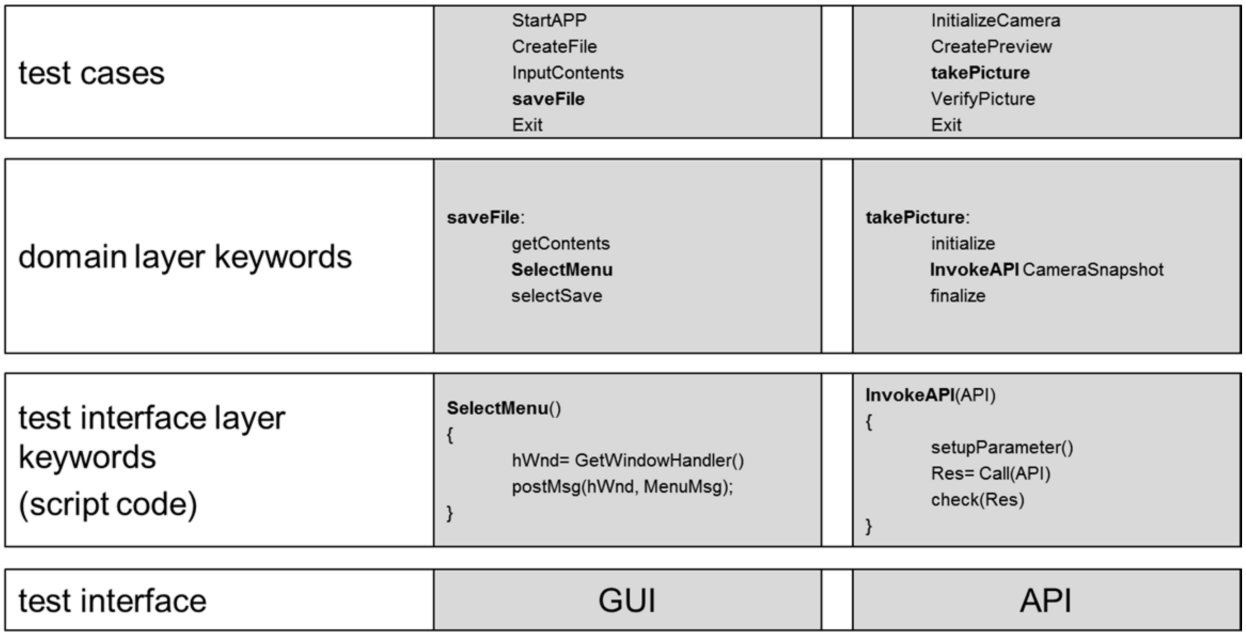

Figure 2 — Example for defining test cases by keywords at several layers

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).

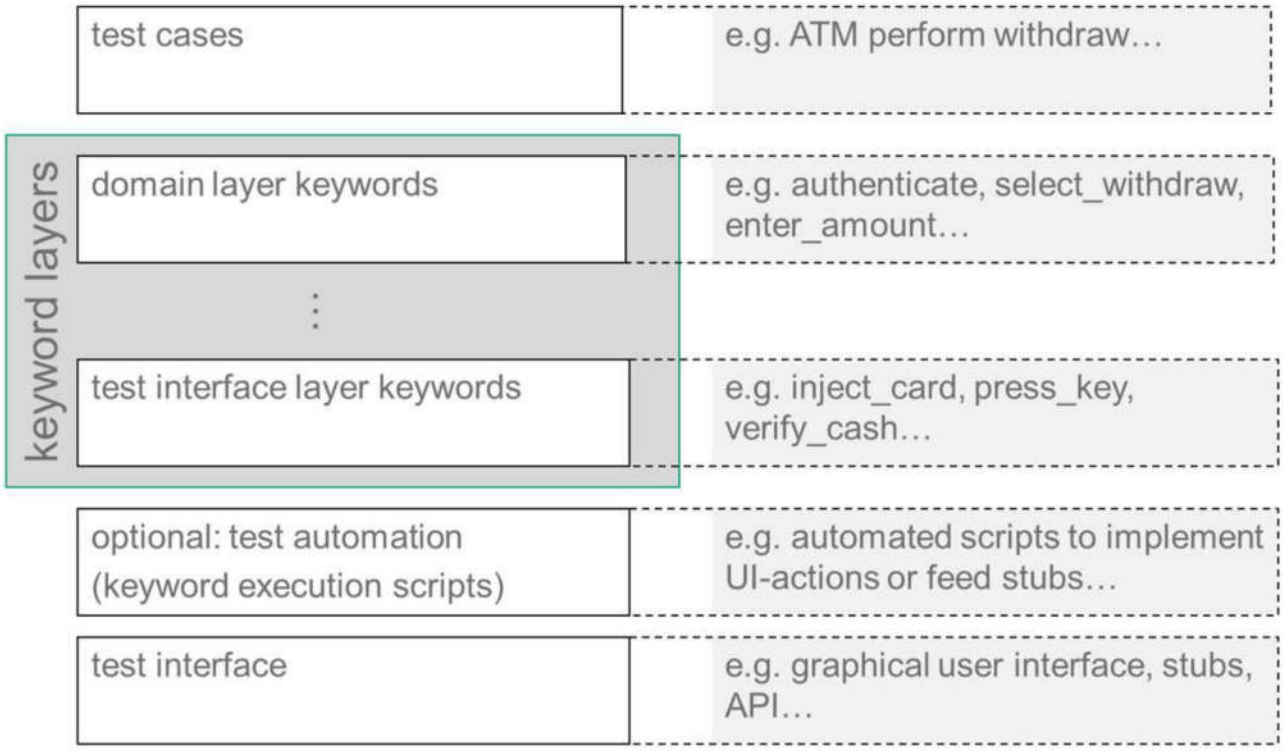

Figure 3 — Multiple layers in Keyword-Driven Testing

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.

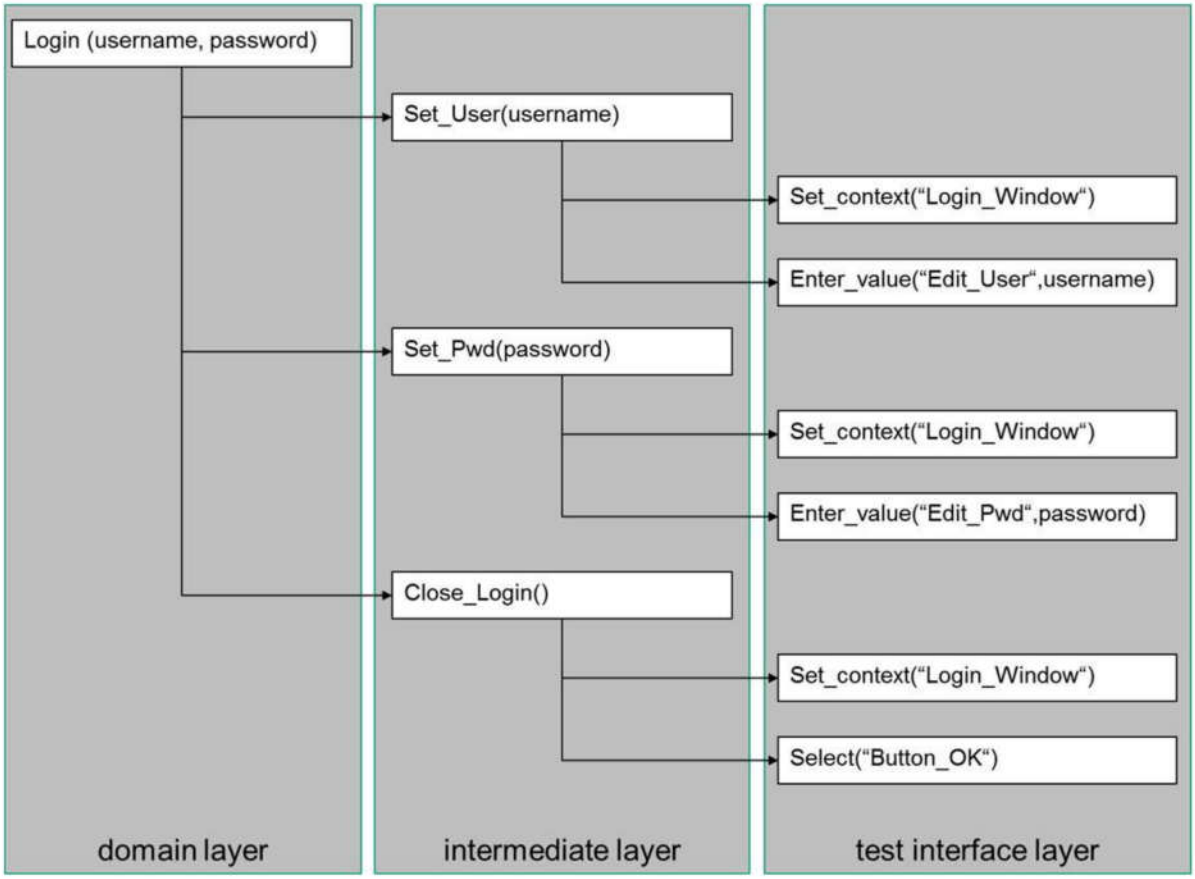

Figure 4 — Example for using composite keywords with data

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).

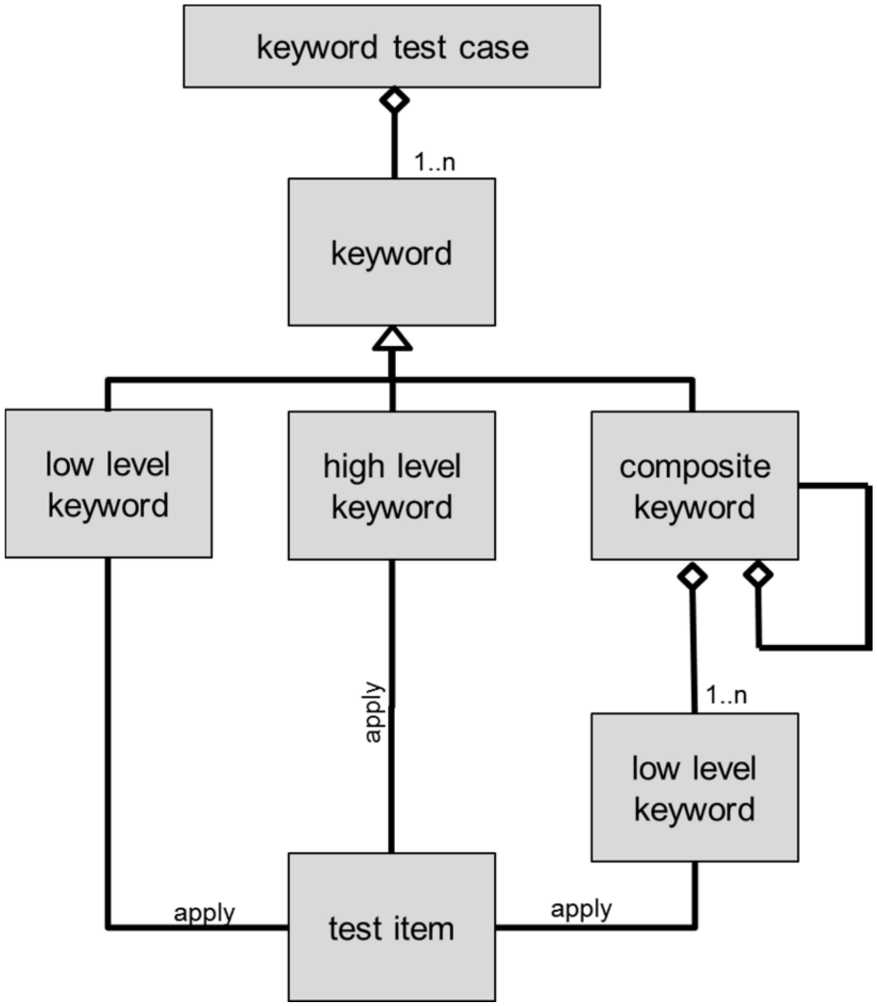

Figure 5 — Keyword test cases composed from keywords at different levels

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.

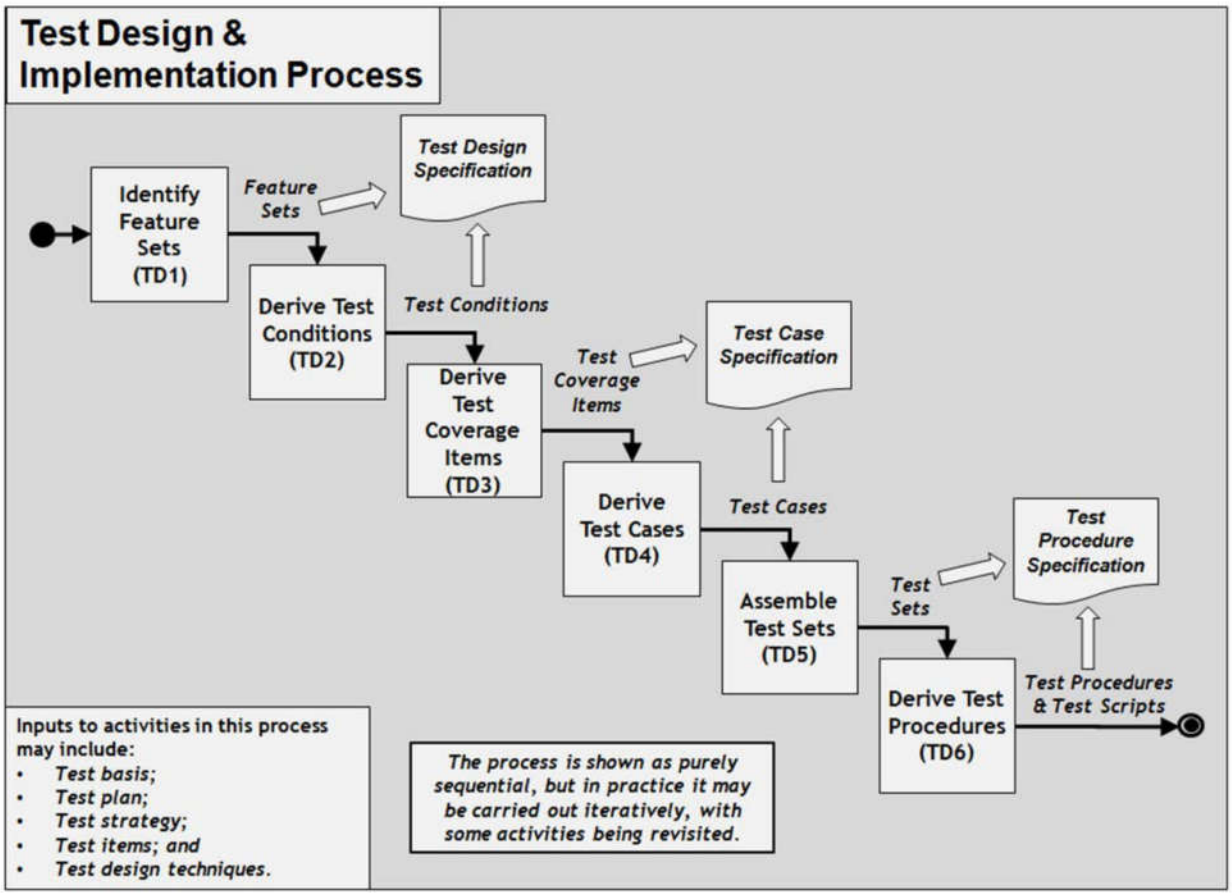

Figure 6 — ISO/IEC/IEEE 29119-2 Test Design & Implementation Process

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.