International Standard

ISO/IEC/IEEE 29119-5

Second edition
2024-12

Software and systems engineering — Software testing —

Part 5: Keyword-driven testing

Ingénierie du logiciel et des systèmes — Essais du logiciel —

Partie 5: Essais axés sur des mots-clés

Reference number
ISO/IEC/IEEE 29119-5:2024(en)

COPYRIGHT PROTECTED DOCUMENT

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

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

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

Published in Switzerland

Contents

Page

Foreword

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

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of 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.

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

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.

This document 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 29119-5:2016), which has been technically revised.

The main changes are as follows:

— updated reference to processes (test design and implementation process) according to ISO/IEC/IEEE 29119-2:2021;

— updated reference to documents (test procedure) according to ISO/IEC/IEEE 29119-3:2021;

— included definitions for terms used in this document, based on ISO/IEC/IEEE 29119-1:2022.

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

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

Introduction

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

This document defines a unified approach for describing test cases in a modular way, which assists with the creation of items like keyword-driven test specifications and test automation frameworks. The term "keyword" refers to the elements which are, once defined, used to compose test cases, such as with building blocks. This document explains the main concepts and application of keyword-driven testing. It also defines attributes of frameworks designed to support keyword-driven testing.

The concepts relating to software testing defined in ISO/IEC/IEEE 29119-1 are also applicable to this document.

The test process model on which the keyword-driven testing framework is based is defined in ISO/IEC/IEEE 29119-2. It comprises test process descriptions that define the software testing processes at the organizational level, test management level and dynamic test level. Supporting diagrams describing the processes are also provided in ISO/IEC/IEEE 29119-2. However, this document describes a specific implementation of the test design and implementation process of ISO/IEC/IEEE 29119-2 for application in keyword-driven testing, in particular in TD3 (6.6.4) and TD4 (6.6.5).

The templates and examples of test documentation as defined in ISO/IEC/IEEE 29119-3 are also applicable to this document.

Software test design techniques that can be used during test design are defined in ISO/IEC/IEEE 29119-4. The application of ISO/IEC/IEEE 29119-4 is assumed when designing test cases that are then described by keywords according to this document.

Software and systems engineering — Software testing —

Part 5: **Keyword-driven testing**

1 Scope

This document defines an efficient and consistent solution for keyword-driven testing by:

— giving an introduction to keyword-driven testing;

— providing a reference approach to implement keyword-driven testing;

— defining requirements on frameworks for keyword-driven testing to enable testers to share their work items, such as test cases, test data, keywords, or complete test specifications;

— defining requirements for tools that support keyword-driven testing; these requirements are applicable to any tool that supports the keyword-driven approach (e.g. test automation, test design and test management tools);

— defining interfaces and a common data exchange format to ensure that tools from different vendors can exchange their data (e.g. test cases, test data and test results);

— defining levels of hierarchical keywords, and advising use of hierarchical keywords; this includes describing specific types of keywords (e.g. keywords for navigation or for checking a value) and when to use "flat" structured keywords;

— providing an initial list of example generic technical (low-level) keywords, such as "inputData" or "checkValue"; these keywords can be used to specify test cases on a technical level and can be combined to create business-level keywords as required.

This document is applicable to all those who want to create keyword-driven test specifications, create corresponding frameworks, or build test automation based on keywords.

2 Normative references

There are no normative references in this document.

3 Terms and definitions

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

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

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

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

— IEEE Standards Dictionary Online: available at https://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
actual result
set of behaviours or conditions of a *test item* (3.23), or set of conditions of associated data or the *test environment* (3.17), observed as a result of *test execution* (3.18)

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

[SOURCE: ISO/IEC/IEEE 29119-1:2022, 3.5]

3.2
composite keyword
keyword (3.8) that comprises two or more other keywords

3.3
data sequencer
component that replaces *keyword* (3.8) parameters with data, if necessary, across several hierarchy levels

3.4
decomposer
component that deconstructs the *high-level keywords* (3.7) across all levels into a sequence of *low-level keywords* (3.14) that eventually comprise it

3.5
domain layer
highest level of abstraction for the *test item* (3.23)

Note 1 to entry: *Keywords* (3.8) on this level are chosen in a way that is familiar to domain experts.

3.6
expected result
observable predicted behaviour of the *test item* (3.23) under specified conditions based on its specification or another source

[SOURCE: ISO/IEC/IEEE 29119-1:2022, 3.35]

3.7
high-level keyword
keyword (3.8) that covers complex activities that can be composed from other keywords and is used by domain experts to assemble *keyword test cases* (3.13)

3.8
keyword
one or more words used as a reference to a specific set of actions intended to be performed during the execution of one or more *test cases* (3.16)

Note 1 to entry: The actions include interactions with the user interface during the test, verification, and specific actions to set up a test scenario.

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

Note 3 to entry: *Composite keywords* (3.2) can be constructed based on other keywords.

Note 4 to entry: Keywords often have parameters so they can be used with different data.

3.9
keyword library
keyword dictionary
repository containing a set of *keywords* (3.8) reflecting the language and level of abstraction used to write *test cases* (3.16)

3.10
keyword-driven testing
testing (3.27) using *test cases* (3.16) composed from *keywords* (3.8)

3.11
keyword-driven testing framework
test framework (3.20) covering the functional capabilities of a keyword-driven editor, *decomposer* (3.4), *data sequencer* (3.3), *keyword library* (3.9), data repository, *test environment* (3.17), and including organizational processes that define their interaction and use

Note 1 to entry: For manual *keyword-driven testing* (3.10) the functional capabilities of a manual test assistant are also covered, while for automated keyword-driven testing the functional capabilities of a *test execution engine* (3.19), *tool bridge* (3.28) and script repository are also covered.

3.12
keyword execution code
implementation of a *keyword* (3.8) that is intended to be executed by a *test execution engine* (3.19)

3.13
keyword test case
sequence of *keywords* (3.8) and the required values for their associated parameters (as applicable) that are composed to describe the actions of a *test case* (3.16)

3.14
low-level keyword
keyword (3.8) that covers only one or very few simple actions and is not composed from other keywords

3.15
manual testing
humans performing tests by entering information into a *test item* (3.23) and verifying the results

Note 1 to entry: Automated testing uses tools, robots, and other *test execution engines* (3.19) to perform tests. Manual testing does not use these items.

[SOURCE: ISO/IEC/IEEE 29119-1:2022, 3.46]

3.16
test case
set of preconditions, inputs and *expected results* (3.6), developed to drive the execution of a *test item* (3.23) to meet *test objectives* (3.24)

[SOURCE: ISO/IEC/IEEE 29119-1:2022, 3.85, modified — Notes 1 to 3 to entry have been removed.]

3.17
test environment
environment containing facilities, hardware, software, firmware, procedures, needed to conduct a test

Note 1 to entry: A test environment can contain multiple environments to accommodate specific test level or test types (e.g. a unit test environment, a performance test environment).

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

[SOURCE: ISO/IEC/IEEE 29119-1:2022, 3.95]

3.18
test execution
process of running a test on the *test item* (3.23), producing *actual results* (3.1)

[SOURCE: ISO/IEC/IEEE 29119-1:2022, 3.99]

3.19
test execution engine
tool that executes *test cases* (3.16) by passing data to the *test item* (3.23), triggering the test item to run and receiving data from the test item

Note 1 to entry: A typical test execution engine can be a part of a *test framework* (3.20), a capture and playback tool or a hardware robot.

3.20
test framework
structured set of principles, guidelines, and practices used for organizing, selecting and communicating *testing* (3.27)

[SOURCE: ISO/IEC/IEEE 29119-1:2022, 3.103]

3.21
test interface
interface to the *test item* (3.23) used to either stimulate the test item, to get responses [e.g. *actual results* (3.1)], or both

Note 1 to entry: The graphical user interface (GUI), application programming interface (API) or service-oriented architecture (SOA) interfaces are typical test interfaces.

Note 2 to entry: Stimulating the test item can involve passing data into it via computer interfaces or attached hardware.

Note 3 to entry: Getting responses includes getting information from the test item or associated hardware.

3.22
test interface layer
lowest level of abstraction for *keywords* (3.8), which interacts with the *test item* (3.23) directly and encapsulates the atomic (lowest level) interactions at the *test interface* (3.21)

3.23
test item
test object
work product to be tested

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

[SOURCE: ISO/IEC/IEEE 29119-1:2022, 3.107]

3.24
test objective
reason for performing *testing* (3.27)

[SOURCE: ISO/IEC/IEEE 29119-1:2022, 3.114, modified — EXAMPLE has been removed.]

3.25
test procedure
sequence of *test cases* (3.16) in execution order, with any associated actions required to set up preconditions and perform wrap-up activities post execution

[SOURCE: ISO/IEC/IEEE 29119-1:2022, 3.120]

3.26
test result
indication of whether or not a specific *test case* (3.16) has passed or failed, i.e. if the *actual results* (3.1) correspond to the *expected results* (3.6) or if deviations were observed

[SOURCE: ISO/IEC/IEEE 29119-1:2022, 3.122]

3.27
testing
set of activities conducted to facilitate discovery and evaluation of properties of *test items* (3.23)

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

[SOURCE: ISO/IEC/IEEE 29119-1:2022, 3.131]

3.28
tool bridge
component that maps and transforms the *keywords* (3.8) to code and data to be executed by the *test execution engine* (3.19)

4 Conformance

4.1 Intended usage

The requirements in this document are contained in Clause 7.

This document provides requirements on the components of frameworks supporting the application of keyword-driven testing. It also provides requirements on conventions for the definition of keywords.

It is recognized that some organizations, projects, or teams may not need to use all of the components defined in this document. Therefore, implementation of this document typically involves selecting a set of components or parts of components suitable for the organization or project.

There are two ways that an implementation can be claimed to conform to 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 providing evidence that all of the keyword-driven testing requirements defined in Clause 7 have been met.

4.3 Tailored conformance

When this document is used for implementing components of frameworks that do not qualify for full conformance, the subset of components for which tailored conformance is claimed should be recorded. Tailored conformance is achieved by demonstrating that all of the requirements for the recorded subset of components defined in Clause 7 have been satisfied.

Where tailoring occurs, the justification shall be provided, either directly or by reference, whenever a requirement defined in Clause 7 is not followed. All tailoring decisions shall be recorded with their rationale, including the consideration of any applicable risks. Tailoring decisions shall be agreed to by the relevant stakeholders.

EXAMPLE A tool vendor provides only part of a keyword-driven testing framework in its portfolio, and thus decides not to implement requirements that are covered by complementary tools (e.g. a vendor only provides an execution engine, but no keyword-driven editor – then the execution engine can still conform to this document).

5 Introduction to keyword-driven testing

5.1 Overview

Keyword-driven testing is a test case specification approach that is commonly used to support test automation and the development of test automation frameworks. However, it can also be used if no automation approach is planned or established.

In principle, keyword-driven testing can be applied at all testing levels (e.g. component testing, system testing) and all types of testing (e.g. functional testing, reliability testing). Keyword-driven testing benefits include the following:

— ease of use;

— understandability;

— maintainability;

— test information reuse;

— support of test automation;

— potential cost and schedule savings.

The fundamental idea in keyword-driven testing is to provide a set of "building blocks", referred to as keywords, that can be used to create manual or automated test cases without requiring detailed knowledge of programming or test tool expertise. The ultimate goal is to provide a basic, unambiguous set of keywords comprehensive enough so that most, if not all, required test cases can be entirely composed of these keywords. The vocabulary included in these dictionaries or libraries of keywords is, therefore, a reflection of the language and level of abstraction used to write the test cases, and not of any standard computer programming language.

For test automation, each keyword must be implemented in software.

NOTE 1 When keywords are not used, test cases are usually written using natural language or written in a computer programming language. Compared with natural language, keywords have the advantage of being less ambiguous and more precise. Compared with a computer programming language, when keywords are well-defined and well-structured, they have the advantage of being understandable by many people who do not have software engineering skills.

A keyword is usually defined at the following two levels.

— At a low level, each keyword is associated with a detailed set of one or more actions that describe the exact steps that are to be performed.

— At a high level, each keyword is ideally assigned a name which is meaningful to non-technical users. This keyword can require a set of input parameters, which would also belong to this level in the structure. The keyword and the parameters together comprise a high-level description of the actions associated with a test case.

Thus, instead of describing each individual action that needs to be taken in each test case, tests can be defined at a higher level of abstraction using keywords. Higher levels of abstraction can be achieved by using composite keywords, which are comprised of other keywords to describe associated actions.

An example of the benefits obtained from both manual and automated keyword-driven test cases is enhanced maintainability. Consider a case where the precise set of actions to carry out a commonly repeated operation has changed. The modularity introduced by keywords allows modification of only the actions for the changed operation in the relevant lower-level keyword, leaving the test cases untouched. Without modularity, it can be necessary to modify each occurrence of this operation in all of the test cases.

Modularization has helped popularize this approach. If test automation is required, a framework can be created to interpret manually created keyword test cases as executable test automation scripts. This is achieved by implementing test automation code for each keyword (e.g. keyword execution code).

NOTE 2 Testing tools can be used to support keyword-driven testing, but the available tools can be limited in their capability to support all the concepts described in this document.

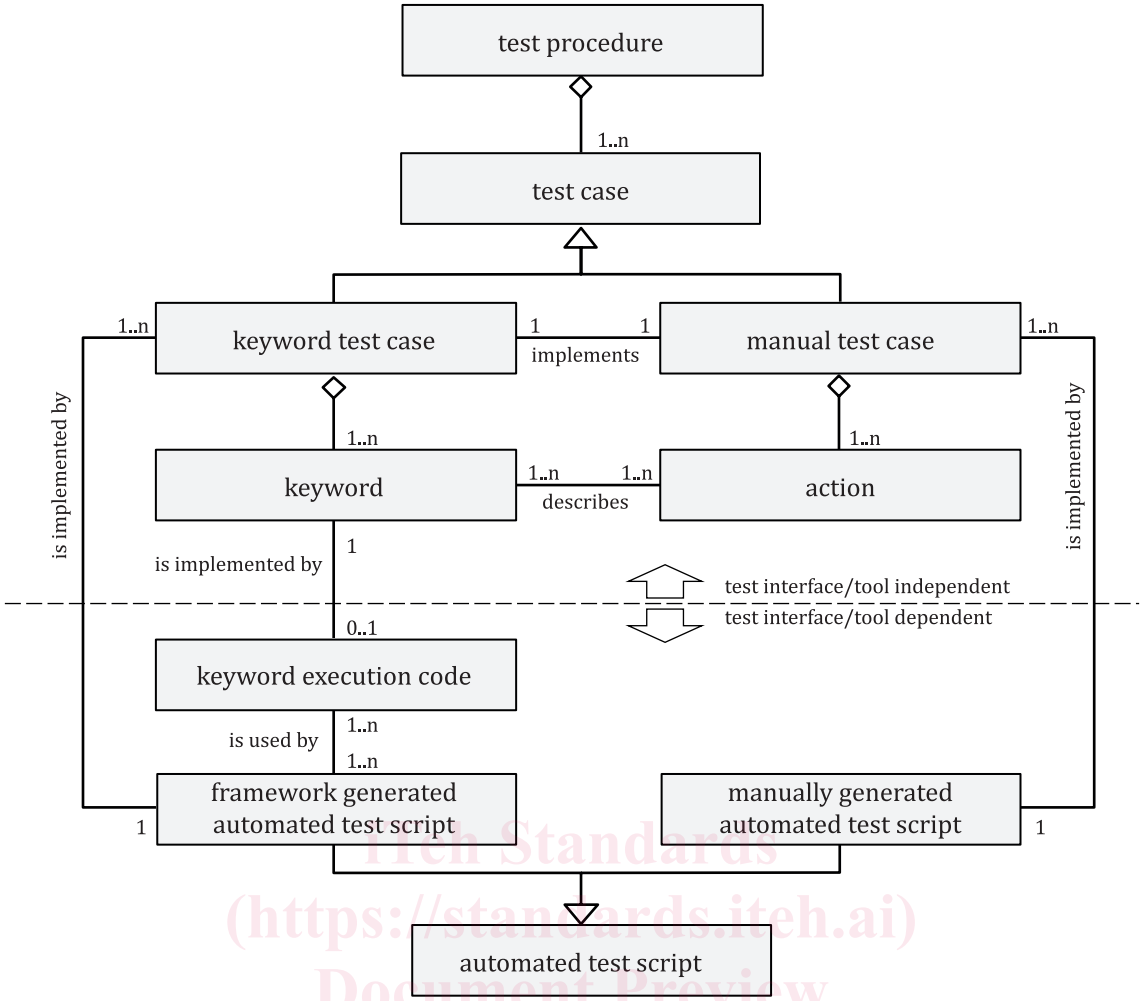

Figure 1 — Relationships between keyword-driven testing entities

A test procedure can have multiple test cases in it, and a test case can, in turn, be part of different test procedures as shown in Figure 1. Test cases can be either manual test cases or keyword test cases. A keyword test case implements a manual test case.

A keyword test case is typically composed of a sequenced series of keywords. Keywords should be chosen to be modular and generic so that they can be reused in many test cases. Keywords can also be used more than once in the same test case. A test case is composed from test actions. Keywords represent test actions.

NOTE 3 It is possible to map several keywords to a single action. It is also possible to define keywords in a way that each keyword represents one action. In these cases, a one-to-one relationship exists between actions and keywords. However, a test designer can decide to structure keywords in a different way (e.g. use more than one keyword to implement an action, or to combine two or more actions into one keyword). Thus, this relationship is not a 1:1 relationship in Figure 1.

Test automation is an option that can be chosen when implementing keyword-driven testing, but a manual approach is also possible. If keyword test cases are automated, each keyword is implemented by keyword execution code. Keyword execution code is specific to the chosen tool or test execution engine and additionally depends on the test interface. For the manual approach, the action described by a keyword is executed manually, so there is no keyword execution code. That is why in Figure 1 the relationship between keyword and keyword execution code is 1 to 0..1.

Test automation is typically highly technical and tool-dependent since it depends on the test interface and on the capabilities of the available tools. In general, keywords can be independent of the test interface (e.g. user interface) and the tools used to execute the test cases.

In this context, automated test scripts can either be generated automatically by a framework or developed manually by a test automation specialist. Automated test scripts are typically developed by testers with programming experience.

NOTE 4 When developing automated test scripts, it is beneficial to align the structure (e.g. levels) of the keywords with the implementation of the automated test scripts.

If a keyword test case or a set of keyword test cases is automated, the framework for keyword-driven testing generates the automated test script based on the keyword execution code.

NOTE 5 A framework for keyword-driven testing does not necessarily "generate" code. The required code can also be prepared by testers and be executed by the framework.

Annex B provides an overview of advantages and disadvantages of keyword-driven testing.

5.2 Layers in keyword-driven testing

5.2.1 Overview

Keywords can represent actions at different abstraction levels. For example, one keyword can refer to a very complex set of activities, like the creation of a contract, which includes a lot of steps, while another keyword can refer to a very simple action, like pressing a button on a graphical user interface. The first keyword is close to the business and end user domain, while the second is closer to the test interface. Keywords that are written at a similar level of detail, and have a similar relationship to the stakeholder's view, are said to belong to the same abstraction layer.

Keyword-driven testing can be organized by using one or more layers. Typical layers are the end user domain layer and the test interface layer.

While many implementations of keyword-driven testing comprise two or three abstraction layers, in some cases it can be necessary to structure keywords in more layers.

The topmost layer is the most abstract layer, which is generally aligned with the wording of the application's users. In practice, the topmost layer is usually the domain layer. However, in some situations the domain layer is not required, and another, more abstract layer is used (e.g. if the test cases are supposed to span several different end user domains, a meta domain layer can be introduced).

The lowest layer is the most detailed layer. It is commonly aligned with the names of test interface elements (e.g. "SelectMenu"). In practice, this layer is usually, but not always, the test interface layer (e.g. as sometimes a test interface layer is not required, or for specific reasons, even more detailed layers can be used).

Most keyword-driven test systems have more than one layer due to factors such as having understandable keyword test cases, maintainability and division of work relying on a multi-layer structure. If only one layer is implemented, it is commonly either at a low level, which affects the readability of the keyword test cases, or at a high level, which can result in more keyword execution code.

In Figure 2, an example is given, showing how test cases for two different test interfaces can be structured by using two layers of keywords.