INTERNATIONAL STANDARD

ISO/IEC/IEEE 29119-5

First edition
2016-11-15

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 29119-5:2016(E)

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

IEC Central Office 3,
rue de Varembé CH-
1211 Geneva 20
Switzerland
E-mail inmail@iec.ch
Web www.iec.ch

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

stds.ipr@ieee.org
www.ieee.org

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. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of ISO/IEC JTC 1 is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is called to the possibility that implementation of this standard may require the use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. ISO/IEEE is not responsible for identifying essential patents or patent claims for which a license may be required, for conducting inquiries into the legal validity or scope of patents or patent claims or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance or a Patent Statement and Licensing Declaration Form, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from ISO or the IEEE Standards Association.

ISO/IEC/IEEE 29119-5 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 7, *Software and systems engineering*, in cooperation with the Software & Systems Engineering Standards Committee of the IEEE Computer Society, under the Partner Standards Development Organization cooperation agreement between ISO and IEEE.

ISO/IEC/IEEE 29119 consists of the following parts, under the general title *Software and systems engineering — Software testing:*

— *Part 1: Concepts and definitions*

— *Part 2: Test processes*

— *Part 3: Test documentation*

— *Part 4: Test techniques*

— *Part 5: Keyword-Driven Testing*

Introduction

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

This part of ISO/IEC/IEEE 29119 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. ISO/IEC/IEEE 29119-5 will explain the main concepts and application of Keyword-Driven Testing. It will also define attributes of frameworks designed to support Keyword-Driven Testing.

The concepts and definitions relating to software testing defined in ISO/IEC/IEEE 29119-1 are also applicable to ISO/IEC/IEEE 29119-5.

The test process model on which the Keyword-Driven Testing framework is based is defined in ISO/IEC/IEEE 29119-2 Test Processes. It comprises test process descriptions that define the software testing processes at the organizational level, test management level and dynamic test level. Supporting informative diagrams describing the processes are also provided in ISO/IEC/IEEE 29119-2. However, ISO/IEC/IEEE 29119-5 describes a specific implementation of the test design and implementation processes of ISO/IEC/IEEE 29119-2; in particular TD4 (Derive Test Cases), TD5 (Assemble Test Sets) and TD6 (Derive Test Procedures) as here applied to Keyword-Driven Testing.

The templates and examples of test documentation as defined in ISO/IEC/IEEE 29119-3 are also applicable to ISO/IEC/IEEE 29119-5.

Software test design techniques that can be used during test design are defined in ISO/IEC/IEEE 29119-4 Test Techniques. The application of ISO/IEC/IEEE 29119-4 is assumed when designing test cases that are then described by keywords according to ISO/IEC/IEEE 29119-5.

This part of ISO/IEC/IEEE 29119-5 has the following structure:

— terms and definitions can be found in clause 4

— an introduction to Keyword-Driven Testing is given in clause 5

— the application of Keyword-Driven Testing is explained in clause 6

— frameworks for Keyword-Driven Testing are described in clause 7

— data interchange is covered in clause 8

— Annex A states naming conventions for keywords

— Annex B names benefits that can be achieved with Keyword-Driven Testing

— Annex C gives advice on how interested parties wanting to use Keyword-Driven Testing can start

— Annex D describes roles that can be used in Keyword-Driven Testing

— Annex E contains examples of basic keywords that can be used to create test cases

— Annex F contains examples for keyword test cases

Software and Systems Engineering — Software Testing — Part 5: Keyword-Driven Testing

1 Scope

This part of ISO/IEC/IEEE 29119 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 could apply 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 may be combined to create business-level keywords as required.

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

2 Conformance

2.1 Intended usage

The requirements in ISO/IEC/IEEE 29119-5 are contained in Clause 7 and in Annex A. ISO/IEC/IEEE 29119-5 provides requirements on frameworks supporting the application of Keyword-Driven Testing. It is recognized that particular projects or organizations may not need to use all of the components defined in this standard. Therefore, implementation of ISO/IEC/IEEE 29119-5 typically involves selecting a set of components or parts of components suitable for the organization or project. There are two ways that an organization can claim to conform to the provisions of this standard.

The organization or individual shall assert whether full or tailored conformance to this standard is claimed.

2.2 Full conformance

Full conformance is achieved by demonstrating that all of the Keyword-Driven Testing requirements (i.e. shall statements) defined in ISO/IEC/IEEE 29119-5 have been satisfied.

2.3 Tailored conformance

When ISO/IEC/IEEE 29119-5 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 (i.e. shall statements) for the recorded subset of components have been satisfied.

Where tailoring occurs, the justification shall be provided, either directly or by reference, whenever a requirement defined in clauses 7 and Annex A of ISO/IEC/IEEE 29119-5 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 Tool vendors may in their portfolio provide only part of a keyword-driven test framework, and thus decide 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 be conforming with the standard).

3 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. 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 24765, *Systems and software engineering – Vocabulary*

ISO/IEC/IEEE 29119-1, *Software and systems engineering – Software Testing – Part 1: Concepts and Definitions*

ISO/IEC/IEEE 29119-2, *Software and systems engineering – Software Testing – Part 2: Test Processes*

ISO/IEC/IEEE 29119-3, *Software and systems engineering – Software Testing – Part 3: Test Documentation*

ISO/IEC/IEEE 29119-4, *Software and systems engineering – Software Testing – Part 4: Test Techniques*

Other standards useful for the implementation and interpretation of this standard are listed in the bibliography.

4 Terms and definitions

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

NOTE Use of the terminology of ISO/IEC/IEEE 29119-5 is for ease of reference and is not mandatory for conformance with ISO/IEC/IEEE 29119-5. The following terms and definitions are provided to assist with the understanding and readability of ISO/IEC/IEEE 29119-5. Only terms critical to the understanding of ISO/IEC/IEEE 29119-5 are included. This clause is not intended to provide a complete list of testing terms. The Systems and Software Engineering vocabulary ISO/IEC/IEEE 24765 can be referenced for terms not defined in this standard. ISO/IEC/IEEE 29119-1 can be referenced for terms related to software testing in general. ISO/IEC/IEEE 29119-5 only defines terms specific to Keyword-Driven Testing.

4.1
domain layer
highest level of abstraction for the test item

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

4.2
high-level keyword
keyword that covers complex activities that may be composed from other keywords and is used by domain experts to assemble keyword test cases

4.3
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

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 can be constructed based on other keywords.

4.4
keyword dictionary
keyword library
repository containing a set of keywords reflecting the language and level of abstraction used to write test cases

4.5
Keyword-Driven Testing
testing using test cases composed from keywords

4.6
Keyword-Driven Testing framework
test framework covering the functional capabilities of a keyword-driven editor, decomposer, data sequencer, manual test assistant, tool bridge, data and script repositories, a keyword library and the test execution environment

4.7
keyword execution code
implementation of a keyword that is intended to be executed by a test execution engine

4.8
keyword test case
sequence of keywords and the required values for their associated parameters (as applicable) that are composed to describe the actions of a test case

4.9
low-level keyword
keyword that covers only one or very few simple actions and is not composed from other keywords

4.10
manual testing
humans performing tests by entering information into a test item and verifying the results

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

4.11
test execution engine
tool implemented in software and sometimes in hardware that can manipulate the test item to execute test cases

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

4.12
test framework
environment that facilitates testing

4.13
test interface
interface to the test item used to stimulate the test item, to get responses (e.g. actual results), or both

Note 1 to entry: The GUI, API or 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 under test or associated hardware.

4.14
test interface layer
lowest level of abstraction for keywords, which interacts with the test item directly and encapsulates the atomic (lowest level) interactions at the test interface

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 for various 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 needs to 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 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, a meaningful name is used to identify the keyword. This keyword may 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 in test cases, 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 may 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 may be limited in their capability to support all the concepts described in this standard.

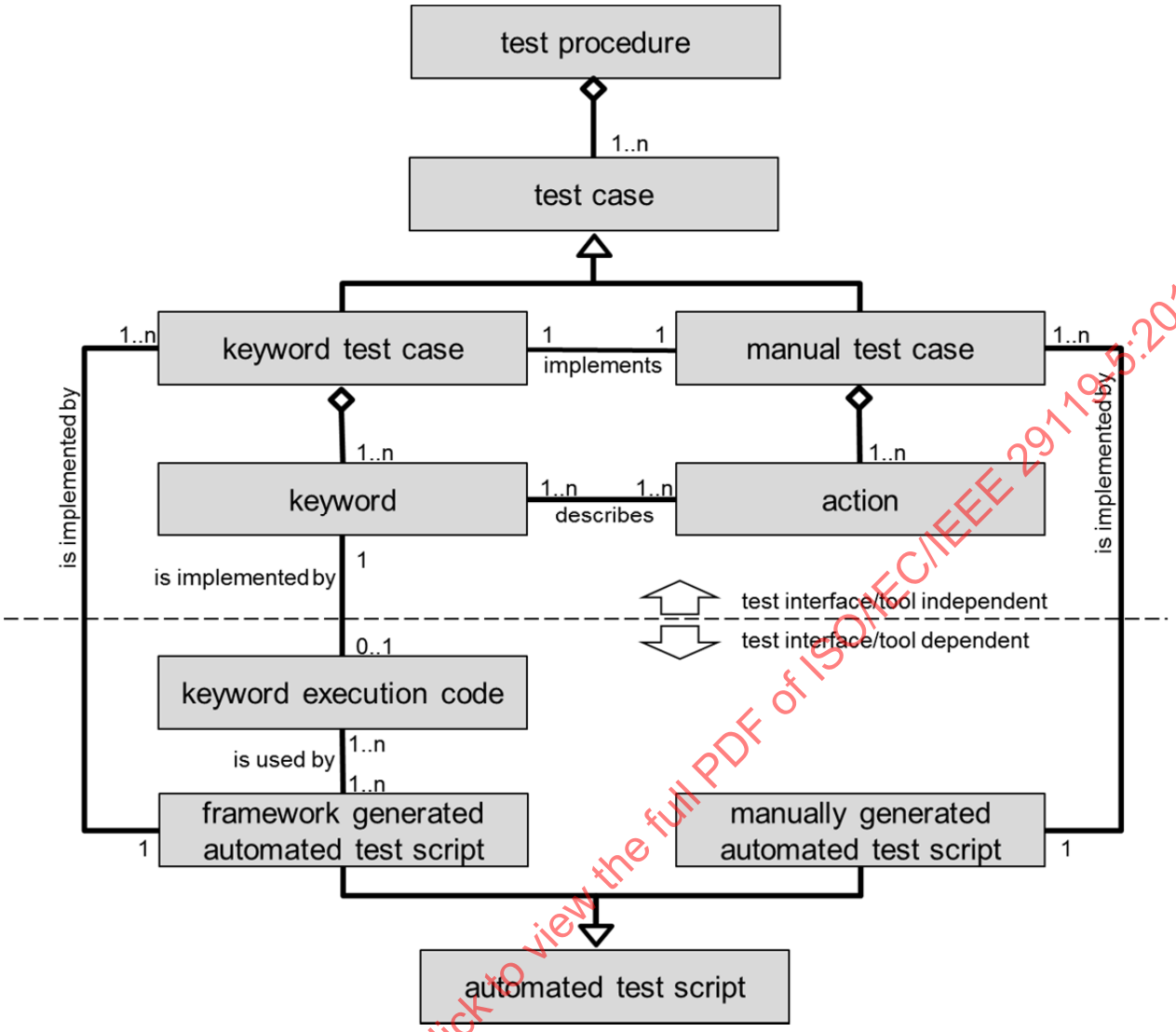

Figure 1 — Relationships between Keyword-Driven Testing entities

A test procedure can have multiple test cases in it, and a test case can, in turn, be part of different test procedures (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). 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 will additionally depend 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 may 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.

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 will comprise two or three abstraction layers, in some cases it may 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 may not be 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. "selectMenuItem"). 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 may be used).

Most Keyword-Driven Test systems will 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 will commonly be 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.

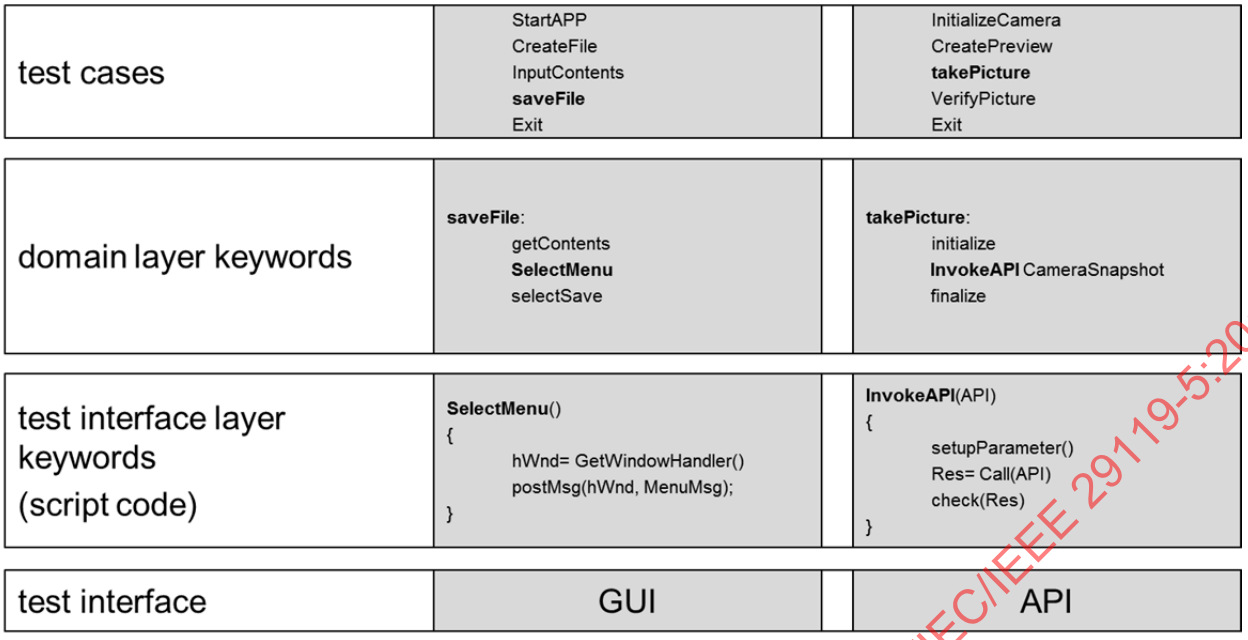

Figure 2 — Example for defining test cases by keywords at several layers

EXAMPLE In Figure 2, two test cases are shown which are designed using a domain layer and a test interface layer. One of the examples sketches a test case for a GUI application, the other for a camera API. In both examples, the implementation of the test cases in respect to test automation is done on the test interface layer. From top to bottom, the example shows a test case for each test item, shows how one of the used composite keywords on the domain layer for both test items could be structured, and gives an idea of how the implementation of one of the basic keywords on the test interface layer for each test item could look.

5.2.2 Domain layer

Keywords in the domain layer correspond to business or domain related activities and reflect the terminology used by domain experts. Because of this, it can be easier for testers at the domain or business level to create test cases.

Keywords developed for the domain layer are generally implementation-independent; that is, the keywords define tests that work regardless of the technology used to implement the test item.

EXAMPLE 1 Consider a keyword test developed to test a word processing application. Domain layer keywords correspond to the activities that are part of the “business of word processing”:

StartApplication <app_name>

ClearBuffer

EnterText “Hello World!”

ReplaceText “Hello”, “Goodbye”, “ALL_OCCURRENCES”

VerifyText “Goodbye World!”

StopApplication

This test is valid for any text editor application that provides a global replace function, (e.g. Notepad, MSWord, Notepad++, GED, EMACS, etc.).

EXAMPLE 2 Frequently used domain layer keywords are "Login" and "CreateAccount"

Tests constructed using domain layer keywords are relatively immune to changes in the implementation of the test item, and can prevent expensive rework over the lifetime of the test item.

NOTE Extensive changes to the application, (e.g. changes in the workflow) can require test cases to be reworked.

5.2.3 Test interface layer

Keywords at the test interface layer refer to a specific type of test interface, (e.g. the graphical user interface (GUI)). The actions needed to address the test items can usually be easily identified. The total number of keywords is typically smaller than at the domain layer, since the test interface is limited.

EXAMPLE 1 A GUI can be used as the test interface. As the GUI controls (along with the associated actions) are mapped to a fixed set of keywords, a small number of keywords is needed. In the same case the domain-related keywords can be very versatile, and may need to be extended according to the needs of the tester, which leads to a much bigger number of keywords.

If automation is desired, the keyword execution code for keywords at the test interface layer is often simpler. However, for a keyword test case composed from keywords at the test interface layer, it may be difficult to see how the interface layer keywords are related to the business domain.

Interface layer keywords usually reflect the underlying implementation technology for the interaction with the test item. For example, keywords such as MenuSelect and PressButton reflect a GUI operation. Using the example above, they would not be applicable to text editors using a command line interface, such as vi, because they correspond to window-based operations.

EXAMPLE 2 Consider a test interface that is a graphical user interface. Keywords are chosen to cover single actions such as "Click" or "Select". These keywords are applied to different elements like lists, grids, or images, and the specific element can be selected by using the keyword with a parameter (See 5.4 Keywords and Data). Some combinations of actions and elements can be excluded.

5.2.4 Multiple layers

To combine the advantages of several layers (e.g. domain layer and test interface layer), a framework is required, which can help manage hierarchical keywords (see section 7 for details about testing frameworks). This way a high-level keyword at the business level (e.g. domain layer keyword), can be built from several lower level keywords at a more technical level (e.g. test interface layer keywords).

Figure 3 illustrates how multiple layers can be used in Keyword-Driven Testing.

In complex settings, three or more layers of keywords are necessary. In the figure, additional intermediate layers are represented by three dots "…".

Using multiple layers requires composite keywords (see 5.3.2 Composite keywords).

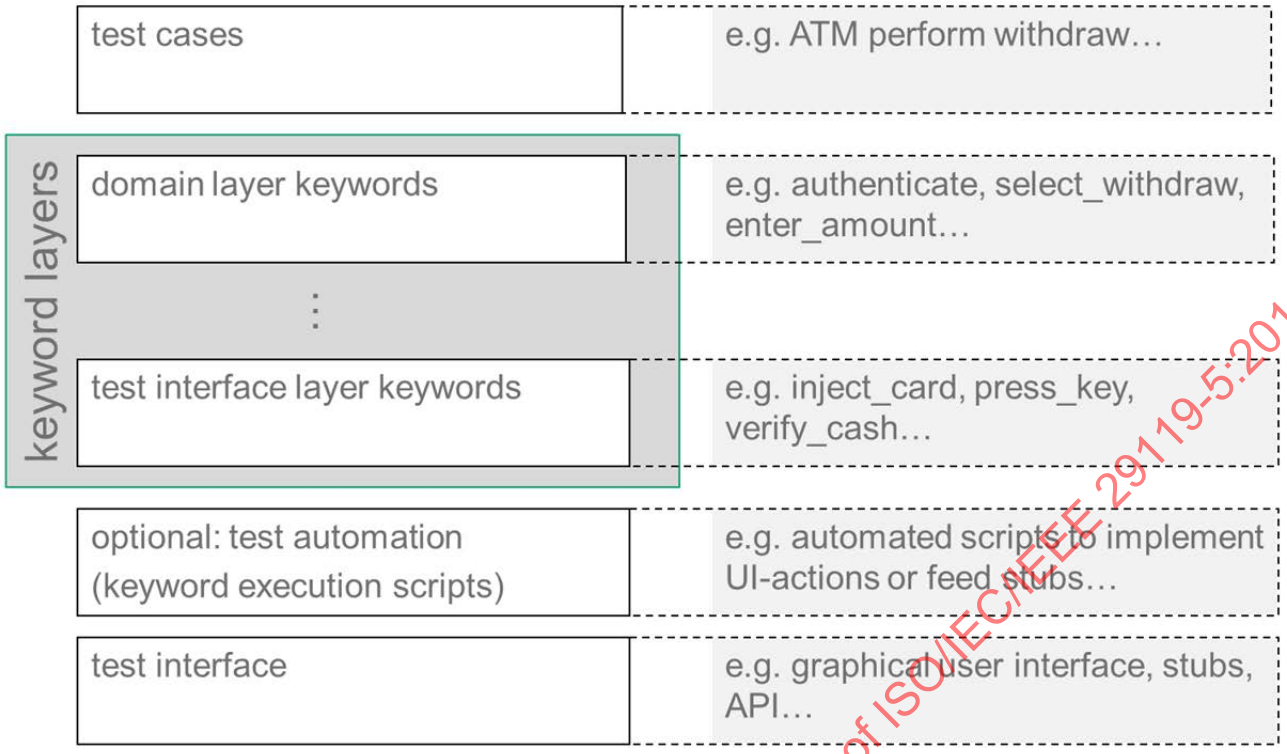

Figure 3 — Multiple layers in Keyword-Driven Testing

NOTE 1 Figure 3 explicitly shows two keyword layers – a domain layer and a test interface layer – and indicates that in between there may be intermediate layers. It is possible, and can be sufficient, to organize keywords in only one layer. However, there might also be situations in which more than two layers are needed.

NOTE 2 In Figure 3, the domain layer keywords are taken from the domain of an ATM test, and are meant to be used to create test cases. The keywords from the test interface layer refer to simple actions that can be applied to the test interface. The keyword "verify_cash" in this example is related to the test interface, and is supposed to cover only one small activity, and used as part of domain layer keywords. In another example it could be designed differently, cover several actions, and then be part of the domain layer.

5.3 Types of keywords

5.3.1 Simple keywords

Simple keywords, which are often used at the test interface layer (e.g. "MenuSelect" or "PressButton"), can be the connection between the test execution tool and higher level keywords at an intermediate layer or domain layer.

Using only keywords at the test interface layer can be sufficient for the definition of test cases and their execution. Exclusive use of simple keywords will lead to test cases with many actions.

Depending on the test item, keywords at the test interface layer could need to interact with different systems such as databases, the system registry or SOA-Messages. This challenge would normally be supported by the automation framework by providing a predefined set of keywords in order to make the technical environment as clear as possible.

In a similar way, the automation framework will support access to the test interface or other interfaces on which the keyword operates (e.g. mouse, keyboard, and touch screen).

Depending on the test interface, it can be possible to operate with a very limited number of simple keywords. A limited vocabulary of keywords is beneficial for composing test cases, since they are easier to remember, use and maintain. If test automation is required, a very limited number of keywords may result in an increased

effort to implement the keyword execution code. This is because if a small number of keywords still has to cover the same complexity, an individual keyword needs to be more flexible or powerful.

EXAMPLE An implementation is structured by the required actions such as "select". That particular action is only implemented once due to an objective to have a small number of keywords, In this case, the single keyword "select" needs to address several types of interface elements, such as for a GUI, lists, tables and radio buttons. The keyword "select" will for that reason be associated with a complex implementation.

5.3.2 Composite keywords

Simple keywords are sufficient to compose and execute test cases but are often insufficient to reflect functional features.

Composite keywords are keywords composed from other keywords. This means that keywords can be organized in different layers (see 5.2). For composite keywords, composite parameters (e.g. a data structure) can be required.

It is often useful to use business-level keywords, such as “login user“. This keyword may be composed of a sequence of lower level keywords, such as “enter username“, “enter password“ and "push login-button“. For more complex business objects, such as large forms for the preparation of contracts, a keyword like “filloutContractformPage1” can be valuable.

EXAMPLE 1 It is common to use composite keywords for verification, (e.g. retrieve a value from the application, compare it with an expected result, and log the result of the comparison in the test execution log).

It is also possible to define a keyword at a higher level (e.g. domain level) with a single keyword at a lower level (e.g. test interface level) to express a different semantic meaning.

EXAMPLE 2 For navigation purposes, a high-level keyword "GoToResultsScreen" is defined by the lower level keyword “Click ResultsButton”

It is also possible to combine several basic keywords to create a complex operation with a higher level of functionality, such as "CreateCustomerAccount", which may include a large number of basic steps.

A composite keyword is a ‘package’ containing a sequence of other keywords. The set of parameters for a composite keyword can be the union of the set of parameters of the keywords that comprise the composite keyword; sometimes however, the implementer of a composite keyword may choose to ‘hide’ one or more parameters by assigning it a literal value within the composite. This is done in the example in Figure 4. The interface layer keyword "Enter_value" has two parameters: the id of the referenced object and the value that is to be inserted. Only the value (e.g. username) is visible on the top layer keyword "login", while the id is hidden from a tester, who only used the composite keyword. This is especially useful if the detailed technical information is irrelevant to the person who designs test cases and operates at the domain layer.

EXAMPLE 3 Figure 4 illustrates how a keyword for a login procedure can be designed as a composite keyword in three layers. At the domain layer, this keyword can be used, (e.g. 'Login ("John","secret")'). This keyword is composed of three keywords at the intermediate layer, "Set_User", "Set_Pwd" and "Close_Login". "Set_User" and "Set_Pwd" both use one of the parameters of the higher layer keyword "Login", while the keyword "Close_Login" requires no parameters or data at all. At the third layer (the interface layer), the basic keywords "Set_context", "Enter_value" and "Select" are used. On this third layer, literal values are used, such as "Login_Window", which has not been provided with the domain layer keyword but will be used the same way every time one of the intermediate layer keywords is used.

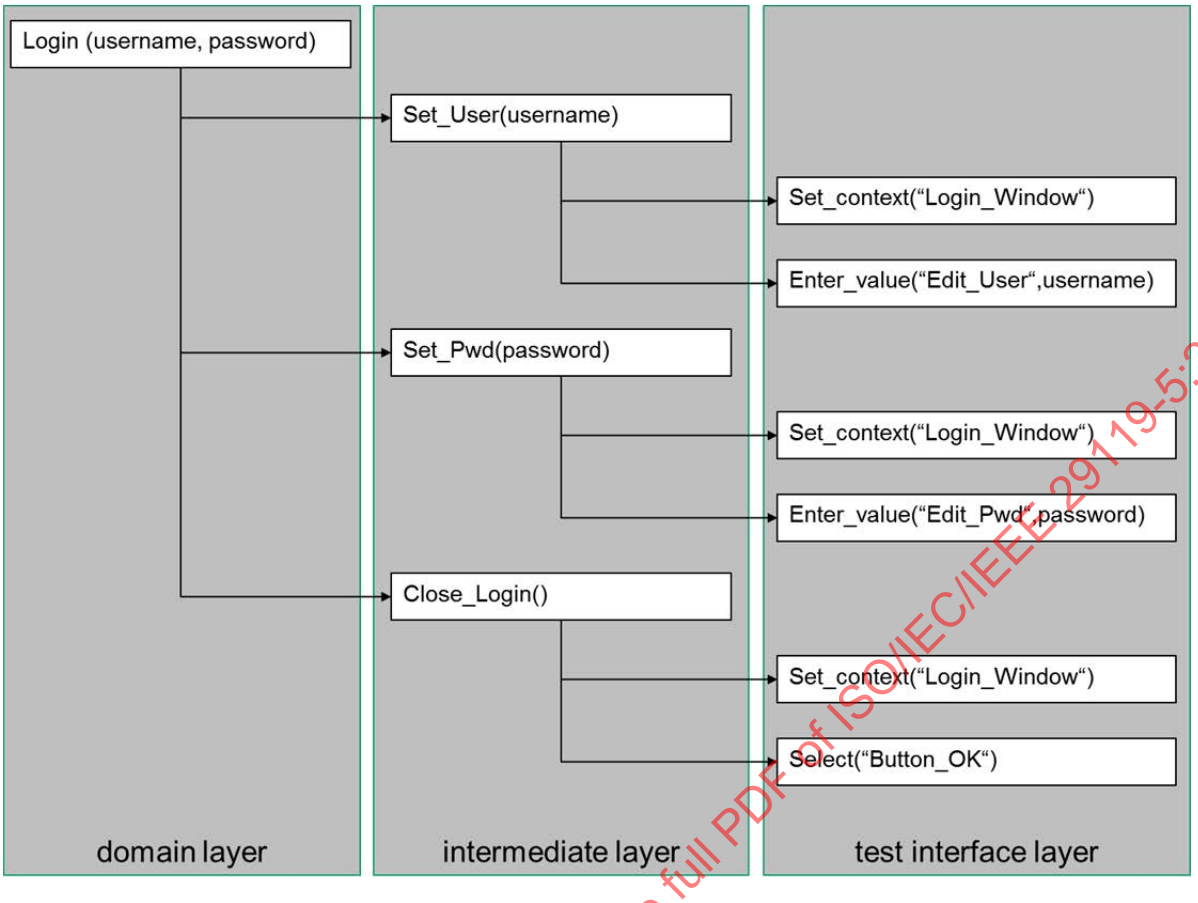

Figure 4 — Example for using composite keywords with data

The following figure explains the relationship between different types of keywords, keyword test cases and the level of keywords that are eventually applied to the test item. The keyword can be low-level, high-level or a composite keyword (Figure 5).

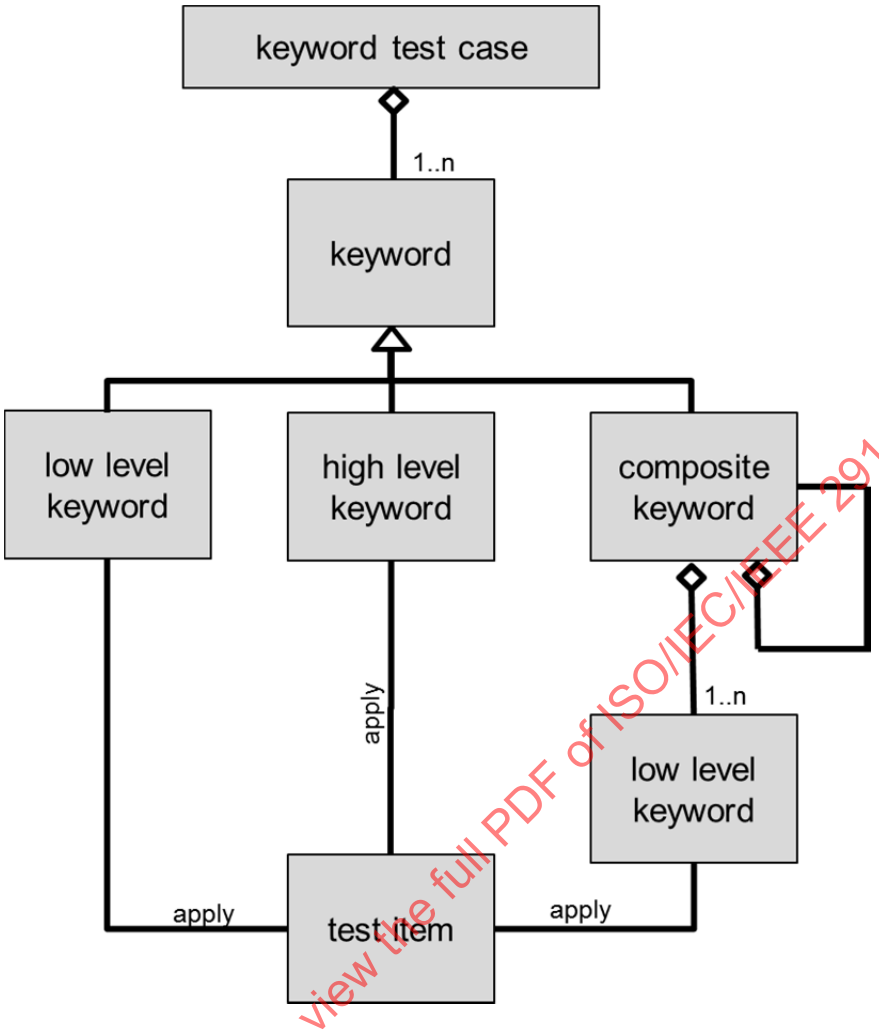

Figure 5 — Keyword test cases composed from keywords at different levels

If composite keywords are not used, keyword test cases can be built from low-level keywords, such as from the test interface layer. Through this approach, testing of the test item will be accomplished by using low-level keywords.

NOTE 1 In figure 5, the composite keyword can be either a low-level keyword, or a high-level keyword.

Consequently, the test cases will be understandable for human testers and machine readable test execution engines. On the other hand, when reading such test cases, it can be hard to recognize the use case or business case addressed by the test case.

By exclusively using high-level keywords, such as business keywords or domain keywords, the derived keyword test cases will generally be more understandable in respect to the addressed use cases or test cases. Testing of the test item will be accomplished by using high-level keywords. Thus human testers need more information about the detailed steps needed to execute the more abstract keywords, especially if they are not familiar with the business domain. If execution engines are used, these execution engines need more information about the detailed steps needed to execute the more abstract keywords as well.

After combining low-level keywords to form composite keywords at a higher level (e.g. combining keywords from the test interface layers with composite keywords at the domain layer) the keyword test case can be composed from these high-level keywords. Such test cases are very easy to understand, as they resemble the related use cases or business cases.

NOTE 2 Figure 5 shows, for composite keywords, two levels of keywords: composite high-level keywords, and low-level keywords. It is possible to have more levels, such as intermediate composite keywords, which are composed of lower level keywords and are used to compose higher level keywords.

To execute the tests, the high-level keywords can be decomposed into low-level keywords, usually using a framework (see 7.2). So testing of the test item will be accomplished by only using low-level keywords, which makes it easy for human testers or test execution engines to identify the necessary actions to perform the test since they will have simple steps to follow.

NOTE 3 Mixing keywords from different layers and using them in one keyword test case is possible, but not recommended, as it may be the source of maintenance problems.

5.3.3 Navigation/interaction (input) and verification (output)

Keywords may be classified into at least two categories: navigation steps (i.e. input to the test item) and verification steps (i.e. output from test item).

Most keywords belong to the first category, (i.e. the navigation steps) because most actions are needed to prepare the test item or perform certain actions on it which will lead to a result. Navigation steps usually are steps that do not verify and log the test result.

The result is then checked by one or more other actions i.e. the verification steps.

The verification steps are related to the result of the test case. For example, if the condition of a verification step is not met, then the test result will be set to "failed".

It may be useful to allow navigation steps to be used for verification.

EXAMPLE 1 A navigation step "AddUser" is required to prepare data for a test case. In some cases it may be used in a context where the addition of a user is supposed to succeed, in other cases it can be used in a situation where the addition is expected to fail. Thus, the keyword can verify whether it successfully creates a user, without marking the test case as "failed". However, the test designer can also decide to mark a test case as "failed" due to the failed execution of that navigation step, although the actual intent of the test case is to verify a result which appears later in the process.

See reasons for tests failing in subclause 5.3.4 Keywords and test results.

Keywords will typically be semantically independent from each other. Therefore, if a keyword is meant to trigger an expected result, the verification of this expected result will be part of the same keyword and not in another keyword.

EXAMPLE 2 Pairs of keywords like "Open the dialog" – "Verify the dialog is opened" are normally avoided when the second keyword is exclusively used following the first one.

5.3.4 Keywords and test result

Keywords can be used to determine the test status and to capture test results. This can include the following:

— Test output

— Conformance to success criteria

— Test execution log files

— Hardware outputs

— System status

— Test failure(s)

There are different reasons why a test execution can fail that can include the following:

— the conducted checks in the test case reveal a mismatch between actual outputs and expected outputs, which might indicate a software failure;

— some steps in the test case cannot be executed, because test execution is blocked.

NOTE Blocked test cases include test cases that cannot be executed due to faults in keywords, keyword execution code or the test environment.

It is useful to recognize the cause of a failed execution at first glance without having to analyse the cause in detail. Thus the framework will set different test results (e.g. failed and/or blocked) accordingly.

The result of an individual keyword execution will normally impact the test result, but that impact depends on the context.

EXAMPLE A keyword is defined to enter text into an edit field. The keyword works the same but the results are interpreted differently depending on context. If the text field is expected to be active and text entered successfully then the test result is set to passed. Conversely, if no text is entered to an active field, the test result is set to failed. On the other hand, if the text field is expected to be inactive and text entered successfully the result is set to failed, whereas if text cannot be entered the test result is set to passed.

The test framework can be designed to handle blocked keywords on the test item. Keywords can then be optionally marked either as “may be blocked” or as “must not be blocked”. In the first case, a blocking (unsuccessful execution) of the keyword would not affect the test result; in the second case, the test result will be affected. A keyword can be marked either globally (the property is default for all applications in test cases) or overridden when it is used in a test case.

The test framework can additionally provide an error recognition mechanism that can take care of errors returned by a keyword. Failures can be logged and described as clearly as possible in order to simplify the correction of errors in the automation framework and investigate its cause, which may be a software-defect.

5.4 Keywords and Data

Keyword-Driven Testing can be enhanced if keywords are associated with data. To allow an association with data, in many cases keywords will need to have parameters which may be fixed, or list driven.

Most keywords will need to have at least one parameter to specify the object they apply to. Some will need another parameter to specify input, (e.g., true/false, a string to type, an option to select in a combo box). This input will generally depend on the type of control and the type of action.

NOTE In the cases where a keyword represents a verification step, the required input for the keyword could be the expected output or a state for the referenced object.

Some keywords may also accept a number of optional inputs; in such cases, the framework needs to hold default values for those that are not provided (e.g. “Click UI_Element 456,123” may refer to a specific co-ordinate in the UI_Element, while “Click UI_Element” with no specified co-ordinate may default to clicking the center of that element).

For composite keywords, which can cover extensive functionality, the number of parameters can grow and the test data can become complex. It is a good practice to decouple the data from the actions. Therefore, multiple parameters can be stored separately and a unique reference to the data is used as input for the keyword.

EXAMPLE A composite keyword "createCustomer" requires data such as first name, surname and address of the customer. Instead of documenting the test data with the keyword test case, it is stored in a database. This allows a single reference to the complex data in the database, and the test case can be extended by providing several sets of data which are associated with the same sequence of actions.

Data-driven testing is a method of storing test data separately from the sequence of actions, which is independent of Keyword-Driven Testing, but is frequently used in conjunction with Keyword-Driven Testing. In data-driven testing, for one test case with a defined sequence of actions, multiple sets of data can be provided. The sequence of actions is then executed for each of the sets of data. Depending on the implementation, the data is either stored in a table, spreadsheet or database. Data-driven testing is an option to decouple the parameters from the test which matches very well with the concepts of Keyword-Driven Testing.

See subclause 6.4 Keywords and data-driven testing.

6 Application of Keyword-Driven Testing

6.1 Overview

This section addresses some concepts which contribute to a successful implementation of Keyword-Driven Testing. While all of these concepts are not required for each keyword test case, test design will benefit from them.

There are six concepts covered in this section.

a) "Identifying keywords" in 6.2;

b) "Composing test cases" in 6.3;

c) "Keywords and data-driven testing" in 6.4;

d) "Modularity and refactoring" in 6.5;

e) "Keyword-Driven Testing in the Test Design Process" in 6.6; and

f) "Converting non keyword-driven test cases into Keyword-Driven Testing" in 6.7.

6.2 Identifying keywords

Identifying Keywords is a pivotal task in Keyword-Driven Testing as the contents, granularity and structure of the keywords can impact the way keyword test cases are defined. It is important to name keywords in a way that appears natural to the people who will be working with them.

When identifying keywords, the following steps are executed:

a) determine the layers needed in the given context and define what sort of keywords (e.g. functionality, granularity) are supposed to be assigned to the layers;

b) identify keywords in the layer based on the definition or scope of each layer.

Generally, keywords are defined by first identifying sets of actions that are expected to occur frequently in the testing. A name (e.g. the keyword) is applied to an action or group of actions. Keywords are applicable in a range of situations. At this point it is useful to determine which of the actions are information-dependent (e.g., time, data, situation, etc.), and so identify which keywords need to be associated with parameters.

A keyword is described by the following information:

— The name of the keyword. It tells the reader what this keyword is expected to do.

— The parameters of the keyword, which can be empty.

— Documentation on the keyword, including the layer in which this keyword is expected to be used, the keyword type (e.g. navigation or verification), the context in which it is to be used, the actions included

with the keyword, either as a description, or as a reference to keywords on a lower layer (see next bullet), and the objectives of the keyword.

— If the keyword is composed from other keywords, a list of the included keywords in the order in which they are used.

Basic Keywords can be identified by observing different interactions available at the test input interface, such as interactions with keyboard, mouse, touchscreen, microphone, API, etc.

Composite keywords can be identified by observing common actions that the user will perform at the UI level.

EXAMPLE "GoTo" would be used instead of "ClickButton", or "Select" instead of "ClickRowInTable".

Typical business behaviour can be encapsulated in a composite keyword (e.g. "CreateNewUser"). Other complex manipulations like interactions with databases can also be candidate keywords in the framework.

The following issues should be considered:

— Uniqueness: each keyword should be unique in its context of use.

— Reusability: the keywords should be defined in a way that best supports future reusability.

— Completeness: keywords should be defined with a view to all known elements and possible interactions of the test interface (e.g. all known objects in the GUI and its dialogs).

— Clarity: all keywords should be defined with a clear and consistent structure.

NOTE 1 All keywords in a layer should have a similar abstraction level.

— Specificity: keywords should not be redundant and should be mutually exclusive (i.e. keywords will represent distinct actions), to ease the test design and to decrease the maintenance effort.

NOTE 2 In some environments it can be useful to use an object-oriented approach to identify and describe keywords. Keywords can, in that approach, be identified by analysing the available objects and methods on the objects in the domain. Keywords can then be described in a style like "OBJ.Action Parameter", where "OBJ" refers to the object which is to be addressed (e.g. a button in a user dialogue box), "Action" refers to the activity (e.g. "press" for a button), and "parameter" refers to a list of additionally-needed parameters. This approach can be useful if all stakeholders performing Keyword-Driven Testing within that environment are familiar with object-orientation.

6.3 Composing test cases

Keyword test cases can be composed from previously-defined keywords. In the process of writing test cases, it can occur that missing keywords are discovered and can, therefore, be defined after that point.

NOTE Keyword test cases can be composed from composite keywords and used to build end-to-end tests.

Within the test case specification (see ISO/IEC/IEEE 29119-3), test cases can be documented using appropriate notation, including the use of tables or databases. The format depends on the available infrastructure (e.g. availability of a test management tool and the plans for automated execution).

Keyword test cases usually contain keywords from a single layer. A clear distinction is made between the layers. This distinction opens the option to distribute the design of different layers to different testers (see 7).

EXAMPLE 1 Test of an ATM using only basic keywords at the test interface layer:

enterValue("Card", 123000789)

enterValue(“PIN”, 1234)

selectObject("Button", "OK")

selectObject(“Button”, “Payment”)

enterValue(“Amount”, “200”)

selectObject(“Button”, “executePayment”)

verifyObject(“Payment”, ”200”)

EXAMPLE 2 Test of an ATM using domain layer keywords:

signInUser(123000789, 1234)

executePayment(“200”)

verifyPayment(“200”)

More examples can be found in Annex F.

6.4 Keywords and data-driven testing

Keywords combined with parameters and separate data sets for these parameters (e.g. data-driven) may offer improved testing. Data-driven testing can be applied when the same sequence of keyword actions are to be used with different sets of data. In this combined approach, the data can be stored separately from the keyword test cases. The data can be stored in constructs such as tables, databases, or real-time generators, and is then read into the keyword test cases. The repeated keyword sequence using different data, in effect, creates new tests.

EXAMPLE Data-driven testing is useful for multi-lingual testing or internationalization testing, where the same test cases are to be performed at user interfaces but with different languages. It should be kept in mind that not all internationalization issues are easily addressed by data-driven testing: in the case of lexicographical sorting, the requested item may have not only have a different label but also a different position.

The following guidelines are taken into consideration for data-driven testing with keywords:

— A keyword does not have to be “loop aware”. In other words, a keyword will ideally work the same whether it is part of a linear sequence or is contained within a loop (such as in a data-driven test). This places the burden of managing the data file and fetching its content on the framework, not on the keyword. It implies that the only method of getting data into and out of a keyword is through its parameters.

— Multiple, non-nested loops in a test case, can be implemented, but are discouraged. Good data-driven test design suggests the use of a single loop in test cases that are data-driven.

— Nested data-driven loops are discouraged. Nesting data-driven loops by more than two is normally avoided.

— The format of the data file and its contents are implementation defined. This standard does not dictate the format of the file (e.g. an implementation can support data from Excel files, text files, or any other file type). Neither does this standard dictate the format of the data items within the file (e.g. XML, ASCII or Unicode text, binary encoding, or any other format is permitted).

NOTE Although Keyword-Driven Testing and data-driven testing are concepts that can be used independently in theory, in practice Keyword-Driven Testing includes data-driven testing.

6.5 Modularity and refactoring

Modularity in Keyword-Driven Testing is used to improve the longevity of the test cases. However, with the passage of time, changes in the test item, new test cases or new people on the team can all lead to maintenance issues.

Possible issues are as follows:

— Redundant keywords: where two or more keywords for the same objective come into existence.

— Unused keywords.

— Conflicts where changes in keywords (e.g. structure or semantic), which fix an issue in a number of test cases, create new issues in other test cases. This has associated cost factors.

— Uncoordinated changes in keywords (e.g. name, semantics, parameters) cause rework or invalidate test cases of other testers.

To avoid these issues, the following maintenance actions should be considered:

— A framework for Keyword-Driven Testing should provide a way of creating a cross-reference for the used keywords, allowing identification of which keywords are used in which places and how frequently they are used. This shows if, and how much, a change in a keyword will affect existing test cases.

— In some organizations, an authority is required who is responsible for all keywords, additions and any changes to existing keywords or how they are used. That authority assures consistency throughout the project including both development and testing stages.

— On a regular basis (e.g. once a month), a keyword review meeting can be held. At this meeting, testers can decide about the introduction or modification of keywords and discuss the structure of the keywords. If an authority is in charge of keywords, they should also be in attendance.

— A clear structure to document the keywords should be produced. Keywords can be grouped by layer, test item and region in the test item (e.g. dialog, objective or others). Keywords that are supposed to be usable by all testers will normally be stored separately from keywords which are only useful for a limited number of people.

— Keywords can be subjected to configuration management practices (e.g. the authority mentioned above). The ability to change keywords would normally be limited to those who need to do changes or the authority. All changes need to be well documented. Access to prior versions (e.g. the option for rolling back) should also be provided.

6.6 Keyword-Driven Testing in the Test Design Process

6.6.1 Overview

The Test Design and Implementation Process defined in ISO/IEC/IEEE 29119-2 (see figure 6) is applicable to this standard. This clause describes the relationship between the activities of this process and Keyword-Driven Testing.

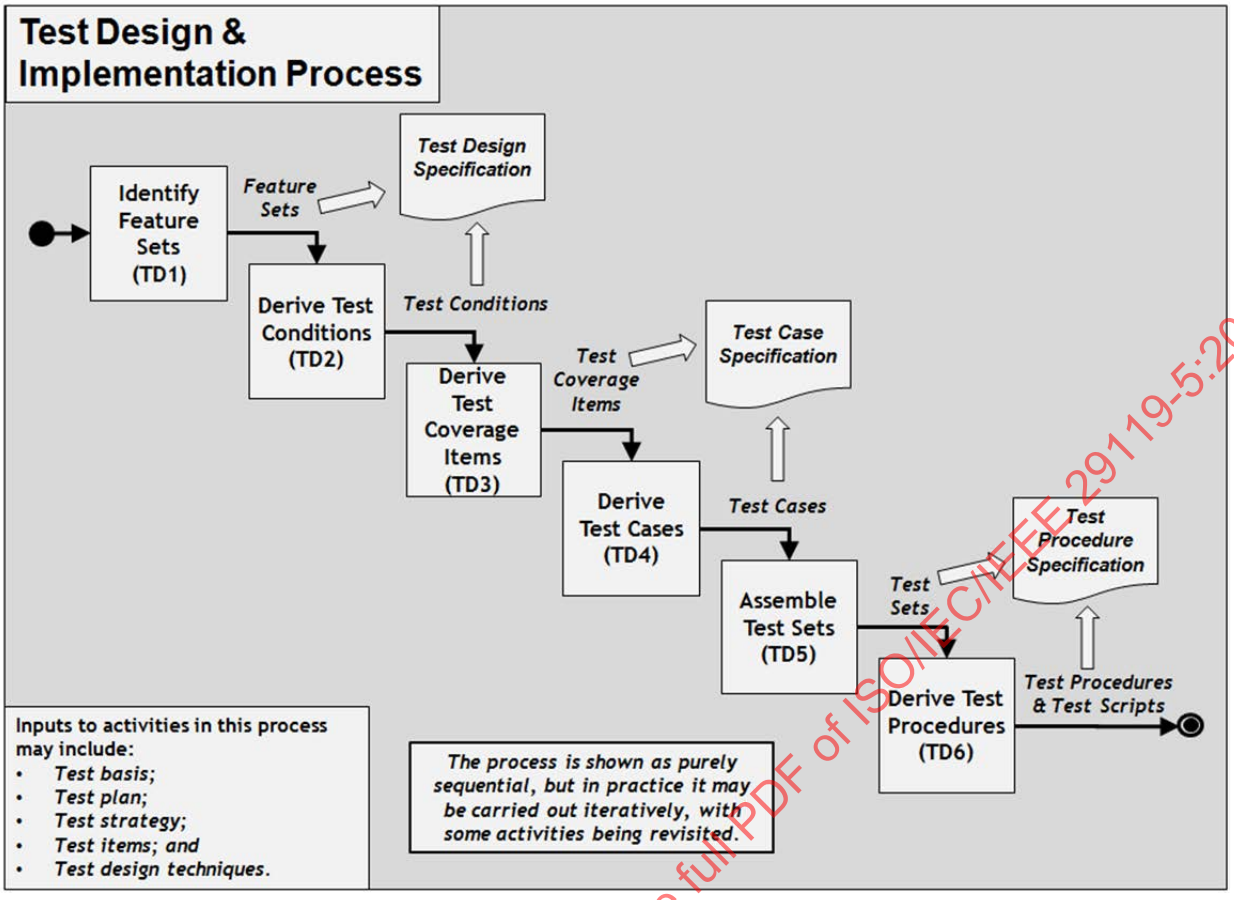

Figure 6 — ISO/IEC/IEEE 29119-2 Test Design & Implementation Process

The Test Design and Implementation Process in subclause 8.2.1 ISO/IEC/IEEE 29119-2 (figure 6) describes six steps from 'Identify Feature Sets' (TD1) to 'Derive Test Procedures' (TD6).

The requirements of TD4 to TD6 in ISO/IEC/IEEE 29119-2 are most applicable to the Test Design & Implementation Process used in Keyword-Driven Testing.

TD1 to TD3 are not addressed in ISO/IEC/IEEE 29119-5. Keyword-Driven Testing is relevant when performing the activities TD4 – 'Derive Test Cases', TD5 – 'Assemble Test Sets' and TD6 – 'Derive Test Procedures'. These will be covered in the following subclauses.

6.6.2 TD1 Identify Feature Sets

TD1 can be applied in Keyword-Driven Testing as defined in ISO/IEC/IEEE 29119-2 and is not covered here.

6.6.3 TD2 Derive Test Conditions

TD2 can be applied in Keyword-Driven Testing as defined in ISO/IEC/IEEE 29119-2 and is not covered here.

6.6.4 TD3 Derive Test Coverage Items

TD3 can be applied in Keyword-Driven Testing as defined in ISO/IEC/IEEE 29119-2 and is not covered here.

6.6.5 TD4 Derive Test Cases

6.6.5.1 Overview

In TD4, Keyword-Driven Testing is focused on composing keyword test cases of simple keywords or composite keywords.

A keyword test case is implemented using keywords. Keywords that support the needed test cases will have been defined prior to TD4. It is possible that new keywords may be identified during the performance of TD4 tasks and in this case iterations of TD4 are likely to be required.

According to ISO/IEC/IEEE 29119-2, test cases are derived by the following steps (TD4 task a):

— Determine pre-conditions

— Select input values

— Select actions (to exercise test coverage items)

— Determine expected results

These design activities identify the different actions that need to be performed to prepare the system for the test, execute the test and verify the test results. Keywords may fulfil one or more of these actions.

Examples of keyword-driven test cases can be found in Annex F.

6.6.5.2 Determine pre-conditions

The tester determines and establishes the needed test pre-conditions and identifies which can be achieved using keywords or other actions. Composite high-level keywords (see 5.3.2) can be appropriate in a situation where multiple actions are required. Additionally in some testing, keywords can be used to prepare system conditions before establishing specific test case pre-conditions (e.g. importing data into a database or setting parameters for application start which can be used over a series of test cases).

6.6.5.3 Select input values

The tester selects input values based on test design considerations and then implements these in keywords. In situations where test cases are supposed to be executed according to the same actions, but with different sets of data to provide different test outcomes, the keywords will normally be developed to support data-driven testing (see 5.6).

6.6.5.4 Select actions

The tester identifies those actions required to exercise the test coverage items. If the required actions are not available from existing keywords, new keywords may be needed or existing keywords can be combined into composite keywords, as appropriate, to provide the needed functionality. Such changes may require iteration on this effort.

NOTE As keywords will sometimes already be defined, the iterative nature of the Test Design and Implementation Process suggests that keywords can be subject to refactoring.

6.6.5.5 Determine expected results

The tester will determine expected results and implement checks or feedback on results using keywords. Keywords can be used to check the results the test item returns and log them accordingly (see 5.3.3 and 5.3.4).

6.6.6 TD5 Assemble Test Sets

Test sets can be formed from keyword test cases (TD5 tasks a and b).

A framework for Keyword-Driven Testing can provide mechanisms for assembling test sets from various test cases (see 29119-2) by applying different criteria (e.g. the same test environment set-up, or keywords with data-driven testing).

6.6.7 TD6 Derive Test Procedures

The developed keyword test cases and test sets become the primary input for deriving test procedures as the keywords are easily read and understood (TD6 tasks a and d).

Additionally, a keyword framework can provide mechanisms to determine the execution order within test sets and across test sets, thus generating the basic structure of a test procedure. The framework can also be designed to ensure that required pre-conditions are set up before test execution. This may require additional generic keywords.

6.7 Converting non keyword-driven test cases into Keyword-Driven Testing

If Keyword-Driven Testing is to be introduced into an existing project, existing test cases can be converted into keyword test cases.

In addition to the advantages of Keyword-Driven Testing, reasons for deciding to convert existing test cases into Keyword-Driven test cases include the following:

a) Uniformity: ensuring that all test cases have a similar structure and style will enhance readability, maintainability and so reduce costs. Future maintenance may be cheaper if only one style of test case is to be maintained.

b) Efficiency: keywords identified from existing test cases may be reusable in future test cases.

c) Automation: the same automation framework may be used for old and new test cases.

d) Understandability: test cases may be used and maintained by non-technical testers (often with business knowledge).

Reasons to keep existing test cases and not convert them into Keyword-Driven test cases include the following:

— The number of existing test cases is large compared with the number of additional test cases that are needed.

— There is proven and maintainable automation for the existing test cases.

— The cost of converting the test cases is expected to exceed the benefits.

The decision to move to Keyword-Driven Testing for existing projects should be carefully considered.

7 Keyword-Driven Testing Frameworks

7.1 Overview

If Keyword-Driven Testing is to be applied, everything necessary to do that needs to be organized in a Keyword-Driven Testing framework. This framework consists of concepts, documents and tools. The framework can have more or less complexity depending on its purpose.

The components of a framework are described in 7.2.

This subclause addresses several points to consider when implementing or using frameworks for Keyword-Driven Testing.

The following subclauses describe attributes for Keyword-Driven Testing frameworks:

— Basic attributes that are necessary to implement Keyword-Driven Testing (see 7.3).

— Advanced attributes providing additional value and are desirable, but are not expected to be available in all frameworks (see 7.4).

The attributes are divided into general, test design tool, and test execution engine aspects as follows:

— General aspects are mostly tool-independent.

— Test design tool aspects are related to a software tool component of the framework that is used to manage keywords, compose test cases from keywords, and assign data.

— Test execution engine aspects are related to a software tool component of the framework which is used to execute the test cases as automated tests.

A framework will likely be composed of a series of tools each providing parts of the needed capabilities. The framework as a whole needs to meet the named requirements.

7.2 Components of a Keyword-Driven Testing framework

7.2.1 Overview

Keyword-Driven Testing is typically supported by a framework. The framework can be realized in a variety of ways including by commercial tools, custom tools, and solutions in the form of script libraries or other supporting elements.

A Keyword-Driven Testing framework will comprise functional units (or functional areas) which are shown in figure 7. This standard does not describe how these functional units are to be implemented. In practice, a commercial or custom software tool can cover these functional units in parts or completely. One or more software tools, along with custom implementations, libraries and organizational processes can form a Keyword-Driven Testing framework. Tools may have a different overall organisation. A framework needs to cover the requirements described in subclauses 7.3 to be compliant with this International Standard (see 2).

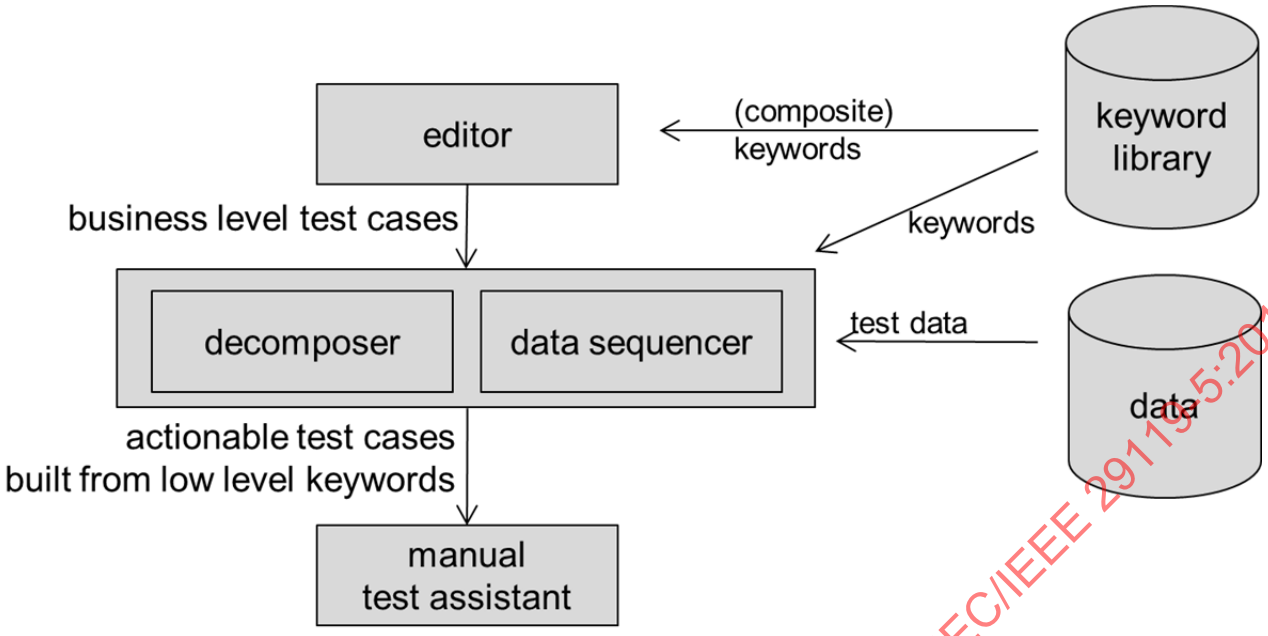

Figure 7 — Components of a Keyword-Driven Testing framework for manual test execution

In Figure 7, a Keyword-Driven Test framework is shown that is restricted to the support of manual testing. If test automation is required, the framework would be comprised of additional elements, as shown in figure 8. A manual test assistant is not required. Instead, an execution engine is used to run the test cases against the subject under test (SUT). A tool bridge is used as a link between the keywords and their representation in the automated test execution environment. The tool bridge's main task is to transform the necessary information into a suitable format for the execution engine.

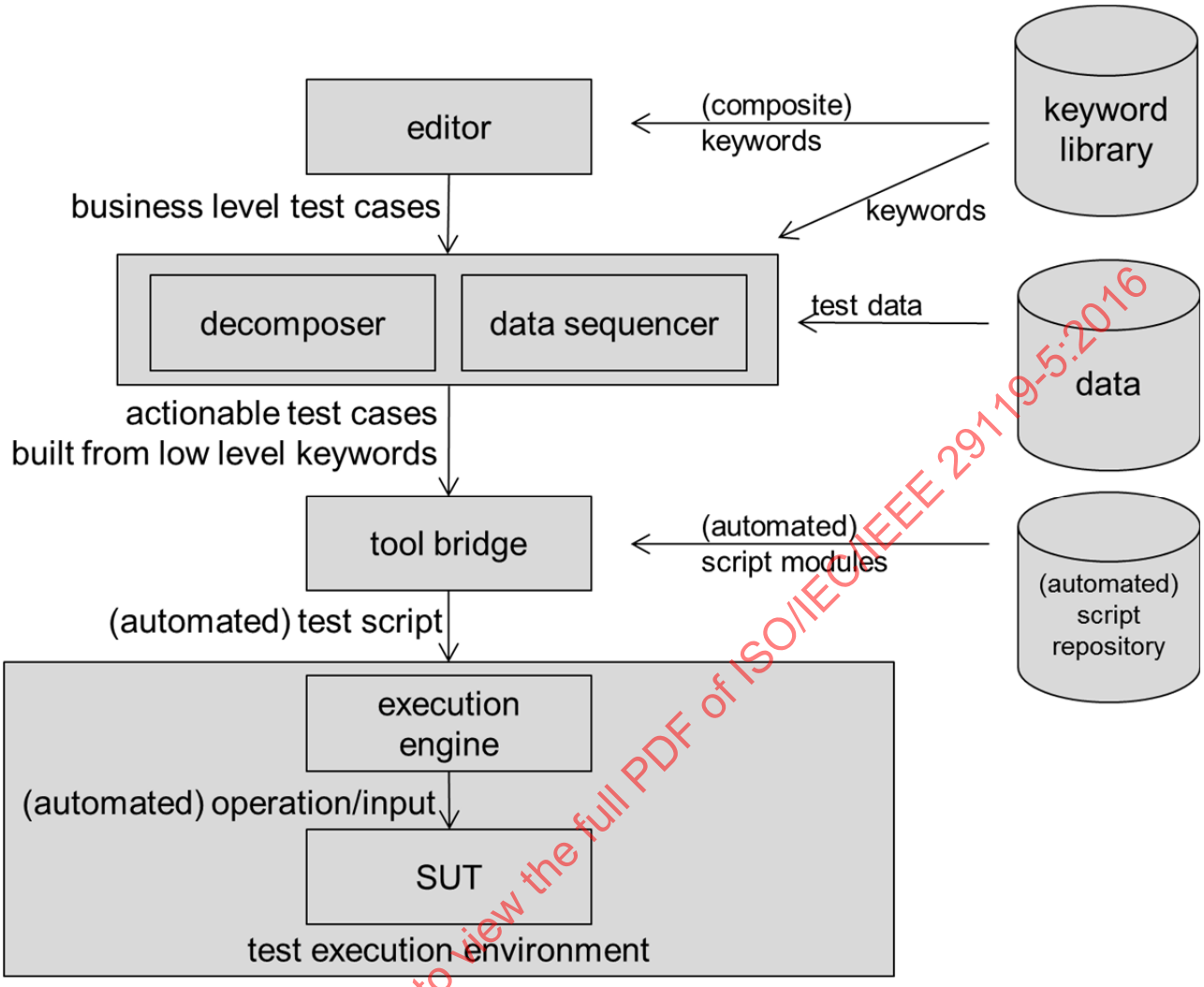

Figure 8 — Components of a Keyword-Driven Testing framework for automated test execution

The functional components which form a Keyword-Driven Testing framework are explained in the following subclauses.

NOTE These component descriptions do not assume any specific implementation of the components. In practice, one specific tool can cover some of these components (e.g. Keyword-driven Editor, Decomposer, Data Sequencer and Manual Test Assistant might be included seamlessly in one test management tool). Other implementations may provide only parts of one of the components in one tool.

7.2.2 Keyword-driven Editor

The Keyword-driven Editor is required to compose keyword test cases from keywords. The keywords can be taken from a keyword library (7.2.8).

In practice, the Keyword-driven Editor can be implemented in various ways.

EXAMPLE Possible implementations of a Keyword-driven Editor include, but are not limited to, a spreadsheet application, a dedicated standalone application or can be part of a test management tool.

7.2.3 Decomposer

The decomposer is required if composite keywords are used. The main task of the decomposer is to transform the keyword test case, which consists of a sequence of high-level keywords, into the appropriate sequence of low-level keywords.

7.2.4 Data sequencer

The data sequencer is required if Keyword-Driven Testing is to be applied with several sets of data associated with one keyword test case. The main task of the data sequencer is to transform the sequence of keywords (e.g. low-level or high-level) which are not yet associated with data to a list of keywords with specific data.

By doing this, the original list of actions, which has been written only once in the Keyword-driven Editor, will be repeated for any desired set of data. All parameters or placeholders are replaced by the final value needed in the respective test case.

The data sequencer can work both on high- and low-level keywords.

NOTE Depending on the implementation, the tasks of the decomposer and the data sequencer can be performed in arbitrary order. Both tasks can be done by the same software implementation.

7.2.5 Manual test assistant

The manual test assistant is only required for manual test execution. Its task is to present the test cases as prepared by the decomposer and data sequencer in an actionable way to the human tester. The tester then performs every single action, as well as documenting the test execution and the results.

In practice, the manual test assistant is frequently part of a test management tool.

7.2.6 Tool bridge

The tool bridge is only required for automated test execution and has a similar function to the manual test assistant used to support manual test execution.

The task of the tool bridge is to provide a connection between the keywords, as they appear in the keyword test case or in the keyword library, and the associated implementation in the test execution environment.

For each keyword passed from the data sequencer or from the decomposer, the tool bridge will, depending on the implementation, request the test execution engine to call the proper script (e.g. keyword execution code), functions, and perform the right actions with the appropriate data, if applicable.

In practice, a tool bridge can be implemented as a separate software tool, as a script in a test automation tool's runtime environment, or as part of a test management tool. Some bridge implementations may be referred to as a "generator," for example when a script or parts of scripts are generated for execution by an automation tool. Additionally, a bridge may be called an "interpreter" or "engine", for example, when a script is executed to interpret the sequence of keywords and calls the corresponding sub-functions.

7.2.7 Test execution environment and execution engine

To support automated Keyword-Driven Testing, the test environment will contain an execution engine with links to the item under test. The execution engine is a tool implemented either by software, hardware or both. Its task is to execute the test cases by performing the actions associated with the keywords.

In practice, the implementation of a test execution engine varies depending on the test object and environment. The execution engine can be a commercial test execution tool, (e.g. a capture and playback tool, or it can also be a hardware appliance, controlled by software, such as a robot).

7.2.8 Keyword library

The keyword library stores keyword definitions for one or more projects or portions of those projects. It is used to store the core information on keywords, such as: name, description, parameters and, in the case of composite keywords, the list of keywords from which the respective keyword is composed or derived. For test automation, it also contains necessary information for the tool bridge to associate the Keywords with the keyword execution code. The keyword library can help the tester to find a keyword.

In practice, a keyword library is supported by a test management tool.

7.2.9 Data

The data element in the Keyword-Driven Testing framework refers to the test data used for the keyword test cases.

Keyword test cases can be designed so the test data is included in the test cases. In this case, external test data is not be required. In other implementations, the keyword test case does not contain actual data, but contains placeholders which need to be substituted with data before the test case can be executed. In this case, the test data needs to be stored. In practice, it is common to store that data in files, in a spreadsheet application, in a dedicated database, or in a test management tool.

7.2.10 Script repository

The Script repository stores keyword execution code. It is only required if Keyword-Driven Testing is done with the aim of executing the test cases automatically.

For automation of Keyword-Driven Testing, each keyword needs to be associated with at least one command, test script or function, which implements the actions associated with that keyword. The script repository stores the technical implementations of the keywords.

In practice, the script repository is frequently implemented by either a test automation tool or stored at a defined location in the file system.

7.3 Basic attributes of the Keyword-Driven Testing framework

7.3.1 General information on basic attributes

This clause defines framework attributes that are generally necessary for the application of Keyword-Driven Testing. It describes attributes which are required in Keyword-Driven Testing frameworks and are necessary for compliance with this International Standard.

The following subclauses structure these attributes and requirements by the components of Keyword-Driven Testing frameworks according to subclause 7.2.

NOTE Requirements concerning data interchange format are not discussed in the following subclauses, instead see clause 8.

7.3.2 General attributes

General attributes that apply to Keyword-Driven Testing frameworks include the following:

a) There shall be documentation recorded describing each keyword.

NOTE 1 This is necessary for people to understand and use the defined keywords appropriately to build their test cases. The description is improved by the inclusion of an example.

b) There shall be documentation recorded for the parameters of each keyword.

NOTE 2 Keyword and parameter documentation includes naming of the keywords and how they are described, the parameters' maximum length, allowed characters, optionally reserved names or characters, and documentation rules.

c) A default value shall be documented for every parameter in case a value for a parameter is missed in the keyword test case definition.

d) There shall be high-level documentation recorded describing the hierarchy of the keywords that can be used.

e) There shall be high-level documentation recorded describing how data is stored and referenced for data-driven tests.

EXAMPLE Data could be stored in a database or in a spreadsheet, and could be organized in columns or rows.

The documentation described above can be part of the Test Plan, Test Policy, Organizational Test Strategy (see ISO/IEC/IEEE 29119-3) or a standalone document with references to/from other test documents.

There are several options of recording this information, such as word processors or test management tools.

7.3.3 Dedicated keyword-driven editor (tool)

When creating keyword test cases, it is recommended that a tool be used which supports the building of test cases.

Requirements that apply to the dedicated Keyword-driven editor include the following:

a) Within the keyword-driven editor, non-composite keywords shall be displayed with their associated actions.

EXAMPLE 1 A keyword "login" could be associated with the actions "press button >>login<<", "enter user name", "enter password" and "press button >>submit<<".

NOTE 1 A keyword like "login" can be designed to be composite or non-composite. In this example it is assumed that the tester has decided to define "login" as a non-composite keyword.

b) For keywords which have been defined with lower level keywords, the user shall be able to access this definition within the keyword-driven editor.

c) The keyword-driven editor shall allow the use of keywords with parameters to support data-driven testing.

d) The keyword-driven editor shall provide the capability to enter comments.

e) The keyword-driven editor shall offer the capability to connect to data sources that are to be used to assign values to parameters.

NOTE 2 Through this capability test cases become keyword-driven and data-driven. While it is possible to use Keyword-Driven Testing without data-driven testing, in practice data-driven testing is so important for efficient Keyword-Driven Testing that frameworks for Keyword-Driven Testing are expected to offer the option of data-driven testing.

EXAMPLE 2 A data source could be a database or a spreadsheet.

f) Within the keyword-driven editor, multiple uses of keywords shall be implemented by reference.

NOTE 3 Copying implementations of keywords can be avoided by using references.

g) The keyword-driven editor shall provide the capability to define the order in which the test cases are to be executed.

NOTE 4 The test execution order is part of deriving test procedures.

7.3.4 Decomposer and data sequencer

Requirements that apply to the decomposer and data sequencer include the following:

a) The decomposer shall be able to process parameters, including assuring that the parameters associated with the higher level keywords are decomposed and associated with the lower-level keywords.

b) The data sequencer shall be able to process parameters.

7.3.5 Manual test assistant (tool)

Requirements that apply to the manual test assistant include the following:

a) The manual test assistant shall support manual test execution based on the defined test cases.

b) The manual test assistant shall provide support for tracking any defect associated with a test failure.

7.3.6 Tool bridge

Requirements that apply to a tool bridge include the following:

a) The tool bridge shall provide the test execution engine with the appropriate execution code to execute the test cases.

7.3.7 Test execution engine

Test execution engines are designed to execute test cases by addressing one or more test interfaces (e.g. an API, a GUI or a hardware interface). A test execution engine can be implemented by software, by hardware or both. A common example of this class of tools is "JUnit".

Requirements that apply to the test execution engine include the following:

a) Keywords that do not express conditions or loops within a test case shall be executed sequentially starting with the first keyword.

NOTE 1 This is in general; but exception handling can require non-sequential execution to process an abort.

b) The execution engine shall be able to identify unimplemented keywords.

NOTE 2 A keyword is unimplemented if there is no execution code for that keyword.

c) The execution engine shall provide support for both literal values and variables in parameters.

NOTE 3 Variable definition can be implemented by configuration files, or by other means.

d) The execution engine shall provide execution results at the keyword level for each execution of each keyword implementation.

NOTE 4 By that, a user will be able to tell from the test results whether a keyword was executed successfully, or, if execution of the keyword failed, why (e.g. text field not writeable, field not present, etc.).

e) The test execution engine shall be able to store the timestamp of its executions with the duration of its execution.

f) The execution engine shall provide an error recognition mechanism as described in subclause 5.3.4.

NOTE 5 As a consequence, the execution engine either limits the number of cycles in a loop or provides another means to make sure that unlimited loops are impossible (e.g. by terminating each loop after a predefined time).

g) The execution engine shall provide a clear definition of PASS/FAIL for a test case whenever there are passed and failed executions in one loop.

NOTE 6 If a loop contains a verification, it could happen that the verification fails for some, but not all loop cycles. The PASS/FAIL definition indicates if this situation will be either "PASS" or "FAIL" for the test case.

h) The execution engine shall include the unique identifier of the execution in the execution logs.

i) The execution engine shall include the unique identifier of the test environment in the execution logs.

j) The execution engine shall include the unique identifier of the test item in the execution logs.

k) The execution engine shall support multi-application keywords by providing a mechanism to select between multiple implementations of a keyword.

NOTE 7 This allows a test case to manipulate more than one application using keywords written for each application. For example, a test case that verifies interoperability of an office application suite should be able to use keywords written for each of the two applications in a single test case.

l) The test results shall be available to the user.

NOTE 8 Other components include test design or test management components.

7.3.8 Keyword library

Requirements that apply to the keyword library include the following:

a) The keyword library shall support the definition of keywords that includes the basic attributes of name, description and parameters.

7.3.9 Script repository

Requirements that apply to the script repository include the following:

a) The script repository shall support the storage of keyword execution code.

b) The script repository shall support the inclusion of references to allow keyword execution code to be associated with its corresponding keyword in the keyword library.

7.4 Advanced attributes of frameworks

7.4.1 General information on advanced attributes

This subclause defines additional attributes that are recommended to achieve the full benefits of Keyword-Driven Testing. Basic Keyword-Driven Testing is possible without these attributes. This subclause does not identify requirements necessary for compliance with this International Standard.

The following sub-clauses structure these attributes in terms of the components of Keyword-Driven Testing frameworks according to subclause 7.2.

7.4.2 General attributes

Keyword-Driven Testing frameworks should support documentation with the following information:

a) There should be high-level documentation recorded that describes the rules of how the keywords can be composed into test cases.

b) There should be high-level documentation recorded which describes the rules of how parameters are described.

c) There should be high-level documentation recorded which describes the rules of how parameters are passed.

d) There should be high-level documentation recorded describing how keywords are defined.

7.4.3 Dedicated keyword-driven editor (tool)

Recommendations that apply to the dedicated Keyword-driven editor include the following:

a) The Keyword-driven editor should provide a function for checking the syntax of the test cases composed of the keywords.

b) The Keyword-driven editor should provide the capability to track keyword usage and provide a cross-reference to indicate in which test cases and composite keywords each keyword is used.

c) During any syntax checking the Keyword-driven editor should check that only defined keywords are used in test cases

NOTE 1 For keywords that have parameters, the Keyword-driven editor should check the correctness of each parameter count, and type.

NOTE 2 The parameter count is the number of parameters which are provided when using a keyword. Examples for parameter types can be (not limited to) a number, text string or something as complex as an address.

d) Undefined keywords should be rejected or at least marked as undefined by the Keyword-driven editor.

e) There should be a capability to define exception handling (e.g. if an exception occurs on test execution, it should be possible to define which clean-up steps are executed) within the Keyword-driven editor.

f) The Keyword-driven editor should allow auto-completion or drag and drop for allowed keywords and their parameters.

g) The Keyword-driven editor should support versioning of keyword test cases.

7.4.4 Decomposer and data sequencer

Recommendations that apply to the decomposer and data sequencer include the following:

a) The decomposer and data sequencer should allow users to implement new keywords (hierarchical keywords) using existing keywords.

b) The decomposer and data sequencer should allow users to create hierarchical structured data from values or other structured data.

7.4.5 Manual test assistant

Recommendations that apply to the manual test assistant include the following:

a) The manual test assistant should provide the capability to attach screenshots or other outputs of the test item to the test log.

7.4.6 Tool bridge

No advanced attributes are defined for the tool bridge.

7.4.7 Test execution environment and execution engine

Recommendations that apply to the test execution environment and execution engine include the following:

a) Keyword execution code should be able to read, store and process data from test items.

b) Variable name space support should be provided.

NOTE 1 Variables defined in configuration files for individual applications could otherwise conflict if the keywords are used in the same test case i.e. a multi- application test case.

c) Context switching when moving between applications in a test case should be supported.

d) Implementations should manage the switch between namespaces (e.g. when changing application references).

e) Testing that multiple users of an application can access the same shared data in parallel should be supported.

f) The execution engine should handle blocked keywords on the test item by continuing test execution with the next appropriate keyword (see 5.3.4)

NOTE 2 The next appropriate keyword is either defined by the test designer, or, if no such definition has been done, the next keyword in the test case.

g) The execution engine should be capable of handling keywords with attributes such as "may be blocked" and "must not be blocked" (see 5.3.4)

h) The execution engine should support data-driven tests.

NOTE 3 This includes, at a minimum, a looping construct that allows iteration over a set of one or more keywords, using data values read from an external data file. See 6.4 for a detailed discussion.

i) There should be a capability to define conditional actions.

j) Support for an application level configuration file should be implemented.

NOTE 4 The configuration file contains zero or more variable/value pairs. The scope of the variables extends to all tests in the test set that executes against the specified test item.

k) An implementation should support at least one instance of a configuration, but is free to support a scheme where more than one configuration file is used.

l) When an exception is handled, there should be a capability to skip actions and ensure that defined clean-up steps are executed.

NOTE 5 This includes the ability for the keyword execution code to request that the test case abort, i.e. those subsequent keywords should not be executed, usually as a result of an unrecoverable situation detected in the requesting keyword.

m) Each execution code for a keyword should be able to allocate the information needed to perform the required actions, such as input parameters or the object of the action. Each step contains all information for performing the action.

n) The execution engine should be able to verify whether keywords received by tables, test management tool etc., match their keyword execution code by comparing count and type of parameters.

o) The execution engine should ensure that all loops in keyword test cases are restricted in a way that infinite loops are prevented.

p) The execution engine should support loops limited by a given number of passes.

q) The execution engine should support loops limited by a fixed time period.

EXAMPLE Limit a loop to wait for a maximum time of 2 seconds, until an event occurs or is expected not to happen anymore.

7.4.8 Keyword library

Recommendations that apply to the keyword library include the following:

a) The keyword library should support the construction of composite keywords from keywords.

b) The keyword library should support versioning of keywords.

c) The keyword library should support the implementation of aliasing, synonyms and internationalization to facilitate the creation of test cases.

7.4.9 Test data support

Recommendations that apply to the test data support provided by the Keyword-Driven Testing framework include the following:

a) The framework should support versioning of test data.

b) The framework should allow the definition of hierarchical data types.

7.4.10 Script repository

Recommendations that apply to the script repository include the following:

a) The framework should support versioning of keyword execution code.

8 Data interchange

Keyword-Driven Testing can be supported by software tools which are components of the Keyword-Driven test framework. The application of the tools requires the capability of the tools to receive (input) and provide (output) the necessary data. The requirements on test data interchange are discussed in this clause.

Keyword-Driven Test data can be interchanged between tools. Data interchange between humans and a software tool, mostly at the user interface, will not be addressed here.

NOTE The term "tool" can refer to both commercial tools and custom-built (non-commercial) parts a framework.

Data interchange in Keyword-Driven Testing should be done by using a standard published by an internationally-recognized standardization body (e.g. ISO, IEEE or OMG).

Annex A
(normative)

Conventions

The following are conventions for keywords:

a) Keywords should contain a verb.

b) Keywords should use the imperative form.

c) Keywords shall provide a description of the associated set of actions.

d) Keyword descriptions should be unambiguous.

e) Keywords should be defined in a way that they are understandable by the stakeholders who will use them when designing test cases.

NOTE 1 This can be verified by reviewing the keywords with the stakeholders.

f) Every keyword shall be unique in its meaning within a framework.

The following example is meant to illustrate items a) to f).

EXAMPLE The keyword "pressButton" contains a verb (a) in imperative form (b). The description could be "This keyword is used to trigger an element of class <button> in the graphical user interface" (c). If it is associated with a parameter that identifies the button (e.g. "pressButton <cancel>" it is unambiguous (d)). This keyword is assumed to be understandable (e) by English speaking stakeholders, as the words "press" and "button" in the keyword's name are taken from the testers' usual vocabulary. Uniqueness of meaning (f) is given as long as no other keyword is introduced which refers to the same activities.

NOTE 2 Natural language can be ambiguous, contain synonyms and homonyms, and can result in unclear and ambiguous test cases.

NOTE 3 Deriving keywords from programming languages is not advisable. Programming languages can be too abstract or difficult to understand. Knowledge of a programming language by the domain experts who will specify the test cases cannot be assumed.

Annex B
(informative)

Benefits and Issues of Keyword-Driven Testing

B.1 General benefits of Keyword-Driven Testing

By composing all of the test cases from a fixed and defined set of keywords, the benefits can include the following:

— Keywords can be defined in natural language meaning that, test cases can be written with more or less detail, depending on the project's needs.

— Test cases become clear and understandable. This supports efficient manual test execution.

— Using unambiguous and precisely defined keywords allows the option to select whether the execution of a test case is done manually, or is done with automation. In the case of automation, it is expected that the keywords will be implemented as keyword-scripts.

— Testers working at the business level do not require technical understanding of the test automation framework to be able to create and edit test cases.

— Testers working at the technical level can implement or perform keyword-driven test cases, even if they have limited or no understanding of the business domain.

— Testers on a technical level can implement test cases using a language that is understandable to domain experts and that can be reviewed by them for business correctness. If this is done, then Keyword-Driven Testing can help to close a frequently perceived gap between the business level and the technical level.

— Maintenance of the keyword scripts at the technical level is unlikely to affect the test cases. So, in general, there is no need in re-specifying or re-formulating the keyword test case if the technical implementation of the keywords is adjusted.

— Sensitivity to changes (which can create the need for maintenance effort) is reduced.

— Portability of test suites is easier to achieve, (e.g. if a similar system with almost the same business cases has to be tested then many of the keywords can be reused).

— Test cases composed of keywords can be created faster than those written in natural language.

— Refactoring of test cases is cheaper.

B.2 Benefits of Keyword-Driven Testing for test automation

Benefits of Keyword-Driven Testing in the case of test automation can include:

— Automated functional tests can be implemented before the test item actually exists, either by using existing keyword libraries with their corresponding automation scripts, or by defining new keywords and adding the automation scripts later as the test interface is defined.

— A limited set of keywords implies a limited effort for implementing test automation, (e.g. usually, one automated keyword script for each keyword will be sufficient).

— As long as test cases are constructed from the established set of keywords, once these keywords have been implemented, new test cases do not need any additional implementation effort to be automated.

— Maintenance of test cases for business reasons will not affect the implementation of the keyword scripts, as long as the set of keywords and the semantics of the keywords are not changed.

B.3 Benefits of Keyword-Driven Testing for manual testing

When using Keyword-Driven Testing with manual testing the benefits can include the following:

— Faster test execution can be achieved because the tester remembers the functionality of a reused keyword and no interpretation effort is needed for reused keywords.

— Testers are guided more precisely to achieve test-to-test repeatability and consistency.

NOTE Keyword-Driven Testing is one approach of gaining these benefits. There can be other approaches to achieving similar benefits.

B.4 Possible issues with Keyword-Driven Testing

Using Keyword-Driven Testing can result in additional costs and also in a delay in constructing test cases which can equate to higher project costs. In later project phases, these initial investments can pay off due to the benefits listed earlier in this annex, including faster implementation of additional test cases and saving time when editing test cases. Realizing these benefits may be more difficult for short-term projects that only require a very limited number of test cycles.

Using Keyword-Driven Testing instead of traditional test specification in natural language affects project costs. While the benefits mentioned in the previous clauses of this annex are expected to reduce the project costs, the following possible issues may add to the project efforts:

— In the initial phase, when Keyword-Driven Testing is started, keywords need to be identified, and in case of desired test automation, implemented and tested. This is a considerable additional effort that needs to be considered in planning.

— Personnel has to be trained to use keywords for test case specification.

— Continuous maintenance and support of the keyword library will require support staff, budget and time to be assigned. These will need to be considered when designing the keyword library. The additional effort may result in delay for constructing test cases.

NOTE The additional effort pays off the more often the keywords and the implementation can be reused.

Annex C
(informative)

Getting started with Keyword-Driven Testing

C.1 General

This annex provides assistance for applying Keyword-Driven Testing. It is offered to help those who do not have experience with Keyword-Driven Testing but want to learn how to start doing it.

In many cases, Keyword-Driven Testing will be done by performing two major activities that are described in the following subclauses:

— Identifying Keywords

— Composing Test Cases

Although these activities can be conducted sequentially, they will often be applied iteratively or concurrently. This is especially true if Keyword-Driven Testing is already established, and, while defining new test cases, the test designer recognizes the need for further Keywords.

However, in principle, both activities are required, and when starting Keyword-Driven Testing, it is advisable to focus on these activities.

C.2 Identifying Keywords

Keyword-Driven Testing requires the identification and definition of Keywords.

There are several sources which can be used to identify Keywords, which include the following:

a) Exploratory testing
During exploratory testing, the tester observes which steps are performed. Some steps are related and are performed together. A new keyword is defined by assigning a meaningful name to this collection of steps.

If the sequence of steps can be used with different data, the keyword will take parameters according to that data.

To document that keyword, the name, the steps, a description, and when applicable, the parameters are noted. Once these activities are completed, when defining new test cases, instead of using the steps, the name of the keyword will be used.

b) Business experts
Keywords can also be defined by interviewing business experts. A test analyst asks questions of the business expert. These questions can be "what should the application do?", "how can I verify proper behaviour?" or "what needs to be tested?". The answers provided by the business experts will naturally be formulated in a business or domain related language. A test analyst can now identify keywords by finding core terms which probably occur frequently.

It is possible that there are different terms (used by business experts) referring to the same set of activities; a test analyst needs to be aware of that and try to identify duplicates.

Starting from the names of the keywords which have been agreed on with the business experts, the test analyst needs to work out which steps are involved with that keyword. The documentation is performed as in a) above.

c) Test interface
Keywords can also be defined starting from the test interface. As the number of interface elements is limited and usually small, a limited number of keywords can be defined addressing these interface elements. And contrary to a) and b) above, which define high-level keywords on a domain layer, this approach will define low-level keywords on a test interface layer.

This approach can be rewarding if there is a focus on test automation, as the final limited set of keywords can be matched with keyword execution code. If a proper Keyword-Driven test framework is available, specified test cases can be available as automated tests almost immediately. The documentation is performed as in a) above.

d) Documented test procedures and test cases
Available test procedures and test cases can also be a valuable source of keywords. The actions in the test cases are examined. As a first step, each action can be treated as a new keyword. If two or more of these keywords turn out to refer to the same activities, they will be replaced by only one keyword, which best describes the activities.

If some of the keywords only occur in a certain sequence with others, they can be replaced either by one higher level keyword, or by a hierarchical keyword. The documentation is performed as for a) above.

It can be sufficient to use only one of these sources, but usually information from several sources is used.

In all cases, it can happen that the created pool of keywords is not sufficient to describe test cases, as there may be activities that were not recognized as requiring a keyword. These "gaps" will be filled by defining the missing keywords as test effort progresses.

C.3 Composing test cases

Once a basic set of keywords has been defined, these keywords can be used to describe test cases.

The test cases first need to be identified using test techniques as defined in ISO/IEC/IEEE 29119-4, and along with the process steps defined in ISO/IEC/IEEE 29119-2 (e.g. the activities of TD4 which are documented according to ISO/IEC/IEEE 29119-3).

While writing down the actions for the test cases, instead of describing the necessary activities in natural language, the predefined keywords are used.

If several test cases share the same sequence of actions or keywords, but their test data is different, they can be joined into one keyword test case along with different sets of test data.

Annex D
(informative)

Roles and Tasks

D.1 Overview – Roles and Tasks

A sound framework for Keyword-Driven Testing allows different tasks to be performed by different people, which can require different skills, such as test automation skills for the test interface layer, and business knowledge or test skills for the domain layer. This clause describes different roles in Keyword-Driven Testing.

NOTE 1 More roles can be involved in Keyword-Driven Testing or in the test process in general. In this clause, only those roles which are specific for the division of labour supported by Keyword-Driven Testing are discussed.

A single person can be assigned one, several or even all of these roles; but for best efficiency, and to reflect the different capabilities of team members, it can be advisable to assign the roles to different people. This is especially helpful in cases where a single person with all the required skills is not available.

NOTE 2 The following roles can be named differently in practice and the roles' activities can vary.

D.2 Domain expert

The domain expert is often the actual or future user of the test item. A domain expert has in-depth knowledge of how the test item should behave. This knowledge can be focussed on business cases, but can also reflect technical aspects. A person assuming this role ideally should have basic knowledge of test techniques and processes.

Tasks for the domain expert can include the following:

— Provide parts of the test basis by defining use cases, business cases, or paths through the application which need to be considered.

— Design test cases, and contribute to the test design specification.

— Identify business keywords.

The domain expert will closely cooperate with the test designer to perform these tasks.

D.3 Test designer

The test designer analyses complex use cases, requirements documents, and specifications. From these, the test designer derives test cases and subsequently the useful business level keywords. Therefore it is crucial to be able to distinguish relevant and irrelevant information.

The test designer should be in close dialogue with a subject matter expert in addition to analysing requirements or other product information (operation concepts, user guides, etc.) in order to derive useful business-level keywords.

Tasks for the test designer can include the following:

— Define keywords and their interfaces.

— Specify keywords composition and application in test cases.

— Create test cases based on the test basis.

— Rework any rough test cases from domain experts to create test cases and test procedure specifications.

The test designer will closely cooperate with the domain expert to make sure that keywords reflect the domain's language and the test cases are appropriate.

The test designer will closely cooperate with the test automation expert to make sure that the interfaces of the keywords are consistent and that the keyword execution code reflects the keywords in the right way.

D.4 Test automation expert

This role is only needed if test execution is to be automated.

The test automation expert needs to have experience in programming and knowledge about the test tools. The test automation expert needs to understand the scripting languages used in the framework which supports the automated test execution. Furthermore experience as a tester is beneficial and simplifies communication with other testers.

Tasks for the test automation expert can include the following:

— Implement the low-level keywords as executables and help ensure their functionality for test automation.

— Build the framework by selecting and combining appropriate tools on a technical level by adopting or implementing libraries.

— Together with the test designer, the test automation expert decides how to combine low-level keywords with higher level keywords and provides the technical means to support this from the automation side.

— Maintain test implementation scripts or the relevant parts of the scripts help ensure the availability and reliability of the test automation to avoid making a guarantee.

The test automation expert will closely cooperate with the test designer.

Annex E
(informative)

Basic keywords

E.1 Overview

This annex provides a basic list of keywords as an example. These keywords can be applied on a GUI as a test interface. The set of keywords is supposed to be generic and usable for most applications on this test interface. In practice, a test item may require more or different keywords than provided here. Therefore this set of keywords is extendable.

This set of keywords can be useful as an example for other test interfaces, and is offered as a quick start for using Keyword-Driven Testing.

E.2 Basic keywords for a GUI

In the following, basic keywords are listed for testing a GUI. A GUI typically has dialogs that are distinguished by unique titles. Within a dialog there are various GUI-objects. The dialog and its GUI-objects are identified by an identifier (id).

Keyword	Description
`clearContext (id)`	Removes the context from a component. Parameters: id (IN): id of the component
`click (id)`	Simple click with left mouse button on a given component. Parameters: id (IN): id of the target component
`clickWithOptions (id, MOUSE_BUTTON, times, MODIFIER, x, y)`	Extended click with additional options on a given component. Parameters: id (IN): id of the target component MOUSE_BUTTON (IN): one of the values which are defined in parameter MOUSE_BUTTON times (IN) OPTIONAL: how many times to click MODIFIER (IN) OPTIONAL: one of the values which are defined in parameter MODIFIER x (IN) OPTIONAL: x-coordinate relative in component y (IN) OPTIONAL: y-coordinate relative in component
`doubleClick (id)`	Double click with left mouse button on a given component. Parameters: id (IN): id of the target component
`drag (id, item, MOUSE_BUTTON, KEY)`	Drag. The parameter item is provided for selecting the drag source from a tree or list. Other components are normally not dragable. Parameters:

Keyword	Description
	id (IN): id of the target component item (IN) OPTIONAL: in case of a tree or list the id of the treeNode or the listItem MOUSE_BUTTON (IN): one of the values which are defined in parameter MOUSE_BUTTON KEY (IN): one of the values which are defined in parameter KEY
`drop (id, item)`	Drop. Parameters: id (IN): id of the target component item (IN) OPTIONAL: in case of a tree or list the id of the treeNode or the listItem
`getCaption (id, varCaptionValue)`	Writes the caption of target component (id) into varCaptionValue. Parameters: id (IN): id of the target component varCaptionValue (IN): variable with caption of component
`getProperty (id, PROPERTY_NAME, varPropertyValue)`	Writes the value of the given property (PROPERTY_NAME) of the target component (id) into varPropertyValue. HINT: In case of no hit the interaction fails and the parameter gets the value UNDEFINED. Parameters: id (IN): id of the target component PROPERTY_NAME (IN): one of the properties which are defined in parameter PROPERTY_NAME varPropertyValue (IN): variable with value of property
`getText (id, varText)`	Writes the text of the target component (id) into varText. Parameters: id (IN): id of the target component varText (IN): variable with text of component
`moveMouse (target_id, target_item)`	Move the mouse to the component with id target_id. Parameters: target_id (IN): id of the target component target_item (IN) OPTIONAL: in case of a tree or list the optional id of the treeNode or the listItem
`openContextMenu (id)`	Opens the context menu of the given component. Parameters: id (IN): id of the component
`pressKey (id, MODIFIER, KEY)`	Presses a key or key combination (with modifier). Please note: this interaction is intended for testing keyboard commands and shortcuts. For entering text into a text area or text field, please use the "setText" interaction instead. Parameters: id (IN): id of the target component or the value "UNUSED" in which case the key press happens on the currently focussed component MODIFIER (IN) OPTIONAL: a combination of one or more modifier keys (such as "shift" or "alt") KEY (IN): the key to be pressed

Keyword	Description
`setContext (id)`	Sets the context 'passively' for the given component (programming construct). IMPORTANT: This is in contrast to setWindowActive which brings a window / dialog 'actively' to the foreground. Parameters: id (IN): id of the component
`setFocus (id)`	Sets the focus on the given component. IMPORTANT: For cells, tree items, list items and menu items it isn't possible to set the focus. The focus can only be set for tables, trees, lists and menus, respectively. Parameters: id (IN): id of the component
`setText (id, text)`	Sets or clears text in the given component. Parameters: id (IN): id of the component text (IN): the text
`verifyCaption (id, OPTION_PATTERN_MATCHING, expectedCaptionValue)`	Verifies the expected caption (expectedCaptionValue) of the target component (id) regarding search algorithm (OPTION_PATTERN_MATCHING). Parameters: id (IN): id of the target component OPTION_PATTERN_MATCHING (IN): specifies the format of search algorithm expectedCaptionValue (IN): expected caption
`verifyProperty (id, PROPERT_NAME, OPTION_PATTERN_MATCHING, expectedPropertyValue)`	Verifies the expected property value (expectedPropertyValue) of the target component (id) regarding search algorithm (OPTION_PATTERN_MATCHING). Parameters: id (IN): id of the target component PROPERTY_NAME (IN): one of the properties which are defined in parameter PROPERTY_NAME OPTION_PATTERN_MATCHING (IN): specifies the format of search algorithm expectedPropertyValue (IN): variable with value of property
`verifyText (id, OPTION_PATTERN_MATCHING, expectedCaptionText)`	Verifies the expected text (expectedText) of the target component (id) regarding search algorithm (OPTION_PATTERN_MATCHING). Parameters: id (IN): id of the target component OPTION_PATTERN_MATCHING (IN): specifies the format of search algorithm expectedCaptionText (IN): expected caption
`waitForExist (id, maxTimeToWait)`	Waits until a component exist. Parameters: id (IN): id of the target component maxTimeToWait (IN): waiting period in milliseconds
`waitForNotExist (id,`	Waits until a component does not exist.

Keyword	Description
`maxTimeToWait)`	Parameters: **id** (IN): id of the target component **maxTimeToWait** (IN): waiting period in milliseconds

Table E.1 — Example of generic basic keywords

There are further GUI objects that require specific, specialized or extensible keywords.

Examples can be found in the table below:

Keyword	Description
`selectMenuItem(id, OPTION_NUMBER_OR_NAME, menuItem)`	Selects a menu item (depends on value of OPTION_NUMBER_OR_NAME). Parameters: id (IN): id of the target menu OPTION_NUMBER_OR_NAME (IN): defines whether the following parameter is defined by its name or number menuItem (IN): name or number of menu item dependent on value of OPTION_NUMBER_OR_NAME
`selectListItem (id, MODIFIER, OPTION_NUMBER_OR_NAME, listItem)`	Selects one list item. MODIFIER allows to define a multiselect interaction by repeating this interaction. The list items can be given by name or number (dependent on the value of OPTION_NUMBER_OR_NAME) . A list can be a list view, a combobox, a dropdown list, radio button or tabcard. Parameters: id (IN): id of the target list MODIFIER (IN) OPTIONAL: one of the values which are defined in parameter MODIFIER OPTION_NUMBER_OR_NAME (IN): defines whether the following parameter is defined by its name or number listItem (IN): name or number of list item dependent on value of OPTION_NUMBER_OR_NAME
`startApplication (currentClientID, propertyFile)`	Starts the application, using a property file for application settings. The property file is expected to be in the directory properties. Parameters: currentClientID (IN): id of the application propertyFile (IN): filename of the property file

Table E.2 — Example of specialized basic keywords

E.3 Example application of basic keywords

The following example shows a keyword test case structured in three layers: low-level keywords are used at the test interface layer, an intermediate layer combines the low-level keywords to an application-related vocabulary, and business keywords use these keywords from the intermediate layer at the domain layer.

Each keyword is written in a function-like style, i.e. it consists of a unique name followed by none, one or more parameters placed in braces. The parameters used at the domain layer are passed through the intermediate layer to the test interface layer.

This test of a car's configuration program addresses the use case for configuring a car with some accessories. As a final action, the calculated price will be verified.

In practice, in this example a test case would be written using only domain layer keywords. The other layers are provided for a better understanding.

domain layer	intermediate layer	test interface layer
startCarConfigurator (“login”, “password”, “english”)		
	startCarConfiguratorCmdLine()	
		startApplication (“carConfigurator”, “E:\CarConfigurator.ini”)
	login (“login”, “password”)	setText (“userField”, “login”)
		setText (“pwdField”, “password”)
		click (“loginBtn”)
	setLanguage (“english”)	
		selectMenuItem (“mainMenu”, NAME, “Language”)
		selectMenuItem (“menuLanguage”, NAME, “english”)
selectVehicle (“Rolo”, “red”)		
	selectTabcard (“Cars”)	
		setContext (“carConfig”)
		selectListItem (“tabbedPane”, UNUSED, NAME, “Vehicles”)
	selectVehicleByNameAndColour (“Rolo”, “red”)	
		selectListItem (“vehicleList”, UNUSED, NAME, “Rolo”)
		selectListItem (“colourList”, UNUSED, NAME, “red”)
selectAccessories (“[Steering wheel, brown, leather]”, “[Mats, black, textile]”)		
	selectTabcard (“Accessories”)	
		setContext (“carConfig”)
		selectListItem (“tabbedPane”, UNUSED, NAME, “Accessories”)
	selectAccessoriesByNameColourMaterial (“Steering wheel”, “brown”, “leather”)	selectListItem (“accessoryNameList”, UNUSED, NAME, “Steering wheel”)