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).

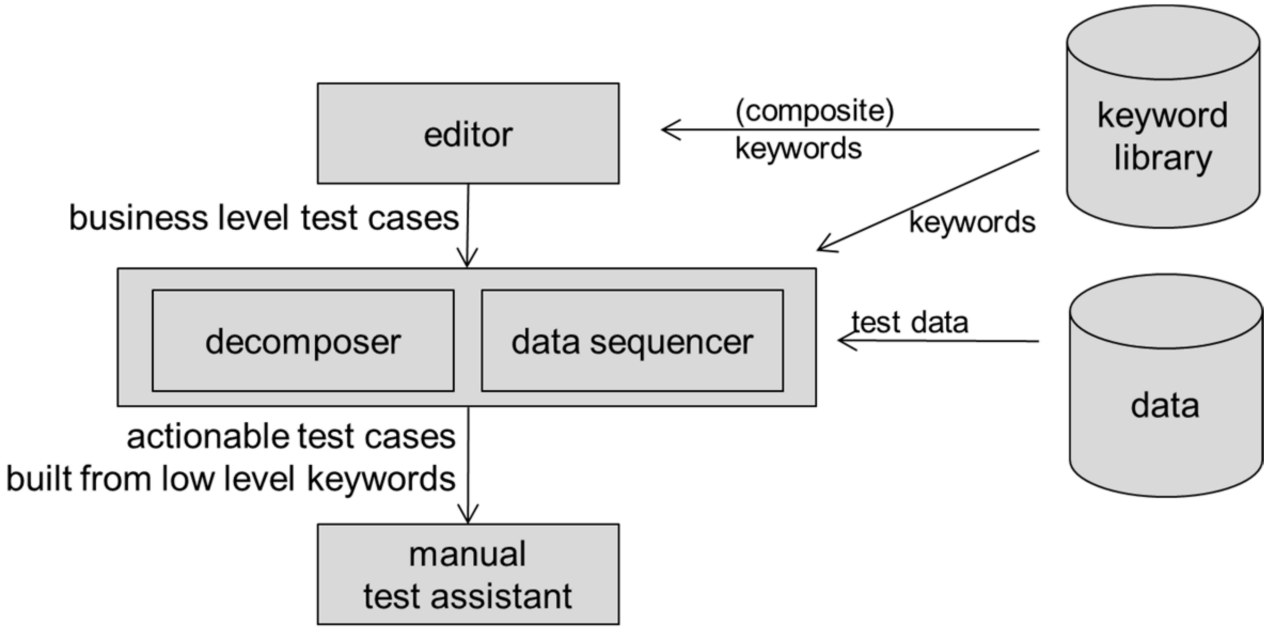

Figure 7 — Components of a Keyword-Driven Testing framework for manual test execution

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.

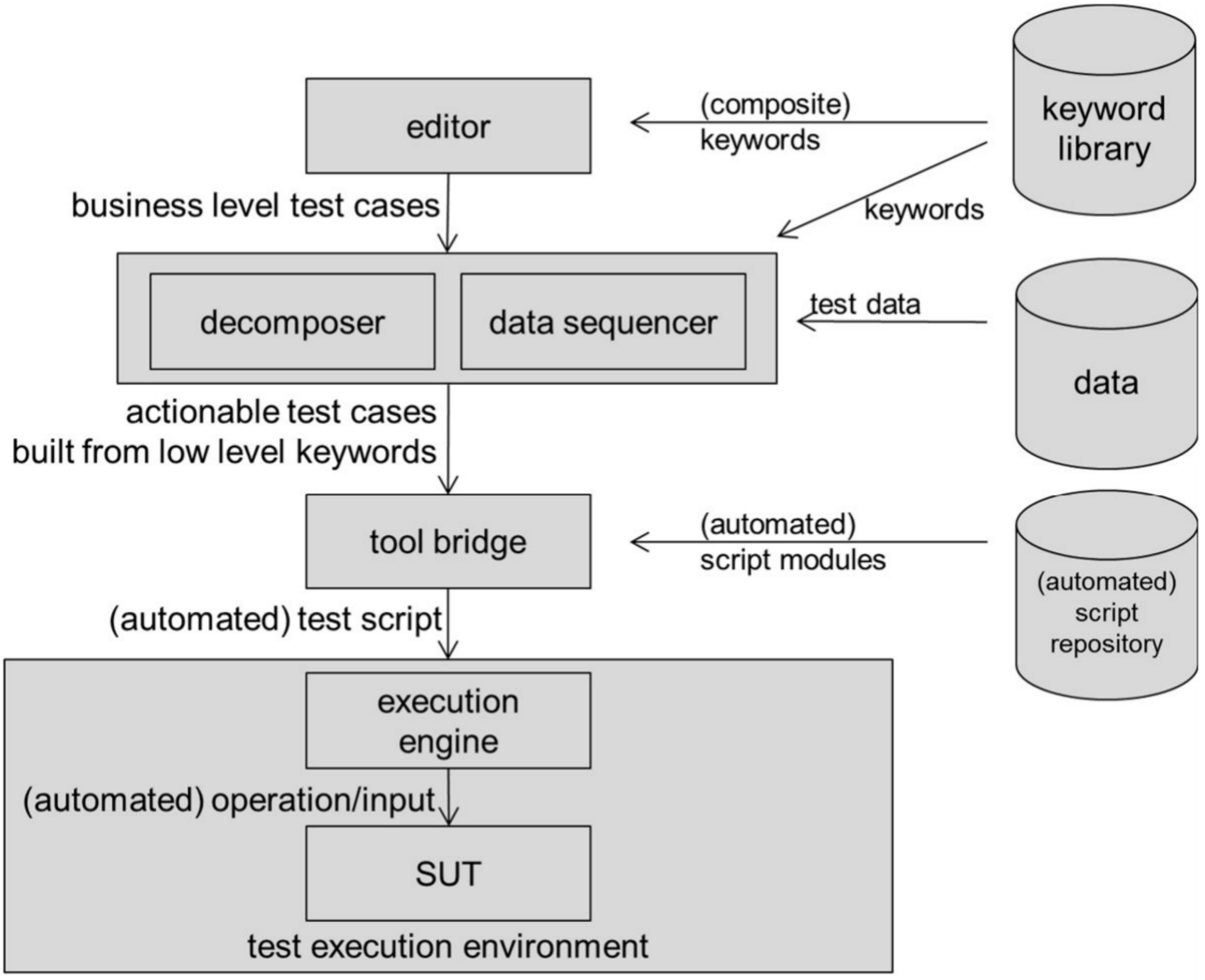

Figure 8 — Components of a Keyword-Driven Testing framework for automated test execution

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.