interactions between the *test item* (3.52) and other systems

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

3.44
state transition testing
specification-based *test case* (3.49) design technique based on exercising transitions in a state model

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

3.45
statement testing
structure-based *test case* (3.49) design technique based on exercising *executable statements* (3.30) in the source code of the *test item* (3.52)

3.47
sub-path
path (3.39) that is part of a larger path

3.46
syntax testing
specification-based *test case* (3.49) design technique based on exercising elements of a formal definition of the *test item* (3.52) inputs

EXAMPLE *Backus-Naur Form* (3.1) is commonly used for defining the syntax of test item inputs.

3.48
test
activity in which a system or component is executed under specified *conditions* (3.13), the results are observed or recorded, and an evaluation is made of some aspect of the system or component

3.49
test case
set of preconditions, inputs and *expected results* (3.32), developed to drive the execution of a *test item* (3.52) to meet *test objectives* (3.55)

Note 1 to entry: A test case is the lowest level of test implementation documentation (i.e. test cases are not made up of test cases) for the *test level* (3.53) or *test type* (3.56) for which it is intended.

Note 2 to entry: Test case preconditions include the required state of the test environment, data (e.g. databases) used by the test item, and the test item itself.

Note 3 to entry: Inputs are the data information and actions, where applicable, used to drive *test execution* (3.51).

3.50
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* (3.57)

Note 1 to entry: ISO/IEC/IEEE 29119 (all parts) does not use the concept of test conditions, but instead uses the concept of a *test model* (3.54) for test design. See Annex H for an explanation.

3.51
test execution
process of running a *test* (3.48) on the *test item* (3.52), producing actual results

3.52
test item
test object
work product to be tested

EXAMPLE Software component, system, requirements document, design specification, user guide.

3.53
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

EXAMPLE The following are common test levels, listed sequentially: unit/component *testing* (3.57), integration testing, system testing, system integration testing, acceptance testing.

Note 1 to entry: It is not always necessary for a *test item* (3.52) to be tested at all test levels, but the sequence of test levels generally stays the same.

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

3.54
test model
representation of the *test item* (3.52), which allows the *testing* (3.57) to be focused on particular characteristics or qualities

EXAMPLE Requirements statements, *equivalence partitions* (3.28), state transition diagram, use case description, *decision table* (3.23), input syntax description, source code, *control flow* (3.14) graph, parameters and values, *classification tree* (3.10), natural language.

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

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.

Note 3 to entry: A test model can include one or more *test conditions* (3.50).

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

3.55
test objective
reason for performing *testing* (3.57)

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

3.56
test type
testing (3.57) that is focused on specific quality characteristics

EXAMPLE Security testing, functional testing, usability testing, and performance testing.

Note 1 to entry: A test type can be performed at a single *test level* (3.53) or across several test levels (e.g. performance testing performed at a unit test level and at a system test level).

3.57
testing
set of activities conducted to facilitate discovery and evaluation of properties of *test items* (3.52)

Note 1 to entry: Testing activities include planning, preparation, execution, reporting, and management activities, insofar as they are directed towards testing.

4 Conformance

4.1 Intended usage

The requirements in this document are contained in Clauses 5 and 6. It is recognized that particular projects or organizations will not need to use all of the techniques defined by this document. Implementation of ISO/IEC/IEEE 29119-2 involves using a risk-based approach to select a subset of techniques suitable for a given project or organization. There are two ways that an organization or individual can claim conformance to the provisions of this document – full conformance and tailored conformance.

The organization shall assert whether it is claiming full or tailored conformance to this document.

4.2 Full conformance

Full conformance is achieved by demonstrating that all of the requirements (i.e. 'shall' statements) of the chosen (non-empty) set of techniques in Clause 5 and the corresponding test coverage measurement approaches in Clause 6 have been satisfied.

EXAMPLE An organization can choose to conform only to one technique, such as boundary value analysis. In this scenario, the organization would only be required to provide evidence that they have met the requirements of that one technique in order to claim conformance to this document.

4.3 Tailored conformance

Tailored conformance is achieved by demonstrating that the chosen subset of requirements from the chosen (non-empty) set of techniques and corresponding test coverage measurement approaches have been satisfied. Where tailoring occurs, justification shall be provided (either directly or by reference) whenever the normative requirements of a technique defined in Clause 5 or measure defined in Clause 6