AS/NZS ISO/IEC/IEEE 29119.4:2022
ISO/IEC/IEEE 29119-4:2021

Australian/New Zealand Standard™

Software and systems engineering — Software testing

Part 4: Test techniques



Documents
Document Sharing Hub

Full document details: AS NZS ISO IEC IEEE 29119.4-2022

AS/NZS ISO/IEC/IEEE 29119.4:2022

This Joint Australian/New Zealand Standard™ was prepared by Joint Technical Committee IT-015, Software and Systems Engineering. It was approved on behalf of the Council of Standards Australia on 12 May 2022 and by the New Zealand Standards Approval Board on 4 May 2022.

This Standard was published on 27 May 2022.

The following are represented on Committee IT-015:

Australian Computer Society
Australian Digital Health Agency
Australian Society for Technical Communication (NSW)
Department of Defence (Australian Government)
Engineers Australia
Griffith University
Institute of IT Professionals New Zealand
IT Service Management Forum Australia
NSW Business Chamber
Systems Engineering Society of Australia
Universities New Zealand
University of New South Wales
University of Southern Queensland
University of Technology Sydney

This Standard was issued in draft form for comment as DR AS/NZS ISO/IEC/IEEE 29119.4:2022.

Keeping Standards up-to-date

Ensure you have the latest versions of our publications and keep up-to-date about Amendments, Rulings, Withdrawals, and new projects by visiting:

www.standards.org.au

www.standards.govt.nz

ISBN 978 1 76113 768 6

Documents
Document Sharing Hub

Full document details: AS NZS ISO IEC IEEE 29119.4-2022

AS/NZS ISO/IEC/IEEE 29119.4:2022
ISO/IEC/IEEE 29119-4:2021

Australian/New Zealand Standard™

Software and systems engineering — Software testing

Part 4: Test techniques

Originated as AS ISO/IEC/IEEE 29119.4:2018.
Second edition 2022.

COPYRIGHT

© ISO/IEC/IEEE 2022 — All rights reserved
© Standards Australia Limited/the Crown in right of New Zealand, administered by the New Zealand Standards Executive 2022

All rights are reserved. No part of this work may be reproduced or copied in any form or by any means, electronic or mechanical, including photocopying, without the written permission of the publisher, unless otherwise permitted under the Copyright Act 1968 (Cth) or the Copyright Act 1994 (New Zealand).

Documents
Document Sharing Hub

Full document details: AS NZS ISO IEC IEEE 29119.4-2022

Preface

This Standard was prepared by the Joint Standards Australia/Standards New Zealand Committee IT-015, Software and Systems Engineering, to supersede AS/NZS ISO/IEC/IEEE 29119.4:2018.

The objective of this document is to define test design techniques that can be used during the test design and implementation process that is described in AS/NZS ISO/IEC/IEEE 29119.2:2022.

Each technique follows the test design and implementation process that is defined in AS/NZS ISO/IEC/IEEE 29119.2:2022. This document is intended for testers, test managers, and developers, particularly those responsible for managing and implementing software testing.

This document is identical with, and has been reproduced from, ISO/IEC IEEE 29119-4:2021, *Software and systems engineering — Software testing — Part 4: Test techniques.*

As this document has been reproduced from an International document, a full point substitutes for a comma when referring to a decimal marker.

Australian or Australian/New Zealand Standards that are identical adoptions of international normative references may be used interchangeably. Refer to the online catalogue for information on specific Standards.

The terms “normative” and “informative” are used in Standards to define the application of the appendices or annexes to which they apply. A “normative” appendix or annex is an integral part of a Standard, whereas an “informative” appendix or annex is only for information and guidance.

Documents
Document Sharing Hub

© Standards Australia Limited/Standards New Zealand 2022

Full document details: AS NZS ISO IEC IEEE 29119.4-2022

Contents

Preface ii
Foreword v
Introduction vii
1 Scope 1
2 Normative references 1
3 Terms and definitions 1
4 Conformance 7
4.1 Intended usage 7
4.2 Full conformance 7
4.3 Tailored conformance 7
5 Test design techniques 8
5.1 Overview 8
5.2 Specification-based test design techniques 10
5.2.1 Equivalence partitioning 10
5.2.2 Classification tree method 12
5.2.3 Boundary value analysis 12
5.2.4 Syntax testing 14
5.2.5 Combinatorial test design techniques 15
5.2.6 Decision table testing 18
5.2.7 Cause-effect graphing 18
5.2.8 State transition testing 19
5.2.9 Scenario testing 20
5.2.10 Random testing 21
5.2.11 Metamorphic testing 21
5.2.12 Requirements-based testing 22
5.3 Structure-based test design techniques 23
5.3.1 Statement testing 23
5.3.2 Branch testing 23
5.3.3 Decision testing 24
5.3.4 Branch condition testing 25
5.3.5 Branch condition combination testing 25
5.3.6 Modified condition/decision coverage (MCDC) testing 26
5.3.7 Data flow testing 27
5.4 Experience-based test design techniques 29
5.4.1 Error guessing 29
6 Test coverage measurement 30
6.1 Overview 30
6.2 Test measurement for specification-based test design techniques 30
6.2.1 Equivalence partition coverage 30
6.2.2 Classification tree method coverage 30
6.2.3 Boundary value analysis coverage 31
6.2.4 Syntax testing coverage 31
6.2.5 Combinatorial test design techniques coverage 31
6.2.6 Decision table testing coverage 32
6.2.7 Cause-effect graphing coverage 32
6.2.8 State transition testing coverage 32
6.2.9 Scenario testing coverage 33
6.2.10 Random testing coverage 33
6.2.11 Metamorphic testing coverage 33
6.2.12 Requirements-based testing coverage 33
6.3 Test measurement for structure-based test design techniques 33
6.3.1 Statement testing coverage 33

Documents
Document Sharing Hub
Full document details: AS NZS ISO IEC IEEE 29119.4-2022

6.3.2 Branch testing coverage 33
6.3.3 Decision testing coverage 34
6.3.4 Branch condition testing coverage 34
6.3.5 Branch condition combination testing coverage 34
6.3.6 Modified condition/decision coverage (MCDC) 34
6.3.7 Data flow testing coverage 35
6.4 Test measurement for experience-based testing design techniques — Error guessing coverage 35

Annex A (informative) **Testing quality characteristics** **36**

Annex B (informative) **Guidelines and examples for the application of specification-based test design techniques** **49**

Annex C (informative) **Guidelines and examples for the application of structure-based test design techniques** **102**

Annex D (informative) **Guidelines and examples for the application of experience-based test design techniques** **122**

Annex E (informative) **Guidelines and examples for the application of grey-box test design techniques** **125**

Annex F (informative) **Test design technique effectiveness** **128**

Annex G (informative) **ISO/IEC/IEEE 29119-4 and BS 7925-2 test design technique alignment** **131**

Annex H (informative) **Test models** **133**

Bibliography **134**

IEEE Notices and Abstract **136**

Documents
Document Sharing Hub

Full document details: AS NZS ISO IEC IEEE 29119.4-2022

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.c).

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.

Documents
Document Sharing Hub
© Standards Australia Limited/Standards New Zealand 2022
Full document details: AS NZS ISO IEC IEEE 29119.4-2022

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.

© Standards Australia Limited/Standards New Zealand 2022

Documents
Document Sharing Hub

Full document details: AS NZS ISO IEC IEEE 29119.4-2022

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.

Documents Document Sharing Hub
© Standards Australia Limited/Standards New Zealand 2022
Full document details: AS NZS ISO IEC IEEE 29119.4-2022

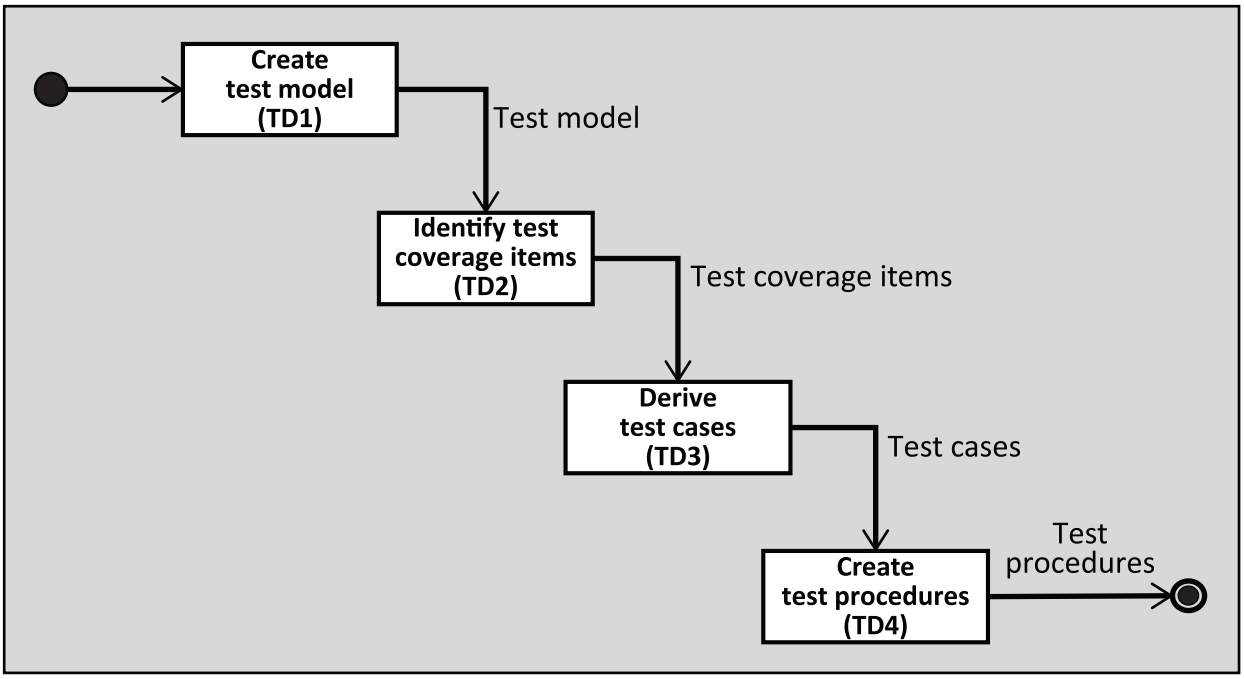


Figure 1 — ISO/IEC/IEEE 29119-2 test design and implementation process

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.

© Standards Australia Limited/Standards New Zealand 2022

Documents
Document Sharing Hub
Full document details: AS NZS ISO IEC IEEE 29119.4-2022

Australian/New Zealand Standard

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

© Standards Australia Limited/Standards New Zealand 2022

Documents Document Sharing Hub

Full document details: AS NZS ISO IEC IEEE 29119.4-2022

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

© Standards Australia Limited/Standards New Zealand 2022

Documents
Document Sharing Hub
Full document details: AS NZS ISO IEC IEEE 29119.4-2022

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.

Documents Document Sharing Hub
© Standards Australia Limited/Standards New Zealand 2022
Full document details: AS NZS ISO IEC IEEE 29119.4-2022

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

© Standards Australia Limited/Standards New Zealand 2022

Documents
Document Sharing Hub

Full document details: AS NZS ISO IEC IEEE 29119.4-2022

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

© Standards Australia Limited/Standards New Zealand 2022

Full document details: AS NZS ISO IEC IEEE 29119.4-2022

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).

© Standards Australia Limited/Standards New Zealand 2022

Documents
Document Sharing Hub

Full document details: AS NZS ISO IEC IEEE 29119.4-2022

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 are not followed completely. All tailoring decisions shall be recorded with their rationale, including the consideration of any applicable risks. Tailoring shall be agreed by the relevant stakeholders.

Documents Document Sharing Hub
© Standards Australia Limited/Standards New Zealand 2022
Full document details: AS NZS ISO IEC IEEE 29119.4-2022

5 Test design techniques

5.1 Overview

This document defines test design techniques for specification-based testing (5.2), structure-based testing (5.3) and experience-based testing (5.4). In specification-based testing, the test basis (e.g. requirements, specifications, models or user needs) is used as the main source of information to design test cases. In structure-based testing, the structure of the test item (e.g. source code or the structure of a model) is used as the primary source of information to design test cases. In experience-based testing, the knowledge and experience of the tester is used as the primary source of information during test case design. For specification-based testing, structure-based testing and experience-based testing, the test basis is used to generate the expected results. These classes of test design techniques are complementary, and their combined application typically results in more effective testing.

Although the techniques presented in this document are classified as specification-based, structure-based or experience-based, in practice some of them can be used interchangeably (e.g. branch testing can be applied to the specification of the graphical user interface of a web-based system). This is demonstrated in Annex E. In addition, although each technique is defined independently of all others, in practice some can be used in combination with other techniques.

EXAMPLE The test coverage items derived by applying equivalence partitioning can be used to identify the input parameters of test cases derived for scenario testing.

This document uses the terms specification-based testing and structure-based testing; these categories of techniques are also known as "black-box testing" and "white-box testing" (or "clear-box testing") respectively. The terms “black-box” and “white-box” refer to the visibility of the internal structure of the test item. In black-box testing the internal structure of the test item is not visible (hence the black box), whereas for white-box testing the internal structure of the test item is visible. When a technique is applied while utilising a combination of knowledge from the test item’s specification and structure, this is often called "grey-box testing".

This document defines how the generic test design and implementation process steps TD1(create test model), TD2 (identify test coverage items), and TD3 (derive test cases) from ISO/IEC/IEEE 29119-2 (see Introduction) shall be used by each technique. It does not provide context-specific definitions of the techniques that describe how each technique should be used in all situations. Annex B, Annex C, Annex D and Annex E provide detailed examples that demonstrate how to apply the techniques.

The techniques that are defined this document are shown in Figure 2. This set of techniques is not exhaustive. There are useful techniques that are used by testing practitioners or researchers that are not included in this document.

© Standards Australia Limited/Standards New Zealand 2022

Documents
Document Sharing Hub

Full document details: AS NZS ISO IEC IEEE 29119.4-2022

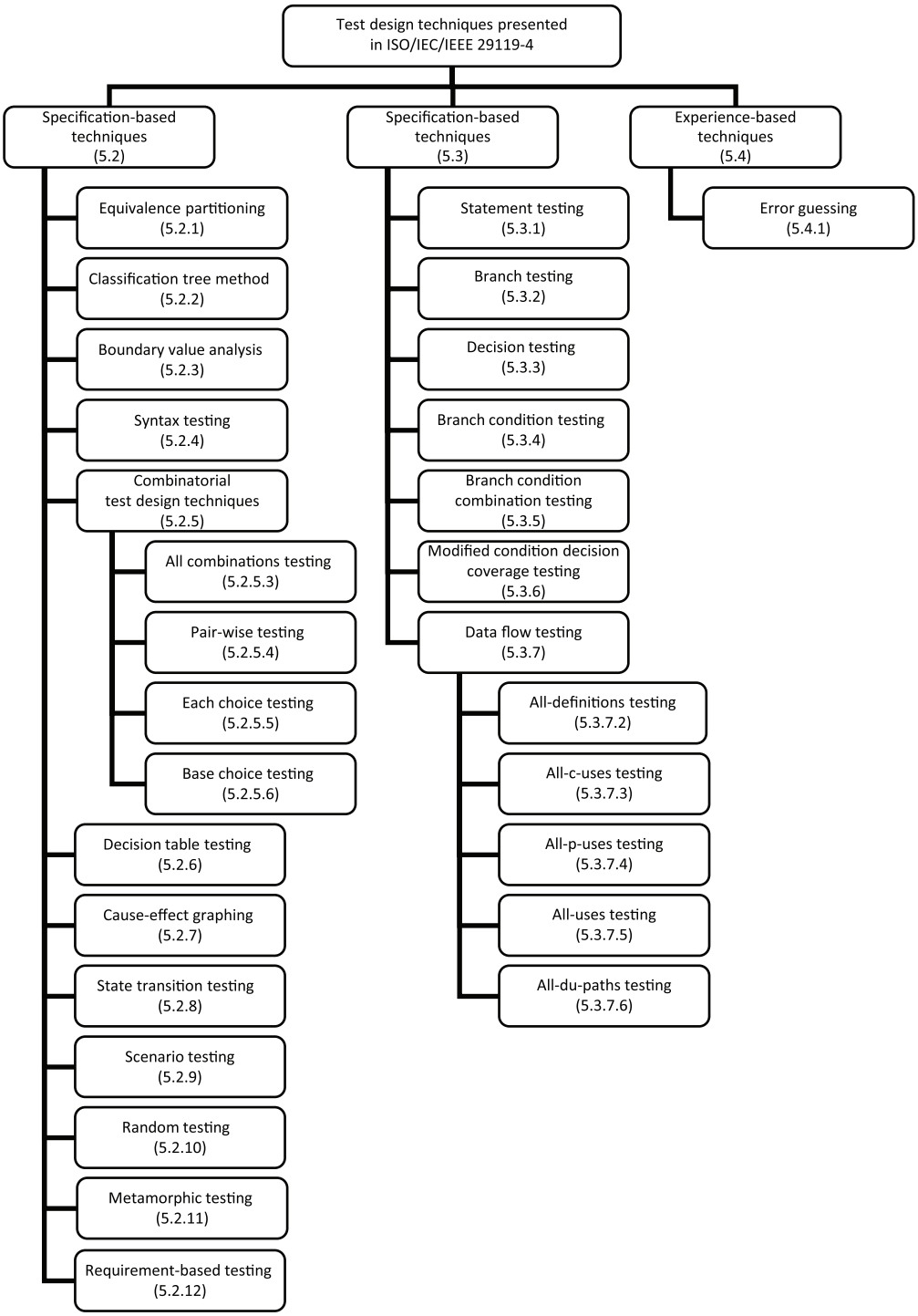


Figure 2 — The set of test design techniques presented in this document

Of the four activities in the test design and implementation process (see Figure 1), test design techniques provide unique and specific guidance on the derivation of test models (TD1), identification of test coverage items (TD2) and the derivation of test cases (TD3). Therefore, each technique is defined in terms of these three activities.

There are varying levels of granularity within steps TD1 (create test model), TD2 (identify test coverage items) and TD3 (derive test cases). Within each technique, the term "model" is used to describe the concept of preparing a logical representation of the test item for the purposes of deriving test coverage items in step TD2 (e.g. a control flow model is normally used as a test model for identifying test coverage items for all structural techniques).

Documents Document Sharing Hub
© Standards Australia Limited/Standards New Zealand 2022
Full document details: AS NZS ISO IEC IEEE 29119.4-2022

EXAMPLE 1 In state transition testing, if there is a requirement to cover all states then the state model for the test item would form the test model. If there is a requirement to cover all transitions between states, then each transition would be a test coverage item.

In addition, since some techniques share underlying concepts, their definitions contain similar text.

EXAMPLE 2 Both equivalence partitioning and boundary value analysis are based on equivalence classes.

In the test case design activity (TD3) of each technique, test cases that are created may be "valid" (i.e. they contain input values that the test item should accept as correct) or "invalid" (i.e. they contain at least one input value that the test item should reject as incorrect, ideally with an appropriate error message). In some techniques, such as equivalence partitioning and boundary value analysis, invalid test cases are usually derived using the "one-to-one" approach as it avoids fault masking by ensuring that each test case only includes one invalid input value, while valid test cases are typically derived using the "minimized" approach, as this reduces the number of test cases required to cover valid test coverage items (see 5.2.1.4 and 5.2.3.3).

NOTE "Valid" test cases are also known as "positive" test cases. Invalid cases are also known as "negative" test cases.

Although the techniques defined in this document are each described in a separate clause (as if they were mutually exclusive), in practice they can be applied in a blended way.

EXAMPLE 3 Boundary value analysis can be used to select test input values, after which pair-wise testing can be used to design complete test cases from the test input values. Similarly, equivalence partitioning can be used to select the classifications and classes for the classification tree method and then each choice testing can be used to construct test cases from the classes.

The techniques presented in this document can also be used in conjunction with the test types that are presented in Annex A. For example, equivalence partitioning can be used to identify user groups and then representative users (test coverage items) from those groups to create test cases for usability testing.

The definitions of the techniques are provided in Clause 5. The corresponding coverage measures for each technique are presented in Clause 6. These are supported by examples of each technique in Annexes B, C, D and E. Although the examples of each technique demonstrate manual application of the technique, in practice, automation can be used to support some types of test design and execution.

EXAMPLE 4 Statement coverage analysers can be used to support structure-based testing.

Annex A provides examples of how the test design techniques defined in this document can be applied to testing the non-functional quality characteristics defined in ISO/IEC 25010.

5.2 Specification-based test design techniques

5.2.1 Equivalence partitioning

5.2.1.1 General

Equivalence partitioning (BS 7925-2; Myers 1979[19]) is based on exercising equivalence partitions. These equivalence partitions are derived based on the expectation that values within a partition should be treated similarly by the test item.

5.2.1.2 Create test model (TD1)

A test model shall be created that identifies the equivalence partitions for the test item. A suitable test model shows the various functions expected to be performed by the test item from which equivalence partitions can be derived. These functions can be considered as a set of processes, which can be decomposed into input processes, internal processes and output processes.

© Standards Australia Limited/Standards New Zealand 2022

Documents
Document Sharing Hub

Full document details: AS NZS ISO IEC IEEE 29119.4-2022

The functions described in the test item's specification can typically be used to identify its internal processes. Consideration of the inputs and outputs of the test item can be used to help determine its probable input and output processes.

Equivalence partitions should be identified for both valid and invalid functions (leading to both valid and invalid outputs) and valid and invalid inputs to the test item. In this context, invalid functions are those that are not included in the specification and which cannot reasonably be expected to be included in a correct implementation of the test item. Similarly, invalid inputs are those that should either be ignored by the test item or cause the test item to output an error message.

Equivalence partitions for a test item can overlap, and in some situations, one partition can be a subset of another partition. By adding detail to the test model, further equivalence partitions can often be derived, often as subsets of existing partitions. As more detail and partitions are added, the number of test cases required to fully exercise the test model is likely to increase.

EXAMPLE For a test item expecting lowercase alphabetical characters as (valid) inputs, invalid input partitions can be derived containing integers, reals, uppercase alphabetical characters, symbols and control characters, depending on the level of rigour required during testing. If each of these partitions were included separately in the model, each would be expected to be exercised to achieve full coverage, whereas if a single partition was created to cover all invalid inputs, then only a single partition would need to be exercised to achieve full partition coverage.

NOTE 1 Invalid equivalence partitions that correspond to inputs, functions or outputs that have not been specified (as is often the case) are typically based on the experience and imagination of the tester, which means different testers will often derive different invalid partitions. This subjective form of test design can also occur when applying experience-based techniques like error guessing.

NOTE 2 Domain analysis (Beizer 1995[2]) is often classified as a combination of equivalence partitioning and boundary value analysis.

5.2.1.3 Identify test coverage items (TD2)

Each equivalence partition shall be identified as a test coverage item.

5.2.1.4 Derive test cases (TD3)

Test cases shall be derived to exercise the test coverage items (i.e. the equivalence partitions). The following steps shall be used during test case derivation:

a) Decide on an approach for selecting combinations of test coverage items to be exercised by test cases. Two common approaches are (BS 7925-2; Myers 1979[19]):

   1) one-to-one, in which each test case is derived to cover a specific equivalence partition;

   2) minimized, in which the minimum number of test cases is derived to cover each equivalence partition at least once.

      NOTE Other approaches to selecting combinations of test coverage items to be exercised by test cases are described in 5.2.5.

b) Select test coverage items for inclusion in the current test case based on the approach chosen in step a).

c) Identify input values to exercise the test coverage items to be covered by the test case and arbitrary valid values for any other input variables required by the test case.

d) Determine the expected result of the test case by applying the inputs to the test basis.

e) Repeat steps b) to d) until the required level of test coverage is achieved.

Documents Document Sharing Hub
© Standards Australia Limited/Standards New Zealand 2022
Full document details: AS NZS ISO IEC IEEE 29119.4-2022

5.2.2 Classification tree method

5.2.2.1 Create test model (TD1)

The classification tree method shall use a test model that partitions the input domain of the test item in terms of "classifications" and represents them graphically in the form of a classification tree. See Grochtmann and Grimm 1993[11] for more details. Each classification consists of a complete and disjoint (non-overlapping) set of "classes" (values the classifications can take). These classes may be further classified (even recursively) into sub-classifications and sub-classes, depending on the level of rigour required in the testing.

NOTE The process of partitioning in the classification tree method is similar to equivalence partitioning. The key difference is that in the classification tree method, the partitions (which are classifications and classes) are completely disjoint, whereas in equivalence partitioning, they can overlap depending on how the technique is applied.

5.2.2.2 Identify test coverage items (TD2)

There are two approaches for identifying test coverage items for classification trees:

— When the required test coverage is based on "minimality", then the test coverage items shall be identified as the classes represented by leaf nodes in the classification tree.

— When the required test coverage is based on "maximality", then the test coverage items shall be identified as unique combinations of the classes represented by leaf nodes in the classification tree.

5.2.2.3 Derive test cases (TD3)

Test cases shall be derived to exercise the test coverage items. When deriving test cases, combinations of classes are included in each test case.

NOTE The combinations of classes are often illustrated in a combination table (see Figure B.1).

The following steps shall be followed during test case derivation:

a) Based on the selected coverage criterion, select a combination of classes for inclusion in the current test case that has not already been covered by a test case.

b) Identify input values for any classes that do not already have an assigned value in the current test case.

c) Determine the expected result of the test case by applying the inputs to the test basis.

d) Repeat steps a) to c) until the required level of test coverage is achieved.

5.2.3 Boundary value analysis

5.2.3.1 Create test model (TD1)

Boundary value analysis shall use a model of the test item that partitions the inputs and outputs of the test item into a number of ordered sets and subsets (i.e. partitions and sub-partitions) with identifiable boundaries, where each boundary is then used to identify test coverage items. See BS 7925-2 and Myers 1979[19] for more details. These partitions correspond to those equivalence partitions defining ordered sets that would form the test model for equivalence partitioning (see 5.2.1.2).

EXAMPLE For a partition defined as integers from 1 to 10 inclusive, there are two boundaries, where the lower boundary is 1 and the upper boundary is 10, and these two input values would be covered by testing.

NOTE For output boundaries, corresponding input partitions are derived based on the processing described in the test item's specification. Test inputs are then selected from the input partitions that cause the output partitions to occur during testing.

© Standards Australia Limited/Standards New Zealand 2022

AVA Documents Document Sharing Hub

Full document details: AS NZS ISO IEC IEEE 29119.4-2022

5.2.3.2 Identify test coverage items (TD2)

One of the following two options for the identification of test coverage items shall be used:

— two-value boundary testing; or

— three-value boundary testing.

For two-value boundary testing, two test coverage items shall be identified for each boundary, corresponding to values on the boundary and an incremental distance outside the boundary of the equivalence partition. This incremental distance shall be defined as the smallest significant value for the data type under consideration.

For three-value boundary testing, three test coverage items shall be identified for each boundary, corresponding to values on the boundary and an incremental distance each side of the boundary of the equivalence partition. This incremental distance shall be defined as the smallest significant value for the data type under consideration.

NOTE 1 Some partitions can have only a single boundary identified in the test basis. For example, the numerical partition "age ≥ 70 years" has a specified lower boundary but not an obvious upper boundary. In some cases, a boundary value imposed by the actual implementation can be used as a boundary value, such as the largest value that is accepted by the input field (such decisions are ideally documented; for example, in the test specification documentation).

NOTE 2 Two-value boundary value testing is typically adequate in most situations; however, three-value boundary testing can be required for certain circumstances (e.g. for rigorous testing to check that no errors were made in determining the boundaries of variables in the test item by both testers and developers).

NOTE 3 In both two- and three-value boundary value testing, contiguous partitions (partitions that share a boundary) will result in duplicate test coverage items, in which case it is typical practice to only exercise these duplicated values once. For an example of duplicate boundaries, see B.2.3.5.

5.2.3.3 Derive test cases (TD3)

Test cases shall be derived to exercise the test coverage items. The following steps shall be used during test case derivation:

a) Decide on an approach for selecting combinations of test coverage items to be exercised by test cases. Two common approaches are (BS 7925-2; Myers 1979[19]):

   1) one-to-one, in which each test case is derived to exercise a specific test coverage item;

   2) minimized, in which test coverage items are included in test cases such that the minimum number of test cases are derived to cover all test coverage items at least once.

   NOTE Other approaches to selecting combinations of test coverage items to be exercised by test cases are described in 5.2.5.

b) Select test coverage items for inclusion in the current test case based on the approach chosen in step a).

c) Identify arbitrary valid values for any other input variables required by the test case that were not already selected in step b).

d) Determine the expected result of the test case by applying the input(s) to the test basis.

e) Repeat steps b) to d) until the required level of test coverage is achieved.

Documents Document Sharing Hub
© Standards Australia Limited/Standards New Zealand 2022
Full document details: AS NZS ISO IEC IEEE 29119.4-2022

5.2.4 Syntax testing

5.2.4.1 Create test model (TD1)

Syntax testing shall use a formal model of the inputs to a test item as the basis for test design. See Beizer 1995[2] and Burnstein 2003[5] for more details. This syntax model is represented as a number of rules, where each rule defines the format of an input parameter in terms of "sequences of", "iterations of", or "selections between" elements in the syntax. The syntax may be represented in a textual or diagrammatic format.

EXAMPLE 1 Backus-Naur Form is a formal meta-language that can be used to define the syntax for the inputs to a test item in a textual format.

EXAMPLE 2 An abstract syntax tree can be used to represent formal syntax diagrammatically.

5.2.4.2 Identify test coverage items (TD2)

In syntax testing, test coverage items shall be identified based on two goals: positive testing, in which test coverage items are identified to cover the valid syntax in various ways, and negative testing, in which test coverage items are identified to deliberately violate the rules of the syntax. Test coverage items for positive testing are known as "options" of the defined syntax, whereas test coverage items for negative testing are known as "mutations" of the defined syntax.

The following guidelines may be used to identify "options" (although alternative guidelines may be used where appropriate):

a) Whenever selection is mandated by the syntax, an "option" is identified for each alternative provided for that selection.

EXAMPLE 1 For the input parameter "colour = Blue | Red | Green" (where | represents the "OR" Boolean operator), three options "Blue", "Red" and "Green" will be derived as test coverage items.

b) Whenever an iteration is mandated by the syntax, at least two "options" are identified for the iteration; one with the minimum number of repetitions and the other with more than the minimum number of repetitions.

EXAMPLE 2 For the input parameter "letter = [A – Z | a – z]+" (where "+" represents "one or more"), two options, "one letter" and "more than one letter" will be identified as test coverage items.

c) Whenever iteration is mandated with a maximum number of repetitions, at least two "options" are identified for the iteration; one with the maximum number of repetitions and the other with less than the maximum number of repetitions.

EXAMPLE 3 For the input parameter "letter = [A – Z | a – z]^{100}" (where "^{100}" represents the fact that a letter can be chosen up to 100 times), two options "100 letters" and "less than 100 letters" will be identified as test coverage items.

To identify "mutations", the defined syntax may be mutated in various ways, often dependent on the experience and creativity of the tester.

EXAMPLE 4 For the input parameter "colour = Blue | Red | Green", one "mutation" can be to introduce an invalid value for the parameter as the test coverage item, such as selecting the value "Yellow", which does not appear in the input parameter list. Other example mutations are provided in B.2.4.5.

© Standards Australia Limited/Standards New Zealand 2022

Documents Document Sharing Hub

Full document details: AS NZS ISO IEC IEEE 29119.4-2022

5.2.4.3 Derive test cases (TD3)

Test cases for syntax testing shall be derived to cover the identified options and mutations. The following steps shall be used during test case derivation:

a) Decide on an approach for selecting combinations of test coverage items to be exercised by test cases, where two common approaches are:

   1) one-to-one, in which each test case is derived to exercise a specific option or mutation; or

   2) minimized, in which options and mutations are included in test cases such that the minimum number of test cases are derived to cover all options and mutations at least once.

   NOTE 1 Other approaches to selecting combinations of test coverage items to be exercised by test cases are described in 5.2.5.

   NOTE 2 Using a minimized approach for mutations can lead to such unrealistic test inputs that it can be unlikely that the test item will fail to identify the test input as invalid.

b) Select test coverage items for inclusion in the current test case.

c) Identify input values to exercise the test coverage items to be covered by the test case and arbitrary valid values for any other input variables required by the test case.

d) Determine the expected result of the test case by applying the inputs to the test basis.

e) Repeat steps b) to d) until all identified options and mutations are exercised.

5.2.5 Combinatorial test design techniques

5.2.5.1 Overview

Combinatorial test design techniques (Grindal, Offutt and Andler 2005[10]) are used to systematically derive a meaningful and manageable subset of test cases that cover the test model and the associated test coverage items that are derivable during testing. The combinations of interest are defined in terms of test item parameters and the values these parameters can take. Where numerous parameters (each with numerous discrete values) must interact, this technique enables a significant reduction in the number of test cases required without compromising functional coverage.

5.2.5.2 Create test model (TD1)

All combinatorial test design techniques shall use a test model comprising a set of P-V pairs that combines each of the test item parameters (P) taking on each of their possible values (V).

The test item parameters represent particular aspects of the test item that are relevant to the testing, and often correspond to the input parameters to the test item, but other aspects may also be used.

EXAMPLE 1 In configuration testing, the parameters can be various environment factors, such as operating system and browser.

Each test item parameter can take on various values. For use in this technique the set of values needs to be finite and manageable. Some test item parameters may be naturally constrained to only take on a small set of possible values while other test item parameters may be far less constrained.

EXAMPLE 2 A test item parameter that is naturally constrained to a small number of values is a “day_of_week” parameter that can only take on seven specific values “[Monday | Tuesday | Wednesday | Thursday | Friday | Saturday | Sunday]”.

EXAMPLE 3 A test item parameter that is far less constrained is a parameter consisting of any Real number, which exists along a potentially infinite number line.

Documents Document Sharing Hub
© Standards Australia Limited/Standards New Zealand 2022
Full document details: AS NZS ISO IEC IEEE 29119.4-2022

For unconstrained test item parameters, it may first be necessary to apply other test design techniques, such as equivalence partitioning or boundary value analysis, to reduce the large set of possible values for a parameter to a manageable subset.

5.2.5.3 All combinations testing

5.2.5.3.1 Identify test coverage items (TD2)

In 'all combinations' testing, the test coverage items shall be members of the set of all unique combinations of P-V pairs, such that each parameter is included at least once in the set. See Grindal, Offutt and Andler 2005[10] for more details.

5.2.5.3.2 Derive test cases (TD3)

Test cases shall be derived in which each test case exercises one unique combination of P-V pairs. The following steps shall be used during test case derivation:

a) Select test coverage items for inclusion in the current test case that have not already been covered by a test case.

b) Determine the expected result of the test case by applying the input(s) to the test basis.

c) Repeat steps a) and b) until the required level of test coverage is achieved.

NOTE The minimum number of test cases required to achieve 100 % all combinations testing coverage corresponds to the product of the number of P-V pairs for each test item parameter.

EXAMPLE A test item with two parameters for digits (10 possible values) and lower-case alphabetic characters (26 possible values) would result in 260 test cases.

5.2.5.4 Pair-wise testing

5.2.5.4.1 Identify test coverage items (TD2)

In pair-wise testing, the test coverage items shall be unique pairs of P-V pairs, where each P-V pair within the pair is for a different test item parameter. See Grindal, Offutt and Andler 2005[10] for more details.

EXAMPLE A test item with two parameters for months (12 values) and gender (two values) results in 24 possible pairs of P-V pairs (e.g. January-female, January-male, February-female, February-male, March-female).

Instead of all possible combinations of the parameters (as was required for all combinations testing), this technique covers all possible pairs of the selected values within the total set, thereby efficiently exercising the test item with fewer test cases. Pair-wise testing is also known as "all pairs" testing.

5.2.5.4.2 Derive test cases (TD3)

Having identified the P-V pairs, test cases shall be derived steps to exercise pairs of P-V pairs, where each test case exercises one or more unique pairs. The following shall be used during test case derivation:

a) Select test coverage items for inclusion in the current test case, where pairs of P-V pairs cover one or more pairs of parameter values that have not yet been included in a test case.

b) Identify arbitrary valid values for any other parameters present in the test case.

c) Determine the expected result of the test case by applying the inputs to the test basis.

d) Repeat steps a) to c) until all unique pairs of P-V pairs have been exercised.

© Standards Australia Limited/Standards New Zealand 2022

Documents
Document Sharing Hub

Full document details: AS NZS ISO IEC IEEE 29119.4-2022

The minimum number of test cases required to achieve 100 % pair-wise testing is not easily calculated. A near-optimal set may be considered acceptable and may be calculated using one of the following three options:

— manually determine a near-optimal set using an algorithm;

— use an automated tool (implementing an algorithm) to determine a near-optimal set; or

— use orthogonal arrays (Mandl 1985[18]) to determine a near-optimal set.

5.2.5.5 Each choice testing

5.2.5.5.1 Identify test coverage items (TD2)

In each choice (or 1-wise) testing, the test coverage items shall be members of the set of P-V pairs such that each parameter value is included at least once in the set. See Grindal, Offutt and Andler 2005[10] for more details.

5.2.5.5.2 Derive test cases (TD3)

Test cases shall be derived to exercise P-V pairs, where each test case exercises one or more P-V pairs that have not been previously included in a test case. The following steps shall be used during test case derivation:

a) Select test coverage items for inclusion in the current test case, where at least one selected test coverage item has not been included in a prior test case.

b) Identify arbitrary valid values for any other parameters present in the test case.

c) Determine the expected result of the test case by applying the inputs to the test basis.

d) Repeat steps a) to c) until the required level of test coverage is achieved.

The minimum number of test cases required to achieve 100 % each choice testing corresponds to the maximum number of values any one of the test item parameters can take.

EXAMPLE A test item with three parameters for digits (10 possible values), months (12 possible values) and lower-case alphabetic characters (26 possible values) would result in 26 test cases.

5.2.5.6 Base choice testing

5.2.5.6.1 Identify test coverage items (TD2)

In base choice testing, the test coverage items shall be sets of P-V pairs for each of the input parameters, where all parameters except one are set to their “base” value and the final parameter is set to one of its other valid values. See Grindal, Offutt and Andler 2005[10] for more details.

NOTE There are a number of approaches for choosing the base values for each parameter, but generally the base choice is the most ‘important’ value that the parameter can take. For example, the base values can be chosen from the operational profile, from the typical path in scenario testing, or from the most frequently used values for the parameter.

5.2.5.6.2 Derive test cases (TD3)

Having identified the P-V pairs, base choices for each of the parameters shall be chosen. Test cases shall be derived by setting all but one parameter to its base choice and then setting the remaining parameter to each of its valid values until the required level of test coverage is achieved. The following steps shall be used during test case derivation:

a) Derive the base-choice test case by setting each parameter to its “base” value.

Documents Document Sharing Hub
© Standards Australia Limited/Standards New Zealand 2022
Full document details: AS NZS ISO IEC IEEE 29119.4-2022

b) Create a new test case by setting one parameter to a valid (non-base choice) value while keeping the remaining set of parameters to their base choice values.

c) Determine the expected result of the new test case by applying the inputs to the test basis.

d) Repeat steps b) and c) until the required level of test coverage is achieved.

5.2.6 Decision table testing

5.2.6.1 Create test model (TD1)

Decision table testing shall use a model of the logical relationships (decision rules) between conditions (causes) and actions (effects) for the test item in the form of a decision table, where:

— each cause is expressed as a condition, which is either true or false (i.e. a Boolean) for an input, or combination of inputs;

— each effect is expressed as a Boolean expression represented by an output (i.e. action), or a combination of outputs;

— a set of decision rules define the required relationships between causes and effects.

See BS 7925-2 and Myers 1979[19] for more details.

NOTE If conditions consist of multiple values rather than simple Booleans, this results in an "extended entry" decision table, the testing of which can be handled by equivalence partitioning.

5.2.6.2 Identify test coverage items (TD2)

In decision table testing, the test coverage items shall be the decision rules, which define the relationship between a unique combination of the test item's conditions and outputs.

5.2.6.3 Derive test cases (TD3)

Test cases shall be derived to exercise each of the decision rules (test coverage items). The following steps shall be used during test case derivation:

a) Select a decision rule from the decision table for implementation as a test case.

b) Identify input values to satisfy the input conditions of the decision rule to be covered by the test case.

c) Determine the expected result of the test case by applying the inputs to the decision table.

d) Repeat steps a) to c) until the required level of test coverage is achieved.

NOTE If the decision table contains dependent input conditions then this can result in infeasible combinations (e.g. two separate conditions, "age less than 18" and "age greater than 65" both set to true). In this situation such infeasible decision rules need to be identified and ideally documented, and do not need to be covered during testing.

5.2.7 Cause-effect graphing

5.2.7.1 Create test model (TD1)

Cause-effect graphing shall use a model of the logical relationships (decision rules) between causes (e.g. inputs) and effects (e.g. outputs) for the test item in the form of a cause-effect graph, where:

— each cause is expressed as a condition, which is either true or false (i.e. a Boolean) for an input, or combination of inputs;

— each effect is expressed as a Boolean expression represented by an output, or a combination of outputs;

© Standards Australia Limited/Standards New Zealand 2022

AVA Documents Document Sharing Hub

Full document details: AS NZS ISO IEC IEEE 29119.4-2022

— a set of decision rules define the required relationships between causes and effects.

See BS 7925-2, Myers 1979[19], and Nursimulu and Probert 1995[20] for more details.

The cause-effect graph models logical relationships between causes and effects as a Boolean logic network with Boolean operators, and can optionally model syntactic and semantic constraints over relationships between causes and relationships between effects (see Figures B.5 and B.6).

5.2.7.2 Identify test coverage items (TD2)

In cause-effect graphing, the test coverage items shall be the decision rules, which define the relationship between a unique combination of the test item's causes and effects.

5.2.7.3 Derive test cases (TD3)

Test cases shall be derived to exercise the test coverage items. A corresponding decision table may be produced from the cause-effect graph and used to derive the test cases. The following steps shall be used during test case derivation:

a) Select a decision rule to be implemented in the current test case.

b) Identify input values to exercise the decision rule to be covered by the test case.

c) Determine the expected result of the test case by applying the inputs to the cause-effect graph (or corresponding decision table).

d) Repeat steps a) to c) until the required level of test coverage is achieved.

5.2.8 State transition testing

5.2.8.1 Create test model (TD1)

State transition testing shall use a model of the states the test item can occupy, the transitions between states, the events which cause transitions and the actions that can result from the transitions. See BS 7925-2 and Chow 1978[7] for more details. The states of the model shall be discrete, identifiable and finite in number. An individual transition may be constrained by an event guard, which defines a set of conditions that must be true when the event occurs, in order for the transition to occur. In state transition testing, the test coverage typically applies to states or transitions of the state model, depending on the required coverage. The model may take the form of a state transition diagram, a state table or another suitable representation.

5.2.8.2 Identify test coverage items (TD2)

In state transition testing, test coverage items will change depending on the chosen test completion criterion. Possible test completion criteria include but are not limited to the following:

— states, in which test coverage items shall be derived to enable all states in the state model to be "visited";

— single transitions (0-switch coverage), in which test coverage items shall be derived to cover valid single transitions in the state model;

— all transitions, in which test coverage items shall be derived to cover both valid transitions in the state model and "invalid" transitions (transitions from states initiated by events in the state model for which no valid transition is specified);

— multiple transitions (N-switch coverage), in which test coverage items shall be derived to cover valid sequences of N + 1 transitions in the state model.

Documents Document Sharing Hub
© Standards Australia Limited/Standards New Zealand 2022
Full document details: AS NZS ISO IEC IEEE 29119.4-2022

NOTE "1-switch" coverage is a popular variant of "N-switch" coverage that requires pairs of transitions to be exercised.

5.2.8.3 Derive test cases (TD3)

Test cases for state transition testing shall be derived to exercise the test coverage items. The following steps shall be used during test case derivation:

a) Select test coverage items for inclusion in the current test case.

b) Identify input values to exercise the test coverage items to be covered by the test case.

c) Determine the expected result of the test case by applying the inputs to the test basis (the expected result may be defined in terms of outputs and the states visited as described in the state model).

d) Repeat steps a) to c) until the required level of test coverage is achieved.

5.2.9 Scenario testing

5.2.9.1 Create test model (TD1)

Scenario testing shall use a model of the sequences of possible interactions between the test item and other systems (in this context users are often considered to be other systems) for the purpose of testing usage flows involving the test item. See Desikan and Ramesh 2007[9] for more details. The test model typically includes a set of scenarios related to the test item, where each scenario covers one sequence of interactions.

In scenario testing, the test model shall identify:

— the "main" scenario, which is the typical sequence of actions that are expected of the test item or an arbitrary choice when no typical sequence of actions is known; and

— "alternative" scenarios that represent alternative (non-main) scenarios that may be taken through the test item.

NOTE 1 Alternative scenarios can include abnormal use, extreme or stress conditions, exceptions and error handling.

NOTE 2 Scenario testing is typically used for conducting "end-to-end functional testing", such as during system testing or acceptance testing.

NOTE 3 In many situations, it is good practice to have identified scenarios reviewed and agreed by relevant stakeholders, such as business users.

One common form of scenario testing is use case testing (Hass 2008[12]), which utilises a use case model of the test item that describes how the test item interacts with one or more actors for the purpose of testing sequences of interactions (i.e. scenarios) between the test item and actors.

NOTE 4 In use case testing, the use case model describes how various actions are performed by the test item as a result of various triggers from the actors. An actor can be a user or another system.

NOTE 5 Transaction flow testing (Beizer 1995[2]) is often classified as a type of scenario testing.

5.2.9.2 Identify test coverage items (TD2)

The test coverage items shall be the main and alternative scenarios identified from the test model.

© Standards Australia Limited/Standards New Zealand 2022

Documents
Document Sharing Hub

Full document details: AS NZS ISO IEC IEEE 29119.4-2022

5.2.9.3 Derive test cases (TD3)

Test cases for scenario testing shall be derived by covering each scenario (test coverage item) with at least one test case. The following steps shall be used during test case derivation:

a) Select a scenario to exercise in the current test case.

b) Identify input values to exercise the scenario covered by the test case.

c) Determine the expected result of the test case by applying the inputs to the test basis.

d) Repeat steps a) to c) until the required level of test coverage is achieved.

5.2.10 Random testing

5.2.10.1 Create test model (TD1)

Random testing shall use a representation of the input domain of the test item that defines the set of all possible input values. See BS 7925-2, Craig and Jaskiel 2002[8], and Kaner 1988[16] for more details.

5.2.10.2 Identify test coverage items (TD2)

There are no recognized test coverage items for random testing.

5.2.10.3 Derive test cases (TD3)

Test cases for random testing shall be chosen by randomly selecting input values from the input domain of the test item according to the chosen input distribution. The following steps shall be used during test case derivation:

a) Select an input distribution for the selection of test inputs.

EXAMPLE Types of input distributions include the normal distribution, uniform distribution and operational profile.

b) Generate random values for the test inputs based on the input distribution chosen in step a).

c) Determine the expected result of the test case by applying the inputs to the test basis.

d) Repeat steps b) and c) until the required testing has been completed.

NOTE 1 The required testing can be defined in terms of a number of tests executed, an amount of time spent testing or some other measure of completion.

NOTE 2 Step b) is typically automated.

5.2.11 Metamorphic testing

5.2.11.1 Create test model (TD1)

The test model for metamorphic testing shall be a set of metamorphic relations, which define properties of the test item in relation to multiple inputs and their expected outputs. See Chen 2018[23] for more details. The properties defined by the metamorphic relation allow follow-up test cases to be generated from a source test case. A metamorphic relation for a test item describes how a change in the test inputs from the source test case to the follow-up test case affects a change (or not) in the expected outputs from the source test case to the follow-up test case.

Documents Document Sharing Hub
© Standards Australia Limited/Standards New Zealand 2022
Full document details: AS NZS ISO IEC IEEE 29119.4-2022

EXAMPLE 1 A test item measures the distance between a start and end point. The source test case has test inputs A (start point) and B (end point) and an expected result C (distance) from running the test case. The metamorphic relation states that if the start and end points are swapped, then the expected result remains unchanged. Thus, a follow-up test case can be generated with B as the start point, A as the end point and C as the distance.

EXAMPLE 2 A test item predicts the age of death for an individual based on a set of lifestyle parameters. A source test case has various test inputs, including 10 cigarettes smoked per day, and an expected result of age 58 years from running the test case. The metamorphic relation states that if a person smokes more cigarettes, then their expected age of death will probably decrease (and not increase). Thus, a follow-up test case can be generated with the same input set of lifestyle parameters, except with the number of cigarettes smoked increased to 20 per day. The expected result (the predicted age of death) for this follow-up test case can now be set to less than or equal to 58 years.

NOTE In many situations, it is good practice to have identified metamorphic relations reviewed and agreed by relevant stakeholders, such as business users.

5.2.11.2 Identify test coverage items (TD2)

The test coverage items shall be the individual metamorphic relations, which comprise the test model.

5.2.11.3 Derive test cases (TD3)

Test cases for metamorphic testing shall be derived by covering each metamorphic relation (test coverage item) with at least one test case. The following steps shall be used during test case derivation:

a) Select a test coverage item (metamorphic relation) for inclusion in the current test case.

b) Create a source test case (or identify an existing source test case), execute it and validate the result.

c) Use the metamorphic relation to derive one or more follow-up test cases.

NOTE 1 The expected result for the follow-up test case will not always be an exact value but will often be described as a function of the actual result achieved by executing the source test case (e.g. expected result for follow-up test case > actual result for source test case).

NOTE 2 A single metamorphic relation can often be used to derive multiple follow-up test cases (e.g. a metamorphic relation for a function that translates speech into text can be used to generate multiple follow-up test cases with the same speech at different input volume levels but with the same text as the expected result).

d) Repeat steps a) to c) until the required level of test coverage is achieved.

5.2.12 Requirements-based testing

5.2.12.1 Create test model (TD1)

The test model for requirements-based testing shall be a set of atomic requirements. See Pinkster et al, 2006[24] for more details.

NOTE Requirements-based testing is typically performed when the requirements are specified as a list of atomic textual requirements, such as 'The system shall collect and store the date of birth of all registered users.'

5.2.12.2 Identify test coverage items (TD2)

The test coverage items shall be the individual atomic requirements, which comprise the test model.

© Standards Australia Limited/Standards New Zealand 2022

Documents Document Sharing Hub

Full document details: AS NZS ISO IEC IEEE 29119.4-2022

5.2.12.3 Derive test cases (TD3)

Test cases for requirements-based testing shall be derived by covering each atomic requirement (test coverage item) with at least one test case. The following steps shall be used during test case derivation:

a) Select a test coverage item (atomic requirement) for inclusion in the current test case.

b) Create a test case that exercises the requirement.

NOTE Test cases for requirements-based testing are typically positive test cases that confirm whether stated requirements are met. Negative test cases are normally derived using other, complementary, test techniques.

c) Repeat steps a) and b) until the required level of test coverage is achieved.

5.3 Structure-based test design techniques

5.3.1 Statement testing

5.3.1.1 Create test model (TD1)

A model of the source code of the test item, which identifies statements as either executable or non-executable, shall be identified. See BS 7925-2 and Myers 1979[19] for more details.

NOTE The developer's source code is typically used as the test model for statement testing.

5.3.1.2 Identify test coverage items (TD2)

Each executable statement shall be a test coverage item.

5.3.1.3 Derive test cases (TD3)

The following steps shall be followed during test case derivation:

a) Identify a control flow sub-path that reaches one or more executable statements that have not yet been executed during testing.

b) Determine the test inputs that will cause the control flow sub-path to be exercised.

c) Determine the expected result from exercising the control flow sub-path by applying the corresponding test inputs to the test basis.

d) Repeat steps a) to c) until the required level of test coverage is achieved.

5.3.2 Branch testing

5.3.2.1 Create test model (TD1)

A control flow model that identifies branches in the control flow of the test item shall be created. See BS 7925-2 for more details.

A branch is:

— a conditional transfer of control from any node in the control flow model to any other node; or

— an explicit unconditional transfer of control from any node to any other node in the control flow model; or

— when a test item has more than one entry point, a transfer of control to an entry point of the test item.

Documents Document Sharing Hub
© Standards Australia Limited/Standards New Zealand 2022
Full document details: AS NZS ISO IEC IEEE 29119.4-2022

NOTE 1 Complete branch testing covering 100 % of all branches requires all arcs (links or edges) in the control flow graph to be tested, including sequential statements between an entry and exit point that contains no decisions.

NOTE 2 Branch testing can require testing of both conditional and unconditional branches, including entry and exit points of a test item, depending on the level of test coverage required.

5.3.2.2 Identify test coverage items (TD2)

Each branch in the control flow model shall be a test coverage item.

NOTE Function and method calls are not identified as separate test coverage items in branch testing.

5.3.2.3 Derive test cases (TD3)

The following steps shall be followed during test case derivation:

a) Identify a control flow sub-path that reaches one or more branches that have not yet been executed during testing.

b) Determine the test inputs that will cause the control flow sub-path to be exercised.

c) Determine the expected result from exercising the control flow sub-path by applying the test inputs to the test basis.

d) Repeat steps a) to c) until the required level of test coverage is achieved.

If there are no branches in the test item then a single test case is still required to achieve full branch coverage.

If there are multiple entry points to the test item, sufficient test cases will be required to cover each entry point.

5.3.3 Decision testing

5.3.3.1 Create test model (TD1)

A control flow model of the test item that identifies decisions shall be created. Decisions are points in the test item where two or more possible outcomes (and hence sub-paths) may be taken by the control flow (BS 7925-2; Myers 1979[19]). Typical decisions are used for simple selections (e.g. if-then-else in source code), to decide when to exit loops (e.g. while-loop in source code) and in case (switch) statements (e.g. case-1-2-3-…-N in source code).

5.3.3.2 Identify test coverage items (TD2)

The decision outcomes from each decision shall be identified as test coverage items.

5.3.3.3 Derive test cases (TD3)

The following steps shall be followed during test case derivation:

a) Identify a control flow sub-path that reaches one or more decision outcomes that have not yet been executed during testing.

b) Determine the test inputs that will cause the control flow sub-path to be exercised.

c) Determine the expected result from exercising the control flow sub-path by applying the test inputs to the test basis.

d) Repeat steps a) to c) until the required level of test coverage is achieved.

© Standards Australia Limited/Standards New Zealand 2022

Documents
Document Sharing Hub

Full document details: AS NZS ISO IEC IEEE 29119.4-2022

If there are no decisions in the test model then a single test case is still required to achieve full decision coverage.

5.3.4 Branch condition testing

5.3.4.1 Create test model (TD1)

A control flow model of the test item that identifies decisions and conditions within decisions shall be created. See BS 7925-2 for more details.

5.3.4.2 Identify test coverage items (TD2)

In branch condition testing, the possible Boolean values (i.e. true or false) of the conditions within decisions and the decision outcomes from each decision shall be identified as test coverage items.

EXAMPLE The decision statement "if A OR B AND C then" contains six test coverage items related to the three logical operators and two test coverage items related to the two decision outcomes.

5.3.4.3 Derive test cases (TD3)

The following steps shall be followed during test case derivation:

a) Identify Boolean values of conditions within a decision that have not yet been executed during testing.

b) Determine the set of test inputs that will cause the decision to be exercised with the selected Boolean values of conditions.

c) Determine the expected result by applying the selected test inputs to the test basis.

d) Repeat steps a) to c) until all Boolean values of conditions within decisions have been exercised.

e) Identify a control flow sub-path that reaches one or more decision outcomes, if all of them have not yet been executed during testing.

f) Determine the test inputs that will cause the control flow sub-path to be exercised.

g) Determine the expected result from exercising the control flow sub-path by applying the test inputs to the test basis.

h) Repeat steps e) to g) until the required level of test coverage is achieved.

If there are no decisions in the test item then a single test case is still required to achieve full branch condition coverage.

5.3.5 Branch condition combination testing

5.3.5.1 Create test model (TD1)

A control flow model of the test item that identifies decisions and conditions within decisions shall be created. See BS 7925-2 for more details.

5.3.5.2 Identify test coverage items (TD2)

Each unique feasible combination of Boolean values of conditions within each decision shall be identified as a test coverage item. See BS 7925-2 and Myers 1979[19] for more details.

NOTE Where a decision includes a single condition, two possible "combinations" (true and false) are considered to exist.

Documents Document Sharing Hub
© Standards Australia Limited/Standards New Zealand 2022
Full document details: AS NZS ISO IEC IEEE 29119.4-2022

EXAMPLE The decision statement "if A OR B AND C then" contains 8 test coverage items corresponding to all possible true and false combinations of the three conditions A, B and C. In general, a minimum of 2^N test coverage items are required when there are *N* conditions within a decision.

5.3.5.3 Derive test cases (TD3)

The following steps shall be followed during test case derivation:

a) Identify a combination of Boolean values of conditions within a decision that have not yet been executed during testing.

b) Determine the set of test inputs that will cause the decision to be exercised with the selected combination of Boolean values of conditions.

c) Determine the expected result by applying the selected test inputs to the test basis.

d) Repeat steps a) to c) until the required level of test coverage is achieved.

If there are no decisions in the test item then a single test case is still required to achieve full branch condition combination coverage.

5.3.6 Modified condition/decision coverage (MCDC) testing

5.3.6.1 Create test model (TD1)

A control flow model of the test item that identifies decisions and conditions within decisions shall be created. See BS 7925-2 for more details.

5.3.6.2 Identify test coverage items (TD2)

Each unique feasible combination of individual Boolean values of conditions within a decision that allows a single Boolean condition to independently affect the outcome of the decision shall be identified as a test coverage item. Whether a condition independently affects the outcome of a decision is shown by varying only that condition while holding all other possible conditions within the decision to fixed states.

NOTE Where a decision includes a single condition, two possible "combinations" (true and false) are considered to exist and two test coverage items are identified.

EXAMPLE The decision statement "if A OR B AND C then" requires a minimum of four test coverage items to demonstrate that the three conditions can each independently affect the outcome of the decision. In general, a minimum of $N + 1$ test coverage items are required when there are *N* conditions within a decision.

5.3.6.3 Derive test cases (TD3)

The following steps shall be followed during test case derivation:

a) Identify a combination of Boolean values of conditions defined by a test coverage item within a decision that have not yet been executed during testing.

b) Determine the set of test inputs that will cause the decision to be exercised with the selected combination of Boolean values of conditions.

c) Determine the expected result by applying the selected test inputs to the test basis.

d) Repeat steps a) to c) until the required level of test coverage is achieved.

If there are no decisions in the test item then a single test case is still required to achieve full MCDC coverage.

© Standards Australia Limited/Standards New Zealand 2022

Documents
Document Sharing Hub

Full document details: AS NZS ISO IEC IEEE 29119.4-2022

5.3.7 Data flow testing

5.3.7.1 Create test model (TD1)

In data flow testing, a model of the test item shall be derived that identifies control flow sub-paths through the test item, within which each definition of a given variable is linked to subsequent uses of the same variable and within which there has been no intervening redefinition of the variable's value. See Burnstein 2003[5] for more details.

"Definitions" are where a variable is possibly given a new value (sometimes a definition will result in a variable retaining the same value as it had before). A "use" is an occurrence of a variable in which the variable is not given a new value; "uses" can be further distinguished as either "p-uses" (predicate-use) or "c-uses" (computation-use). A p-use denotes the use of a variable in determining the outcome of a condition (predicate) within a decision, such as a while-loop or if-then-else. A c-use occurs when a variable is used as an input to the computation of the definition of any variable or of an output.

There are a number of forms of data flow testing, which are all based on the same test model. The five forms defined in this document are: all-definitions testing, all-c-uses testing, all-p-uses testing, all-uses testing, and all-du-paths testing.

5.3.7.2 All-definitions testing

5.3.7.2.1 Identify test coverage items (TD2)

A set of control flow sub-paths from each variable definition to some use (either p-use or c-use) of that definition (with no intervening definitions) shall be identified as test coverage items.

5.3.7.2.2 Derive test cases (TD3)

The following steps shall be followed during test case derivation:

a) Identify a control flow sub-path from a variable definition to a subsequent use of that definition that has not yet been executed during testing.

b) Determine the test inputs that will cause the control flow sub-path from the identified definition to be exercised.

c) Determine the expected result from exercising the control flow sub-path by applying the test inputs to the test basis.

d) Repeat steps a) to c) until the required level of test coverage is achieved.

NOTE In practice, these steps can require automation.

5.3.7.3 All-c-uses testing

5.3.7.3.1 Identify test coverage items (TD2)

A set of control flow sub-paths from each variable definition to each c-use of that definition (with no intervening definitions) shall be identified as test coverage items.

5.3.7.3.2 Derive test cases (TD3)

The following steps shall be followed during test case derivation:

a) Identify a control flow sub-path from a variable definition to a subsequent c-use of that definition that has not yet been executed during testing.

Documents Document Sharing Hub
© Standards Australia Limited/Standards New Zealand 2022
Full document details: AS NZS ISO IEC IEEE 29119.4-2022

b) Determine the test inputs that will cause the control flow sub-path from the identified definition to the subsequent c-use to be exercised.

c) Determine the expected result from exercising the control flow sub-path by applying the test inputs to the test basis.

d) Repeat steps a) to c) until the required level of test coverage is achieved.

5.3.7.4 All-p-uses testing

5.3.7.4.1 Identify test coverage items (TD2)

A set of control flow sub-paths from each variable definition to each p-use of that definition (with no intervening definitions) shall be identified as test coverage items.

5.3.7.4.2 Derive test cases (TD3)

The following steps shall be followed during test case derivation:

a) Identify a control flow sub-path from a variable definition to a subsequent p-use of that definition that has not yet been executed during testing.

b) Determine the test inputs that will cause the control flow sub-path from the identified definition to the subsequent p-use to be exercised.

c) Determine the expected result from exercising the control flow sub-path by applying the test inputs to the test basis.

d) Repeat steps a) to c) until the required level of test coverage is achieved.

5.3.7.5 All-uses testing

5.3.7.5.1 Identify test coverage items (TD2)

A set of control flow sub-paths from each variable definition to every use (both p-use and c-use) of that definition (with no intervening definitions) shall be identified as test coverage items.

5.3.7.5.2 Derive test cases (TD3)

The following steps shall be followed during test case derivation:

a) Identify a control flow sub-path from a variable definition to a subsequent p-use or c-use of that definition that has not yet been executed during testing.

b) Determine the test inputs that will cause the control flow sub-path from the identified definition to the subsequent p-use or c-use to be exercised.

c) Determine the expected result from exercising the control flow sub-path by applying the test inputs to the test basis.

d) Repeat steps a) to c) until the required level of test coverage is achieved.

© Standards Australia Limited/Standards New Zealand 2022

Documents
Document Sharing Hub

Full document details: AS NZS ISO IEC IEEE 29119.4-2022

5.3.7.6 All-du-paths testing

5.3.7.6.1 Identify test coverage items (TD2)

The set of all loop-free control flow sub-paths from each variable definition to every use (both p-use and c-use) of that definition (with no intervening definitions) shall be identified as test coverage items.

NOTE All-du-paths testing requires all loop-free sub-paths from a variable definition to its use be tested to attempt to achieve 100 % test item coverage. This differs from all-uses testing, which requires only one path from each variable definition to its use to be tested to attempt to achieve 100 % test item coverage.

5.3.7.6.2 Derive test cases (TD3)

The following steps shall be followed during test case derivation:

a) Identify a control flow sub-path from a variable definition to a subsequent p-use or c-use of that definition that has not yet been executed during testing.

b) Determine the test inputs that will cause the control flow sub-path from the identified definition to the subsequent p-use or c-use to be exercised.

c) Determine the expected result from exercising the control flow sub-path by applying the test inputs to the test basis.

d) Repeat steps a) to c) until the required level of test coverage is achieved.

5.4 Experience-based test design techniques

5.4.1 Error guessing

5.4.1.1 Create test model (TD1)

Error guessing shall use a test model in the form of a checklist of potential defects to focus the testing. See BS 7925-2 and Myers 1979[19] for more details. The checklist is used to identify test inputs that may cause failures, if those defects exist in the test item. The checklist constitutes the test model.

NOTE The checklist of potential defects can be derived by various means, such as taxonomies of known errors, information contained in incident management systems, from a tester's knowledge, experience or understanding of the test item(s) or similar test items or from the knowledge of other stakeholders (e.g. system users or programmers).

5.4.1.2 Identify test coverage items (TD2)

There are no recognized test coverage items for error guessing.

5.4.1.3 Derive test cases (TD3)

Test cases for error guessing shall be derived by selecting a defect type from the checklist and deriving test cases that can detect that defect type in the test item, if it existed. The following steps shall be used during test case derivation:

a) Select a defect type for coverage by the test case.

b) Identify test input values that can be expected to cause a failure corresponding to the selected defect type.

c) Determine the expected result of the test case by applying the test inputs to the test basis.

d) Repeat steps a) to c) until the required testing has been completed (normally this would be covering an agreed number of checklist entries).

Documents Document Sharing Hub
© Standards Australia Limited/Standards New Zealand 2022
Full document details: AS NZS ISO IEC IEEE 29119.4-2022

6 Test coverage measurement

6.1 Overview

The coverage measures defined in this document are based on differing degrees of coverage that are achievable by test design techniques. Coverage levels can range from 0 % to 100 %. In each coverage calculation, a number of test coverage items may be infeasible. A test coverage item shall be defined to be infeasible if it can be shown to not be executable or impossible to be covered by a test case. The coverage calculation shall be defined as either counting or discounting infeasible items; this choice will typically be recorded in the test plan (defined in ISO/IEC/IEEE 29119-3). If a test coverage item is discounted, justification for its infeasibility will typically be recorded in a test report.

When calculating coverage for any technique, the following formula shall be used:

$$C = \left(\frac{N}{T} \times 100 \right) \%$$

where

C is the coverage achieved by a specific test design technique;

N is the number of test coverage items covered by executed test cases;

T is the total number of test coverage items identified by the test design technique.

Specific values for C, N and T for each technique are defined in 6.2 and 6.3. The set of coverage measures presented in 6.2 and 6.3 are designed to be used with the test design techniques presented in this document. They are not designed to be an exhaustive list; there may be other useful coverage measures that are used by organizations that are not mentioned in this clause.

EXAMPLE Other measures that can be required for assessing the completeness of testing include measuring the overall percentage of requirements that have been covered during testing.

6.2 Test measurement for specification-based test design techniques

6.2.1 Equivalence partition coverage

Coverage for equivalence partitioning shall be calculated using the following definitions:

— $C_{equivalence_partitions}$ is the equivalence partition coverage;

— N is the number of partitions covered by executed test cases;

— T is the total number of partitions identified.

6.2.2 Classification tree method coverage

Coverage for the classification tree method shall be calculated using the following definitions:

For minimality coverage, the following definitions shall be used:

— $C_{classification_tree_minimality}$ is the minimality classification tree method coverage;

— N is the number of classes covered by executed test cases;

— T is the total number of classes.

For maximality combination coverage, the following definitions shall be used:

— $C_{classification_tree_maximality}$ is the maximality classification tree method coverage;

© Standards Australia Limited/Standards New Zealand 2022

Documents
Document Sharing Hub

Full document details: AS NZS ISO IEC IEEE 29119.4-2022

— N is the number of unique combinations of classes covered by executed test cases;

— T is the total number of unique combinations of classes.

6.2.3 Boundary value analysis coverage

Coverage for boundary value analysis shall be calculated using the following definitions:

— $C_{\text{boundary_values}}$ is the boundary value coverage;

— N is the number of distinct boundary values covered by executed test cases;

— T is the total number of boundary values.

The decision to apply two-value or three-value boundary testing should be recorded.

6.2.4 Syntax testing coverage

There is currently no industry agreed approach to calculating coverage for syntax testing.

NOTE Calculating coverage as a percentage is not possible for syntax testing due to the potentially extremely large number of options and infinite number of mutations possible.

6.2.5 Combinatorial test design techniques coverage

6.2.5.1 All combinations testing coverage

Coverage for all combinations testing shall be calculated using the following definitions:

— $C_{\text{all_combinations}}$ is all combinations coverage;

— N is the number of unique combinations of P-V pairs covered by executed test cases;

— T is the total number of unique P-V pair combinations (the product of the number of P-V pairs for each test item parameter).

6.2.5.2 Pair-wise testing coverage

Coverage for pair-wise testing shall be calculated using the following definitions:

— $C_{pair\text{-}wise}$ is pair-wise coverage;

— N is the number of unique pairs of P-V pairs covered by executed test cases;

— T is the total number of unique pairs of P-V pairs.

6.2.5.3 Each choice testing coverage

Coverage for each choice testing shall be calculated using the following definitions:

— $C_{\text{each_choice}}$ is each choice coverage;

— N is the number of P-V pairs covered by executed test cases;

— T is the total number of unique P-V pairs.

6.2.5.4 Base choice testing coverage

Coverage for base choice testing shall be calculated using the following definitions:

— $C_{\text{base_choice}}$ is base choice coverage;

Documents Document Sharing Hub
© Standards Australia Limited/Standards New Zealand 2022
Full document details: AS NZS ISO IEC IEEE 29119.4-2022

— N is the number of base choice combinations covered by executed test cases (all but one test item parameter set to the base value, and the remaining test item parameter set to valid values), plus one (for when all test item parameters are set to the base value) if exercised;

— T is the total number of base choice combinations (all but one test item parameter set to the base value, and the remaining test item parameter set to valid values), plus one (for when all test item parameters are set to the base value).

6.2.6 Decision table testing coverage

Coverage for decision table testing shall be calculated using the following definitions:

— $C_{\text{decision_table}}$ is decision table coverage;

— N is the number of decision rules covered by executed test cases;

— T is the total number of feasible decision rules.

6.2.7 Cause-effect graphing coverage

Coverage for cause-effect graphing shall be calculated using the following definitions:

— $C_{\text{cause-effect}}$ is the cause-effect coverage;

— N is the number of decision rules covered by executed test cases;

— T is the total number of feasible decision rules.

6.2.8 State transition testing coverage

Coverage for all states testing shall be calculated using the following definitions:

— $C_{\text{all_states}}$ is all states coverage;

— N is the number of states covered by executed test cases;

— T is the total number of states.

Coverage for single transitions (0-switch coverage) shall be calculated using the following definitions:

— $C_{\text{0-switch}}$ is 0-switch coverage;

— N is the number of single valid transitions executed by test cases;

— T is the total number of single valid transitions.

Coverage for all transitions shall be calculated using the following definitions:

— $C_{\text{all_transitions}}$ is all transitions coverage;

— N is the number of valid and invalid transitions attempted to be covered by executed test cases;

— T is the total number of valid and invalid transitions between identified states initiated by valid events.

Coverage for $N + 1$ transitions (N-switch testing) shall be calculated using the following definitions:

— $C_{\text{N-switch}}$ is N-switch coverage;

— N is the number of $N + 1$ valid transitions covered by executed test cases;

— T is the total number of sequences of $N + 1$ valid transitions.

© Standards Australia Limited/Standards New Zealand 2022

Documents
Document Sharing Hub

Full document details: AS NZS ISO IEC IEEE 29119.4-2022

6.2.9 Scenario testing coverage

Coverage for scenario testing (including use case testing) shall be calculated using the following definitions:

— $C_{\text{scenarios}}$ is scenario coverage;

— N is the number of scenarios covered by executed test cases;

— T is the total number of scenarios.

6.2.10 Random testing coverage

There is currently no industry agreed approach to calculating coverage for random testing.

6.2.11 Metamorphic testing coverage

There is currently no industry agreed approach to calculating coverage for metamorphic testing.

NOTE Calculating coverage as a percentage is often not possible for metamorphic testing due to the potentially extremely large number of follow-up test cases that can be derived from a single metamorphic relation.

6.2.12 Requirements-based testing coverage

Coverage for requirements-based testing shall be calculated using the following definitions:

— $C_{\text{requirements}}$ is requirement coverage;

— N is the number of atomic requirements covered by executed test cases;

— T is the total number of atomic requirements.

6.3 Test measurement for structure-based test design techniques

6.3.1 Statement testing coverage

Coverage for statement testing shall be calculated using the following definitions:

— $C_{\text{statement}}$ is statement coverage;

— N is the number of executable statements covered by executed test cases;

— T is the total number of executable statements.

6.3.2 Branch testing coverage

Coverage for branch testing shall be calculated using the following definitions:

— C_{branch} is branch coverage;

— N is the number of branches covered by executed test cases;

— T is the total number of branches.

For situations where there are no branches in the test item, a single test is required to achieve 100 % branch coverage.

Documents Document Sharing Hub
© Standards Australia Limited/Standards New Zealand 2022
Full document details: AS NZS ISO IEC IEEE 29119.4-2022

6.3.3 Decision testing coverage

Coverage for decision testing shall be calculated using the following definitions:

— $C_{decision}$ is decision coverage;

— N is the number of decision outcomes covered by executed test cases;

— T is the total number of decision outcomes.

For situations where there are no decisions in the test item, a single test is required to achieve 100 % decision coverage.

6.3.4 Branch condition testing coverage

Coverage for branch condition testing shall be calculated using the following definitions:

— $C_{branch_condition}$ is branch condition coverage;

— N is the number of Boolean values of conditions within decisions plus the number of decision outcomes covered by executed test cases;

— T is the total number of Boolean values of conditions within decisions plus the total number of decision outcomes.

For situations where there are no decisions in the test item, a single test is required to achieve 100 % branch condition coverage.

6.3.5 Branch condition combination testing coverage

Coverage for branch condition combination testing shall be calculated using the following definitions:

— $C_{branch_condition_combination}$ is branch condition combination coverage;

— N is the number of combinations of Boolean values of conditions within each decision covered by executed test cases;

— T is the total number of unique combinations of Boolean values of conditions within decisions.

For situations where there are no decisions in the test item, a single test is required to achieve 100 % branch condition combination coverage.

6.3.6 Modified condition/decision coverage (MCDC)

Coverage for modified condition/decision coverage testing shall be calculated using the following definitions:

— C_{MCDC} is modified condition/decision coverage;

— N is the number of unique feasible combinations of individual Boolean values of conditions within decisions that allow a single Boolean condition to independently affect the decision outcome covered by executed test cases;

— T is the total number of unique combinations of individual Boolean values of conditions within decisions that allow a single Boolean condition to independently affect the decision outcome.

For situations where there are no decisions in the test item, a single test is required to achieve 100 % modified condition/decision coverage.

© Standards Australia Limited/Standards New Zealand 2022

Documents
Document Sharing Hub

Full document details: AS NZS ISO IEC IEEE 29119.4-2022

6.3.7 Data flow testing coverage

6.3.7.1 All-definitions testing coverage

Coverage for all-definitions testing shall be calculated using the following definitions:

— $C_{\text{all-definitions}}$ is all-definitions coverage;

— N is the number of definitions associated with definition-use pairs covered by executed test cases;

— T is the total number of definition-use pairs from distinct variable definitions.

6.3.7.2 All-c-uses testing coverage

Coverage for all-c-uses testing shall be calculated using the following definitions:

— $C_{\text{all-c-uses}}$ is all-c-uses coverage;

— N is the number of data definition c-use pairs covered by executed test cases;

— T is the total number of data definition c-use pairs.

6.3.7.3 All-p-uses testing coverage

Coverage for all-p-uses testing shall be calculated using the following definitions:

— $C_{\text{all-p-uses}}$ is all-p-uses coverage;

— N is the number of data definition p-use pairs covered by executed test cases;

— T is the total number of data definition p-use pairs.

6.3.7.4 All-uses testing coverage

Coverage for all-uses testing shall be calculated using the following definitions:

— $C_{\text{all-uses}}$ is all-uses coverage;

— N is the number of definition-use pairs covered by executed test cases;

— T is the total number of definition-use pairs from each definition to all uses of that definition.

6.3.7.5 All-du-paths testing coverage

Coverage for all-du-paths testing shall be calculated using the following definitions:

— $C_{\text{all-du-paths}}$ is all-du-paths coverage;

— N is the number of loop-free sub-paths from each variable definition to each of its uses that can be reached (without an intervening definition of the variable) covered by executed test cases;

— T is the total number of loop-free control flow sub-paths from each variable definition to every use (both p-use and c-use) of that definition (with no intervening definitions).

6.4 Test measurement for experience-based testing design techniques — Error guessing coverage

There is currently no industry agreed approach to calculating coverage for error guessing.

Documents Document Sharing Hub
© Standards Australia Limited/Standards New Zealand 2022
Full document details: AS NZS ISO IEC IEEE 29119.4-2022