TESTING POLICY

Wouter Van Caneghem
Functional Analyst - Fedict

Dirk Dussart
Architect - Fedict

Contents

Date	Version	Description	Author
11/10/2011	1.0	Initial draft version	Wouter Van Caneghem
17/10/2011	1.1	Addition of chapter entitled “Tools”	Wouter Van Caneghem
21/10/2011	1.2	Addition of roles and responsibilities	Wouter Van Caneghem
17/11/2011	1.3	Addition of unit testing	Diederik Delen
12/12/2011	1.4	Addition of test matrix	Wouter Van Caneghem
13/12/2011	1.5	Addition of staging criteria	Wouter Van Caneghem
27/02/2012	1.6	Update of staging criteria and addition of annex	Wouter Van Caneghem
05/03/2012	1.7	Update of test matrix	Dirk Dussart
13/04/2012	1.8	Review of Coralius nv	Coralius nv
14/05/2012	1.9	Amendment to the severities	Wouter Van Caneghem

1. Introduction

1.1. PURPOSE OF THIS DOCUMENT

The purpose of this document is to provide a clear statement as to Fedict's requirements with regard to the testing process for software and to outline who is responsible for which parts of that process. The requirements stated in this document shall apply to all projects. In the specific contract documents or requirement documents, however they may be adapted in line with the needs that apply to the particular project or product concerned.

An additional purpose of this document is to help the supplier to provide a product of the highest possible quality from the first delivery onwards, thereby creating a win-win situation for both parties. In doing this, our aim is to ensure that the effort expended by Fedict for the purpose of testing and retesting and the effort expended by the supplier for the purpose of reworking the product is kept to a minimum.

Chapter 2 contains an overall summary of the testing process, based on the V-model. In that part of the document, we will indicate which test levels lie within the scope of responsibility of Fedict and which ones are the responsibility of the software supplier.

In Chapter 3, we will provide a summary of the deliverables to be provided, together with details of the requirements with which these are required to comply.

In Chapter 4, we will provide additional recommendations regarding the way in which a supplier will be able to fulfil the requirements set, without seeking to impose a specific development cycle or methodology.

In Annex 1, we will provide a list of templates that form part of this policy and will explain how these can be used.

1.2. WHY IS TESTING REQUIRED?

Testing forms an essential precondition to ensure the successful construction and implementation of information systems. The complexity of modern-day software is such that it is almost impossible to implement it correctly the first time around, without any form of verification.

Testing is needed in order to detect potential problems within the software as early as possible, so that they can be corrected at minimum cost.

A second reason to carry out tests is to develop trust in and a knowledge of the product provided.

Defects that exist within a software product can have severe consequences for the "business" and the users alike. Whilst providing a means of avoiding faults as much as possible, testing is also a useful way of demonstrating to management and users that the product supplied fulfils their requirements (is "fit for purpose").

It is important to note in this regard that both the functionality and the non-functional software characteristics play a significant part in asserting that a product fulfils the stated requirements and is useable in its operational context.

1.3. WHAT IS TESTING?

Testing software takes the form of a process that is used to **verify** and **validate** that a software program, application or product:

- Fulfils the business and technical requirements set out in the contract documents, the requirements, the analysis and design documents
- Works as expected and can be used within its operational environment
- Has been implemented with the required non-functional software characteristics

In this regard, verification and validation must be interpreted in the sense of the ISO-9000 standard:

- **Verification:** Confirmation, through the provision of objective evidence, that the stated requirements have been fulfilled.
- **Validation:** Confirmation, through the provision of objective evidence, that the stated requirements for a specific, intended use or application have been fulfilled.

We can interpret this by asserting that verification involves confirming that the software has actually been created, whilst validation sets out to establish that the correct software has been created.

1.4. BASIC PRINCIPLES

A number of principles apply to all forms of testing:

- Principle 1: Testing reveals defects.

  Testing reveals defects that are present, but is unable to provide evidence that no defects are present. Testing reduces the likelihood that the software contains undiscovered defects, but if no defects are found, this cannot be regarded as proof that no defects are present.

- Principle 2: Exhaustive testing is impossible.

  Comprehensive testing (all combinations of inputs/outputs and preconditions) is not feasible, except in trivial cases. Instead of carrying out extensive testing, risk analyses and priorities must be used in order to restrict the effort involved in carrying out tests to the tests that genuinely need to be carried out.

- Principle 3: Test at an early stage.

  The testing activities must begin as early as possible within the software development cycle. This will ensure that the defects are detected at an early stage, with the result that rectifying the defects will be less costly.

- Principle 4: Clustering of defects.

  A small number of the modules contain the largest number of defects discovered during the pre-release tests and/or are responsible for the most operational errors.

- Principle 5: The pesticides paradox.

  If the same set of test cases are carried out once again each time, there will come a time when they no longer reveal any defects. That is the reason why the test cases need to be re-examined on a regular basis. New tests must be written in order to verify different parts of the software, so that new defects may be discovered.

- Principle 6: Testing is context-dependent.

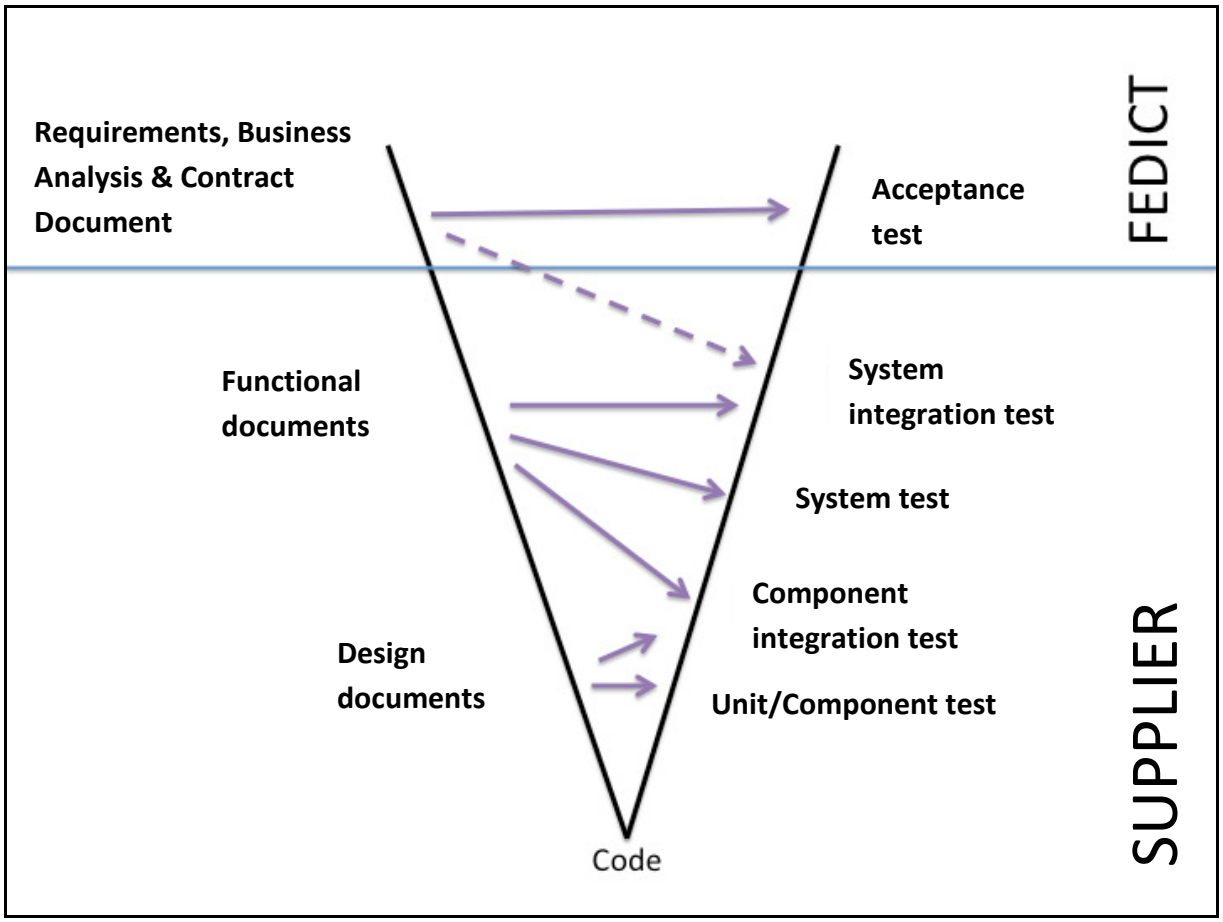
  The test method employed will depend on the context in which the product will ultimately be used. For example, security software will be tested in a different way to an e-commerce website.

- Principle 7: The absence-of-errors fallacy.

  Tracing and rectifying defects will be of no benefit if the system is unusable and/or does not fulfil the needs and expectations of the end-users.

2. The testing process and overall responsibilities

2.1. THE FEDICT V MODEL



The Fedict V model illustrates the levels of testing used, in addition to the overall responsibilities of Fedict and of the supplier.

2.2. LEVELS OF TESTING

Levels of testing take the form of groups of test activities that are collectively managed and directly related to the responsibilities defined as part of a project.

Fedict applies the testing levels as described in the rest of this section.

If a supplier uses different names or different testing levels, he must be able to demonstrate how the terminology he himself uses relates to the terminology used by Fedict. He must also be able to demonstrate that his approach is, at the very least, equivalent to the approach described by Fedict.

It is important to note that a testing level is not the same as a testing stage. Within an incremental or iterative development model, tasks that belong to a clearly-defined testing level will be carried out again and again at various stages of a project.

The V-model is used within this policy in order to indicate who will carry out what types of activity, but is not used as a means of imposing a specific software development cycle.

2.2.1. Reviews

Reviews form an excellent means of detecting faults at an early stage and in a cost-efficient way.

If the supplier expects one or more documents to be reviewed by Fedict, however, it must include a statement to that effect in the tender, listing the documents to be reviewed and providing an estimate of the amount of work involved on Fedict's part. In that regard, it will be possible to assume that a reviewing rate of 6 pages per hour will be sufficient to enable the effective identification of errors.

2.2.2. Unit test level

Unit tests (otherwise known as component tests) seek to identify defects in, and verify the functioning of, testable software modules, programs, items, classes etc.

Unit tests are capable of verifying functionalities, as well as non-functional characteristics. The test cases are based on work products, such as the specifications of a component, the software design or the data model. Unit tests are used in order to be certain that the individual components are working correctly.

Unit tests mainly make use of white-box testing techniques and are therefore carried out with a knowledge of the code being tested and possibly also with the support of the development environment (the unit test framework, debugging tool, etc.). That is why these tests are often carried out by the developer that wrote the code or by a party of a similar calibre. Any defects identified are resolved as soon as they are detected.

From Fedict’s perspective, the unit tests represent an effective alternative to inserting comments in the code, as long as they take place in a structured manner and in a readable form. In the majority of cases, a unit test describes the details, rather than providing a text written in comment style. Unit tests are also maintained, along with the code itself. Comments in the code are often forgotten, which means that comments will soon become dated.

Unit tests also possess a number of characteristics that affect the maintainability of the application. Tests ought to be simple in structure and must be easy to set up. Dependencies must be limited; using stub and driver frameworks make it possible to keep the test set-up simple. If writing a test proves to be a complex matter, the developer must consider implementing the functionality in a different way. Code that is difficult to test is often too complex, thereby increasing the likelihood of faults. One particular guideline that applies in this regard is to ensure that the cyclomatic complexity of individual functions or methods is kept below 15, wherever possible.

Tests must be clearly named. The method or test name must describe what the test sets out to do. Names such as test1 or test2 are unacceptable. It is also important to remember that Fedict must be able to verify the unit tests.

Unit tests must also be independent of one another and it must be possible for every one of them to be conducted separately.

2.2.3. Integration test level

As far as the integration tests are concerned, these set out to test the interfaces between the various components and the interactions with different parts of a system (such as the control system, file system, hardware etc.). Integration tests can exist in more than one level and they can be carried out on test items of various sizes.

We are able to distinguish between 2 separate levels of integration test:

- *Component integration tests:* The testing of the interaction between software components. These are carried out following the unit tests.
- *System integration tests:* The testing of the interaction between various systems. These are carried out following the system tests.

The greater the scope of integration, the more difficult it becomes to isolate defects that exist in a specific component or system.

At any point in the integration, the tests will concentrate solely on integration itself. For example: A new module A is integrated with a module B. During the integration tests, the primary focus will lie on the communication between the modules and not on the functionality of the modules themselves.

2.2.4. System test level

In the case of system tests, our aim is to test the behaviour of an entire system, as defined within the project. For the system tests, no knowledge is required, a priori of the internal design or logic of the system; the majority of techniques used are primarily black box techniques. In the case of the system tests, the environment must correspond as closely as possible with the ultimate production environment, in order to minimise the risk of environment-specific defects. System tests are derived from risk specifications, requirement specifications, business processes, use-cases or other high-level definitions and are carried out within the context of functional requirements. What is more, system tests will also investigate the non-functional requirements (quality attributes).

System tests typically include the following types of test: GUI tests, usability tests, regression tests, performance tests, error-handling tests, maintenance tests, compatibility tests, load tests, security tests, scalability tests, etc.

The intention is that once the system test has been carried out, the supplier is satisfied that the system complies with all of the necessary requirements and is ready to be handed on to the customer (possibly with the exception of interaction with peer systems).

2.2.5. Acceptance test level

Acceptance tests are often the responsibility of the customers or (end-)users of a system. Other stakeholders can also be involved, however. The purpose of acceptance testing is to gain confidence in the system, parts of the system or specific non-functional properties. Locating defects is not the primary aim of acceptance tests. Acceptance tests are a means of checking whether a system is ready for use.

Acceptance tests can occur at more than just one test level. For example:

- Acceptance testing of the usability of a component can take place during unit testing.
- Acceptance testing of a new functional improvement can take place before the system tests.

Acceptance tests typically involve the following elements, which may also be regarded as a separate test level:

- *User acceptance tests:* Verifying the readiness of a system for use by business users
- *Operational acceptance tests:* Testing the acceptance of a system by system administrators. These consist of tests of backup/restore, disaster recovery, user management, maintenance tasks etc.
- *Contractual and guideline tests (conformance test):* The acceptance criteria must be agreed during the contract negotiations. Acceptance then takes place on the basis of those criteria relating to the production of software. These also include standards (government standards, legal standards, security standards, etc.).
- In the absence of any specific agreements, the system must satisfy all of the applicable requirements, as described in the (signed) contract document and associated documents, in addition to the requirements described in all of the more technical documents approved by Fedict.
- *Alpha and beta-testing:* This involves obtaining feedback from potential or existing customers, before the product is launched on the market. Alpha-testing takes place within the developer's own organisation. Beta-testing (field-testing) is carried out by individuals in their own working environment.

2.3. TYPES OF TEST

2.3.1. Functional tests

Functional tests set out to verify the functions that a system, sub-system or component is required to perform. Functional tests can be described in what are known as work products, such as requirement specifications, use-cases, functional specifications, etc.

These describe "what" the system actually does.

The functional tests are based on the functionalities and their interoperability with specific systems. Functional tests can be carried out at any test level. For example, tests upon components may be based upon component specifications.

Functional tests examine the external behaviour of the software and are therefore mainly set up using black-box techniques.

2.3.2. Non-functional tests

Non-functional tests encompass performance tests, load tests, stress tests, usability tests, maintainability tests, reliability tests (this list is not exhaustive).

Non-functional tests test “how” the system works.

Non-functional tests can be carried out at any test level and they verify the non-functional characteristics of systems. In many cases, they do this by measuring quantifiable information that can be compared with pre-set limits, such as the response times for performance tests.

2.3.3. Tests linked to changes (confirmation tests and regression tests)

Once a defect has been discovered and resolved, the software must be retested in order to confirm that the original defect has been successfully removed. This is known as confirmation testing and is sometimes referred to as retesting.

Remark: The debugging of code is a development activity, not a testing activity.

Regression testing is the retesting of previously-tested program code once changes have been made, in order to discover defects (in the unchanged aspects of the software) caused by changes to the software or the environment.

These tests are always carried out once the environment or the software has changed.

Regression tests may also be carried out at any test level and are applicable to functional, non-functional and structural tests.

Regression tests are carried out often and focus on the most critical functionalities, which makes them a suitable candidate to be carried out automatically.

2.4. THE CLASSIFICATION OF TESTING TECHNIQUES

Testing techniques take the form of formal working methods that enable test cases to be derived from documents, models of the software or code.

We can subdivide the various types of test techniques into 2 groups. These are known as white-box tests and black-box tests.

The white-box tests are based on the program code, the program descriptions or the technical design. Explicit use is made of knowledge relating to the internal structure of the system. These tests are typically carried out by the developers and take the form of a test carried out by the developer in order to demonstrate that a program (or module) or a series of programs (or modules) fulfil the requirements set out in the technical specifications.

The black-box tests are based on the functional specifications and quality requirements. In these tests, the system is examined in the form in which it will operate when it finally comes into use. The software is examined as a “black box”, without any knowledge of the internal implementation or of its internal structure. Typically, a number of test-cases are drawn up, based upon the functional specifications and requirements.

Though both groups of test techniques can be used at all levels, the white-box techniques are mainly used at the lower levels of testing, whilst the black-box techniques are used at the higher levels.

The acceptance tests carried out by Fedict will mainly make use of black-box techniques.

2.5. TESTING PROCESS

The supplier is free to select a testing process that corresponds to his development methodology, on condition that he fulfils the requirements set out in Section 3.

In Section 4, we set out a number of guidelines in this regard, without however imposing a specific process.

2.6. RISK-BASED

Risk-based testing implies that a (product) risk analysis is carried out and that testing activities focus upon those parts or aspects of an application that pose the greatest risk.

It is regarded as good practice for the testing strategy to be based, at least in part, on the outcomes of a risk analysis.

Two possibilities exist in that regard:

- Fedict has added a risk analysis to the contract documents or associated documents.
- Fedict has not added a risk analysis to the contract documents or associated documents.

In the first instance, we would advise the supplier to make use of that analysis and, if necessary, to refine it, once the technical aspects of the system being tested have been more clearly defined.

In the second instance, the supplier may carry out an analysis of its own. In that case, Fedict may provide input regarding the impact of the risks involved.

If the supplier is expecting Fedict to take some sort of action in connection with the risk analysis, the supplier must include in the tender:

- A description of its approach with regard to risk analysis and the role to be played by Fedict in that regard
- An estimate of the time that Fedict will require in order to fulfil that role

2.7. RESPONSIBILITIES

In this section, we will indicate where the responsibilities of the suppler and of Fedict lie.

2.7.1. General

We assume that the supplier is a professional in the field of software development and possesses the necessary specialist knowledge.

If, by virtue of that knowledge, the supplier were to discover shortcomings in the specifications that could lead to the system not being fit-for-purpose, we will assume that the supplier will inform Fedict of this without delay, so that a solution may be found in good time. It is clear that the previous assertion may sometimes extend beyond the contractual stipulations, but it is always in the interests of both parties that a usable system is delivered.

The intention is that the supplier will deliver to Fedict a system that has been fully tested and that is operating correctly. This also includes the delivery of the test documentation, as described in the previous section, and possibly specified in more detail in a contract document or a requirements document.

Fedict will then carry out acceptance tests on the system, in order to establish whether all of the requirements have been fulfilled effectively.

2.7.2. Lower test levels

The test levels involving unit tests (component tests), integration tests and system tests and the corresponding levels, as used by the supplier, shall lie entirely within the responsibility of the supplier.

Fedict will carry out random checks in order to establish whether the work products fulfil the stated requirements.

1.1.1.1. System Integration tests

As far as the system integration tests are concerned, it is possible that the supplier’s systems will need to be integrated with systems from other suppliers or from the government. Collaboration will therefore be possible in that regard.

Fedict will act as a facilitator in that regard, but the final responsibility will lie with the supplier of the system being tested.

Fedict will carry out random checks in order to establish whether the work products fulfil the stated requirements.

2.7.3. Acceptance tests

The test analysis, the test design and the test implementation of a basic set of acceptance tests will be carried out by the supplier.

Fedict will investigate on the basis of cross-checks whether these work products fulfil the stated requirements and may also make provisions for additional test scenarios.

Fedict will carry out the tests and will make use of the test results in order to accept the system or to request corrections, as applicable.

2.8. SEVERITY

The severity of a defect often forms a source of discussion while a system is in the process of being accepted.

[Prescriptive section]

Fedict will apply the following definitions with regard to severities. Any deviation from these must be contractually agreed before the start of a project.

Critical	A test case cannot be completed in any way at all. For example: The submitting of data triggers an error message; the application crashes, etc.
High	A test case can be completed, but the behaviour of the application is not in accordance with the specifications. For example: A Cancel button actually triggers a Submit; a list of data is not reproduced correctly, etc.
Medium	A test case can be completed in accordance with the specifications, but there are faults that can be circumvented by means of manual interventions (without amending the source-code). For example: The incorrect checking of an e-mail field, etc.
Low	*Nice to have* There are cosmetic problems (spelling, lay-out, colours) that have little effect on the functioning of the application.

In this context, "not complying with specifications" must therefore be interpreted as: Is not working as described in the contract document, the requirements document or other specifications approved

by Fedict. In the event that various specifications are not consistent with one another, the interpretation that most favourable to Fedict will be used.

[End of prescriptive section]

3. Requirements for the supplier

3.1. GENERAL

This section contains the work products that must be delivered as standard, together with the associated quality criteria and the requirements to install the system being tested in the various environments.

This section is prescriptive, unless explicitly stated that that is not the case.

The requirements stated here may be different to those contained in the contract documents and associated requirements documents, or in the contracts between the supplier and Fedict. The differences may take the form of more or more stringent requirements, or may involve less stringent requirements being imposed.

This section must be regarded as constituting the test requirements with regard to all aspects of the testing process that are not explicitly transposed into contractually relevant documents.

3.2. CONTENT OF THE WORK PRODUCTS TO BE SUPPLIED

The work products to be supplied must, in principle, contain the details referred to in the templates described in Annex 1. Individual contract documents, requirements documents or contracts may contain additional requirements with regard to content.

If the supplier deviates from these, it will need to document the reason for any such deviation, either in the test plans or in the documents containing the deviation, giving the reason for the deviation.

If the supplier uses its own templates, it must ensure that these contain the necessary information. If so requested by Fedict, the supplier must also provide a document mapping the structure it has used, compared to the ones contained in the attached templates.

The fact that these templates have been provided does NOT imply that Fedict is asking that those templates be used for the actual purpose of generating documents.

[non-prescriptive] It is important to note these are primarily intended as a checklist. In a number of cases, it is certainly advisable to store the requested information in a tool, as opposed to in MS Office documents. [end of non-prescriptive section]

3.3. DELIVERABLES

3.3.1. General documents

3.3.1.1 Tender

The supplier will describe in its tender the testing method and testing process that it will use while fulfilling the contract.

Fedict's expectations in that regard are that this takes the form of a structured and controlled process that is capable of delivering the requested deliverables correctly and in good time.

3.3.1.2 Test plan(s)

The supplier will supply a test plan, in which it will set out the purposes of testing and the various objectives for each level of testing. That plan may consist of a single document or of a master test plan and a level test plan (for each test level). The test plan must be submitted *no later than* before the date on which the first delivery of software takes place. The Master test plan must be maintained, until the final delivery of software takes place.

As a minimum, the plan must contain the elements referred to in the corresponding template attached.

Fedict will check whether the plans submitted fulfil the requirements of this policy, together with any additional requirements or standards imposed in the contract document, the requirements document or any other contractual agreements.

It is in the interests of both parties that the plans be delivered and checked as quickly as possible, so as to avoid any reworking or any delays during the acceptance phase.

3.3.1.3 Master test summary report

The supplier shall include a (master) test summary report in each delivery of software to Fedict.

As a minimum, the plan must contain the elements referred to in the corresponding template attached and must report on each level of testing already carried out.

3.3.2. Unit tests

The supplier shall include a summary of the unit test cases with each delivery of software and shall also provide a number of relevant cases of its own. The other test cases must be made available if and when Fedict requests them.

These may take the form of code or of manual test-cases.

As a minimum, the unit test-cases shall include an automated test suite that can be carried out in any testing environment and in the production environments and that provides a minimum statement coverage of 75%. That coverage must be demonstrated in the version of the software that is delivered. This will take place using a tool that may be made available by Fedict.

A test summary report (or automated test dashboard) will be included in the delivery.

[*non-prescriptive*] We would advise the supplier to apply a system of continual integration or a daily build with integrated test-cases. [end of non-prescriptive section]

Fedict will carry out random checks in order to determine whether the unit tests supplied fulfil the requirements of this policy, together with any additional requirements or standards imposed in the contract document, the requirements document or any other contractual agreements.

[*non-prescriptive*] In the case of complex or critical projects, the statement coverage will be increased, or a switch may be made to a more stringent form of coverage, such as decision or condition average. [end of non-prescriptive section]

Input	• Functional documents • Technical design documents • Code of individual components
Actions	• Testing of individual components
Output (deliverables)	• Test-cases (test-case specification), including an automated test suite • Test results • Test summary report (or dashboard)
Role played by Fedict	• Random checks

3.3.3. Integration tests

With each delivery of software to Fedict, the supplier shall also include the component integration test-cases. These may take the form of code or of manual test-cases.

With each delivery of software to Fedict, the supplier shall also include the system integration test-cases, insofar as the system integration has already been carried out or is applicable. These may take the form of automated or manual test-cases.

The supplier shall demonstrate (for example, by submitting a traceability matrix) that:

- All internal components have been integrated
- Each external interface has been tested in relation the following aspects:
    - Communication via the interface concerned is operational.
    - The interpretation of the data exchanged is the same for both systems.

If this is relevant and as soon as the system integration has taken place, the supplier shall also demonstrate that the end-to-end flow of the application has undergone thorough functional testing (end-to-end tests, based upon the business analysis, chain-testing).

[*non-prescriptive*] We would advise the supplier to include at least part of the component integration tests in a system of continuous integration or a daily build. [end of non-prescriptive section]

Fedict will carry out random checks in order to determine whether the integration tests supplied fulfil the requirements of this policy, together with any additional requirements or standards imposed in the contract document, the requirements document or any other contractual agreements.

Input	• Business analysis (in relation to the system integration tests) • Functional documents • Design documents • Source code of the components to be integrated • Artefacts of the components to be integrated
Actions	• Testing of the interfaces between the components • Testing of the interfaces between the system and its peer-systems • End-to-end testing of functional and non-functional quality attributes
Output	• Test-cases (test case design and test case specification) • Test results • Test summary report • Traceability matrices
Role played by Fedict	• Random checks

3.3.4. System tests

With each delivery of software to Fedict, the supplier shall also include the system test-cases, insofar as these are applicable at that point in time. These may take the form of automated or manual test-cases.

The supplier shall demonstrate (for example, by supplying a traceability matrix) that the full extent of the functionality described in functional documents approved by Fedict has been covered and the risk analysis has been observed effectively.

The supplier shall demonstrate (for example, by supplying a traceability matrix) that the non-functional quality attributes allocated for testing at system test level within the relevant test plan have been covered effectively.

Fedict will carry out random checks in order to determine whether the system tests supplied fulfil the requirements of this policy, together with any additional requirements or standards imposed in the contract document, the requirements document or any other contractual agreements.

Input	• Functional documentation
Actions	• Testing of the functionality of the integrated system • Testing of the non-functional quality attributes of the integrated system
Output (deliverables)	• Test cases (test case design and test case specification) • Test results, including a list of all known outstanding defects • Test summary report • Traceability matrices

Role played by Fedict	• Random checks

3.3.5. Acceptance tests

No later than two weeks before the acceptance tests are implemented, the supplier must submit the acceptance test cases. In principle, these will take the form of black-box tests.

If these are automated, documentation must be submitted, stating in detail how these cases must be carried out. That documentation will then form part of the work products to be delivered.

The supplier shall provide Fedict with the necessary training, demonstrations or explanation with regard to the staging criteria, in such a way that Fedict is able to carry out the acceptance tests.

[non-prescriptive] It is the intention that the supplier provides a full and correct set of acceptance tests. Fedict will check these thoroughly and will mainly carry them out itself. It is possible that Fedict may request additional tests or create some itself, once those checks have been carried out. In order to enable the checking and any corrections to take place, delivery must take place in good time before the time at which the acceptance tests get underway. [end of non-prescriptive section]

The supplier shall demonstrate (for example, by providing a traceability matrix) that the full extent of the functionality as described in the requirement documents and/or the contract document and any other contractual undertakings have been covered by the set of tests provided.

Fedict will carry out random checks in order to determine whether the acceptance tests supplied fulfil the requirements of this policy, together with any additional requirements or standards imposed in the contract document, the requirements document or any other contractual agreements.

Fedict may, depending on its specialist knowledge of the field in question, introduce additional validation tests, unless this is expressly forbidden (in the context of the acceptance) by virtue of contractual agreements. Fedict will make available those additional test cases to the supplier, before the time at which the tests get underway.

[non-prescriptive] The purpose of any additional tests will be to complete the validation, based upon the knowledge of experts in the relevant field. In this case too, it is in the interests of both parties that this takes place as soon as possible. [end of non-prescriptive section]

Fedict will carry out the tests, if necessary with the assistance of the supplier.

Fedict will also draw up a test summary report for this test level.

The supplier will incorporate that level test summary report in the final master test summary report.

In the event that the tests are highly complex or require an advanced knowledge of the internal aspects of the application, it is possible that Fedict may observe the tests while they are being carried out by the supplier. In view of the fact that the requirements provide a black-box

description of the system being tested, that situation will however be extremely exceptional and will require prior approval from Fedict.

3.3.5.1 Incident and defect management during acceptance tests

From the time at which the acceptance tests get underway, the supplier shall ensure that Fedict is in possession of a tool and/or a procedure to record defects and incidents. In the case of non-minor projects, a defect management tool will be preferred (such as JIRA, Mantis, or Bugzilla, for example), as opposed to a document or e-mail-based procedure.
Whatever the situation, the supplier shall ensure that defects and incidents do not go unrecorded and are followed-up until such time as they have been declared closed.

Defects found by Fedict may only be declared closed with the consent of Fedict.

From the time at which the application has been installed in the acceptance environment and has been made available to Fedict there, no defect may be declared resolved without Fedict's consent. This shall apply both in the case of defects found by Fedict or by other parties (for example, the supplier itself or an end-user), as well as in the case of defects that were still outstanding from preceding test levels or stages.

At the start of the acceptance tests in the acceptance environment, the supplier shall explicitly provide a list of all defects still outstanding as part of the test summary report from the preceding test level.

Input	• Business analysis • Specialist knowledge of the field in question • Contractual specifications • Standards and guidelines (government, legal, security), as stated in the contract document or associated documents
Actions	• User's acceptability tests • Operational acceptance tests • Contractual acceptance tests • Conformance tests in relation to the standards and guidelines, as stated in the contract document or associated documents
Output (Deliverables) of the supplier	• Test cases (test case design and test case specification) • Traceability matrices • Master test summary report
Output (Deliverables) of Fedict	• Test results • Level test summary report
Fedict	• Random checks of the test cases and the traceability matrices • Implementation of the test, including recording of the test results and any anomalies • Drawing up of a level test summary report

3.4. STAGING CRITERIA

Staging criteria are the requirements that a release candidate must fulfil, before being installed in a higher environment. In this section, we will describe the general staging criteria, however the precise criteria must be laid down during the project itself, or in the contract document, or at a later stage, in consultation with the supplier.

In the event that no other agreements or requirements have been specified, the criteria listed below shall apply.

It is possible to distinguish between the following application environments:

- Development: this is where the development work takes place and the unit and component integration tests are carried out.
- Test: this is where the system testing takes place
- Acceptance: this is where acceptance testing takes place.
- Production: the live environment of the application.

Whether the system integration tests take place during acceptance, or in the testing environment, shall be determined on a project-by-project basis.

3.4.1. Statement coverage

If, in the staging criteria, it is stated that a functionality or non-functional quality criterion has been covered, what we mean is that:

- One or more tests have been written for that functionality or non-functional quality criterion.
- Those tests have been actually been carried out and if they were unsuccessful, a defect has been recorded.
- The tests provide adequate verification of the functionality or non-functional quality criterion being tested, in the sense that they have been developed using techniques described in a test plan that has been inspected and approved by Fedict.

In view of the fact that in some cases, it is not feasible for reasons of time and budget to provide adequate coverage for 100% of the functionalities or non-functional quality criteria, lower statement coverage rates may be included in the contract document.

Furthermore, the supplier may propose lower statement coverage rates in the test plans, preferably linked to the exposure to risks of the relevant quality criteria.

The test plans shall always state how the statement coverage is being measured.

In Section 4.2, we will provide a non-prescriptive example that illustrates Fedict’s expectations.

The statement coverage must always be explicitly approved by Fedict and it is therefore appropriate to negotiate statement coverage rates with Fedict in good time.

3.4.2. Criteria to proceed to TEST

- Statement coverage of 75% of the code by means of an automated set of unit tests
- 100% of the unit tests must be successful

The IT-provider of the application must report these figures clearly with each delivery. In order to measure the statement coverage, the IT provider must make use of a tool. If so requested by Fedict, the supplier must provide an explicit demonstration of that measurement.

- The test plan describing the unit and component integration tests has been inspected and approved by Fedict.
- The test summary report and/or dashboard relating to the unit and component integration tests has been delivered to Fedict.

3.4.3. Criteria to proceed to ACCEPTANCE

3.4.3.1 Bugs

There must not be any unresolved bugs that block the implementation of the test case and for which no temporary solution (workaround). These are bugs with a severity of “Major” or “Critical”.

There are fewer than 5 unresolved bugs of “medium” severity and these are known and accepted by Fedict (for the installation in the acceptance environment).

There are fewer than 20 unresolved bugs of “low” severity and these are known and accepted by Fedict (for the installation in the acceptance environment).

3.4.3.2 Statement coverage

The full functionality as described in the functional documents approved by Fedict has been covered.

The non-functional quality attributes allocated for testing at system test level in the relevant test plan have been covered.

The automated unit test has been carried out in the test environment using the latest version of the software. The test demonstrated that:

- The statement coverage is still ≥ 75%.
- Each of the unit tests has been successful.

Other

- The test plan describing the system tests and component integration tests has been inspected and approved by Fedict.
- The test summary report relating to the system tests has been delivered to Fedict.
- The supplier has transferred the acceptance tests to Fedict and has provided Fedict with the necessary training, demonstration or explanation in order to enable Fedict to carry out the

acceptance tests. This may possibly take place following installation in the acceptance environment, but before the formal transfer of the application to Fedict for acceptance.

3.4.4. Criteria to proceed to PRODUCTION

3.4.4.1 Bugs

There must not be any unresolved bugs with a severity of “Major” or “Critical”.

There are fewer than 5 unresolved bugs of “medium” severity and these are known and accepted by Fedict (for the installation in the production environment).

There are fewer than 20 unresolved bugs of “low” severity and these are known and accepted by Fedict (for the installation in the production environment).

The supplier has drawn up a plan to resolve the remaining defects and this has been approved by Fedict.

If it has been decided, following consultations between the supplier and Fedict, not to resolve certain defects, this must be explicitly documented and approved by Fedict, such as in the aforementioned plan.

3.4.4.2 Statement coverage

The full functionality and the non-functional quality attributes as described in the contract document and/or the requirement documents have been covered. Fedict may however decide to carry out only some of the specified tests.

The automated unit test has been carried out in the acceptance environment using the latest version of the software. The test demonstrated that:

- The statement coverage is still ≥ 75%.
- Each of the unit tests has been successful.

Other

- The test plan describing the acceptance tests has been inspected and approved by Fedict.
- The test summary report relating to the acceptance tests has been provided and has been incorporated into the master test summary report.
- The supplier has transferred the testware (all of the work products supplied that relate to the testing of the application) to the team that will be required to manage or maintain the application, in so far as those tasks will not be carried out by the supplier itself.

4. Recommendations for the supplier

This section contains non-prescriptive information that may help the supplier to comply with Fedict's requirements, especially if the supplier has little experience in the area of structured testing.

4.1. TEST MATRIX

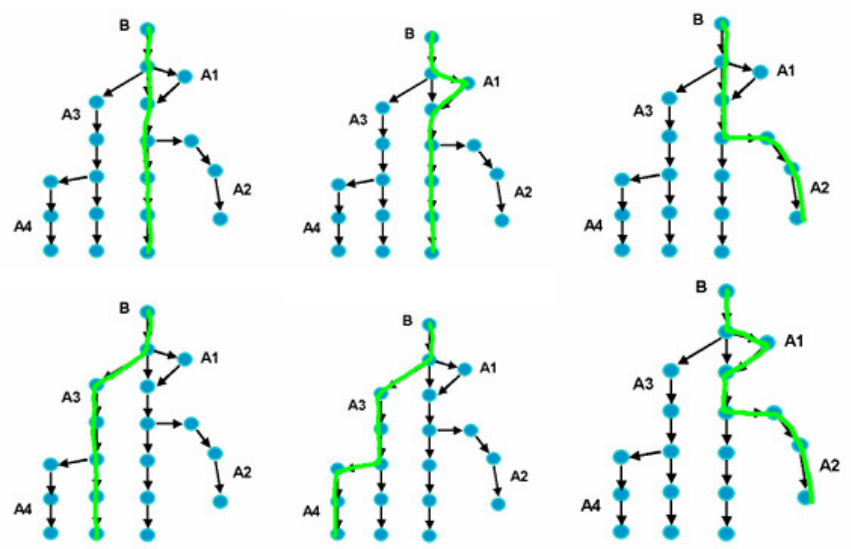
This section illustrates how test-related tasks fit in as part of a developmental cycle and illustrates which tools may be used in that regard.

	BUSINESS ANALYSIS	FUNCTIONAL ANALYSIS	DESIGN	BUILD	CI	TEST	ACCEPTATIE
Unit testen				Test Process: - Test Design - Test Execution Templates: - Test Design Specifications - Test Case Specifications Tools: - JUnit4, Jmock, easyMock - XML-unit Roles&Responsibilities: - Test Designer - Test Automation Engineer - Supplier	Test Process: - Test Execution - Test Reporting - Test Controle Templates: - Test Log - Test Incident Report Tools: - Maven - SonarJ Roles&Responsibilities: - Tester - Supplier		
Integratie testen			Test Process: - Test Design Templates: - Test Design Specifications - Test Case Specifications Tools: - Selenium IDE - Selenium Bromine Roles&Responsibilities: - Test Designer - Test Automation Engineer - Supplier		Test Process: - Test Execution - Test Reporting - Test Controle Templates: - Test Log - Test Incident Report Tools: - Maven Roles&Responsibilities: - Tester - Supplier	Test Process: - Test Execution - Test Reporting - Test Controle Templates: - Test Log - Test Incident Report Tools: - Selenium RC - Selenium Grid Roles&Responsibilities: - Tester - Supplier	
Systeem testen		Test Process: - Test Design Templates: - Test Design Specifications - Test Case Specifications Tools: - Selenium IDE - Selenium Bromine Roles&Responsibilities: - Test Designer - Test Automation Engineer - Supplier				Test Process: - Test Execution - Test Reporting - Test Controle Templates: - Test Log - Test Incident Report Tools: - Selenium RC - Selenium Grid Roles&Responsibilities: - Tester - Supplier	
Acceptatie testen	Test Process: - Test Design Templates: - Test Design Specifications - Test Case Specifications Tools: Roles&Responsibilities: - Test Designer - Test Automation Engineer - Supplier						Test Process: - Test Execution - Test Reporting - Test Controle Templates: - Test Log - Test Incident Report Tools: Roles&Responsibilities: - Tester - Fedict

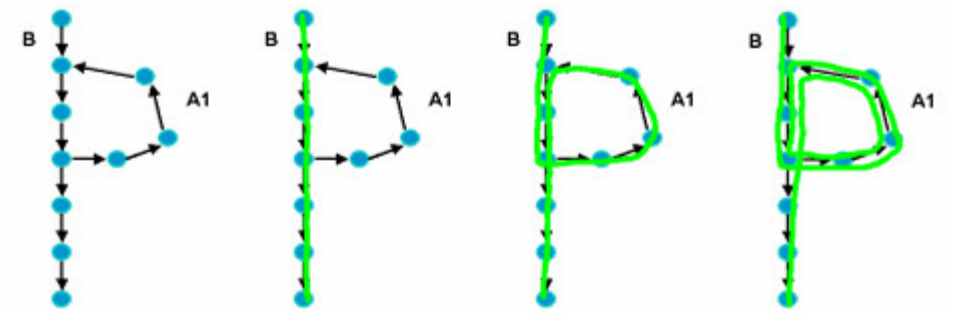
4.2. SCENARIO COVERAGE

This section provides an example of Fedict's expectations with regard to statement coverage, both with regard to the level of detail of the explanation about how coverage is measured, as well as with regard to the statement coverage itself.

A scenario is an instance of a use-case. It demonstrates and describes a specific pathway in the form of a flow of events. In the functional documentation, each use-case is set out by means of an activity diagram. In addition to the basic flow, there are various alternative flows and error flows. To summarise all possible scenarios, it will be necessary to draw all of the possible lines through the graph. The illustration below contains a hypothetical graph representing a use-case with a basic flow B and alternative flows A1, A2, A3 and A4. There are clearly more scenarios than alternative flows, because there is one for A1, another for A2 and a scenario for the combination of the two.



In theory, recurring loops ought to generate an infinite number of scenarios. That is why we choose to do the basic flow once, followed by the loop once and finally the loop for a second time. If the program works for both instances of the loop, we will assume that it will also work if the loop is worked through several times over.



It is not the intention to test all of these flows, but only some of them.

- The basic flow must, at least, be covered by a test scenario.
- After that, there must be a test scenario for each alternative flow.
- Each loop that forms part of the scenario is run through 0 times, once or twice in a test scenario.

1.1. ROLES AND RESPONSIBILITIES

Throughout the testing process, the various activities shall be carried out by individuals with a specific role. The reason for this is that each activity requires specific skills.

In this section, we will describe the most relevant roles. This gives the supplier an idea of the skills and profiles that are necessary in order to carry out testing in a structured and effective manner. It should be understood that one person may undertake several roles.

4.2.1. Test Manager

A test manager will be responsible for the entire testing process. He/she will be responsible for ensuring that the various test phases are carried out correctly, that the various resources are optimally utilised and the correct decisions are taken, based upon the reporting. In practice, he will be responsible for compiling and implementing the Test Plan.

4.2.2. Test Architect

A test architect will be required to formulate an integrated test architecture that will support the entire testing process and will make use of the test infrastructure. In practice, he/she will provide input with regard to the test environments and automation tools etc.

4.2.3. Test Designer

A test designer will be required to document the test cases. He/she will perform that task, based upon the requirements and will convert these into specific test cases that can be carried out by the tester or transformed into automatic scripts.

4.2.4. Tester

A tester will carry out the various test cases, report the results, log any defects and carry out a test coverage analysis. In practice, this will be carried out by a different person for each test level. In most cases, they will be carried out by business analysts, functional analysts, architects and developers or test experts.

4.2.5. Test Automation Engineer

A test automation engineer will be required to automate the test cases described in the test design, using the tools suggested by the test architect. In practice, this will involve the analysts, architects and developers.

4.2.6. Test Methodologist

The test methodologist will be responsible for defining the test methodology and for making sure it is kept up to date. If necessary, the methodology will be adjusted in line with the results of a test process. The test methodologist will evaluate the testing strategy, provide templates and ensure that the methodology is applied in practice.

4.3. TOOLS

Below, we will illustrate a number of tools that can be used to simplify and accelerate the testing process. We would advise suppliers in relation to each of the tools they actually use to take account of the return on investment and to offset the cost of the training and learning time that will be needed.

This section is based on the information in Wikipedia.

4.3.1. Selenium

The selenium suite is made up of several components.

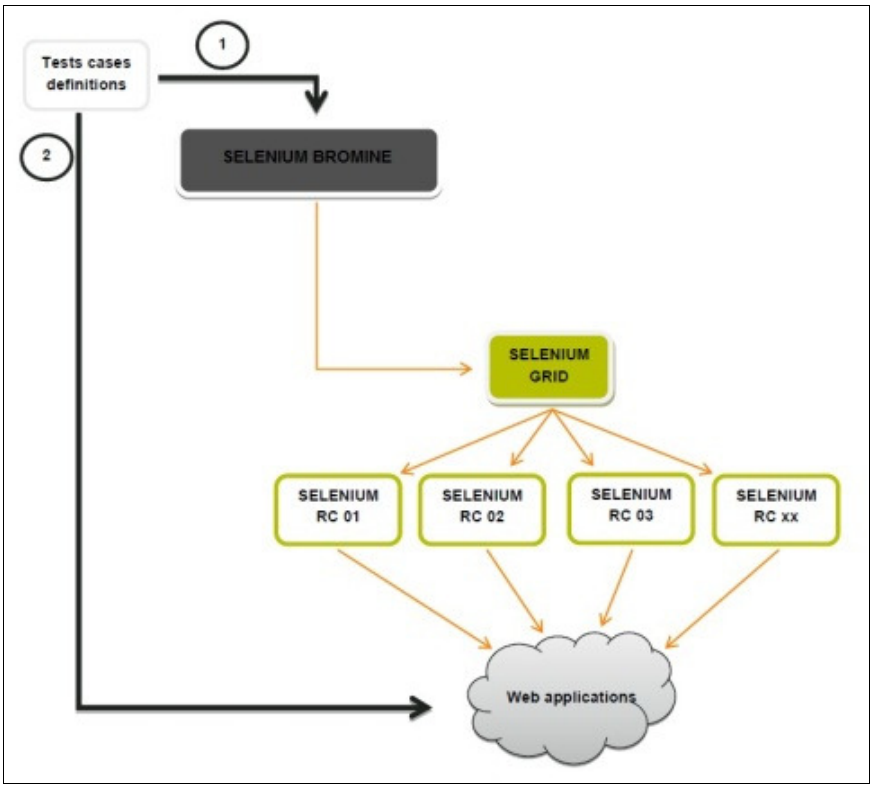
Selenium IDE: an integrated development environment for Selenium scripts. It has been implemented in the form of a Firefox extension and enables test cases to be recorded, edited and debugged.

Selenium RC: a testing tool that enables automatic web application UI tests to be written, in whatever programming language is required, for any http website that makes use of any given JavaScript-enabled browser.

Selenium Grid: a tool that enables multiple test cases to carried out in parallel on several machines and in a heterogeneous environment.

Selenium Bromine: a web-based QA tool for Selenium. This enables Selenium RC tests to be carried out and the results of these to be analysed.

If we bring all of these components together, we will arrive at the following logical architecture, in which 1/ represents the automatic execution of the tests and 2/ the manual execution of the tests:



4.3.2. JUnit

JUnit is a unit test for the Java development language. This is important in the case of a test-driven development. JUnit is an instance of the xUnit architecture for unit test frameworks.

4.3.3. Hudson

Hudson is written in Java and is a continuous integration tool that runs in a servlet container such as Apache Tomcat or GlassFish application server. It supports SCM tools such as CVS, Subversion, Git and Clearcase. It is also capable of executing Apache Ant and Apache Maven-based projects.

4.3.4. Jenkins

Jenkins is an open-source continuous integration tool that is written in Java. Jenkins offers continuous integration services for the development of software that is mainly written in Java. It is a server-based system that operates in a servlet container, such as Apache Tomcat. It supports SCM tools such as CVS, Subversion, Git, Mercurial, Perforce and Clearcase. It is also capable of executing Apache Ant and Apache Maven-based projects.

4.4. TEST PROCESS

In this section, we will illustrate a simple test process, based on IEEE-829 that can be used by the supplier in order to fulfil the requirements stated in this document.

We can distinguish between 3 phases within the test process:

- "Test specification"
- "Test implementation"
- "Test reporting"

We now intend to go through each of those 3 phases. Each phase is made up of its own activities and will generate specific documents.

4.4.1. Test specification

The **Test specification** can be subdivided into 2 parts. On the one hand, we have “Planning and verification”, whilst on the other, we have “Analysis, design and implementation”. Planning and verification forms the activity that verifies the mission, defines the objectives and specifications of the test activities, in order to achieve the objectives. Verification is a continuous process that involves comparing the current demands in relation to the test plan with the status reporting, including the changes to the test plan. Planning and verification may consist of the following activities:

- Specifying the scope and the risks and defining the objectives of the tests
- Determining the strategy
- Taking decisions with regard to what must be tested, what roles apply, how the test activities will take place and how the outcomes of the tests will be implemented.
- Planning of the test design activities
- Planning of implementation and evaluation

During the analysis, design and implementation, the test objectives will be transformed into specific test conditions and test cases. This may involve the following activities:

- A review of the test base (such as requirements, architecture, design, interfaces)
- Evaluation of the testability of the test base and the test objects
- Identification and prioritisation of the test conditions
- Designing of the test cases
- Identifying the necessary test data in order to support the test cases
- Designing the test environment and identifying the infrastructure and tools

4.4.2. Implementation of tests

The **implementation of tests** forms the phase in which the test procedures and scripts are implemented and the results of these (including incidents) are logged. This may involve the following activities:

- Implementation of the tests, manually or by means of a tool

- Logging the results of the implementation
- Comparing the results with the result that had been predicted
- Reporting any setbacks in the form of incidents and analysing these in order to identify the cause
- Repeating the tests as a result of the incidents identified

4.4.3. Test Reporting

Test Reporting forms the phase in which the implementation of the tests is compared with the objectives. This may involve the following tasks:

- Checking of the test logs and the list of defects and comparing these to the exit criteria specified in the test plan
- Investigating whether additional tests are required
- Writing a summary report

4.5. TEST DOCUMENTATION

At all times while the test process is taking place, various documents must be created. Below, we will provide a description of the documents that will be required. A template is available for each document and this can be used as a guide.

4.5.1. Test plan

The test plan consists of a description of how the tests will be managed, planned and implemented. A test plan is a document that is amended throughout the entire test cycle and which describes what must take place, what quality standard applies, what resources will be used, what timescale will be followed, what the risks are and how these can be resolved.

In a test plan, we would expect to see the following as a minimum:

- S *Scope*: what must be tested and what does not need to be tested
- P *People*: training, responsibilities and planning
- A *Approach*: the approach that will be followed when carrying out the tests
- C *Criteria*: entry/exit criteria and suspension/resumption criteria
- E *Environment*: the requirements in relation to the test environment
- D *Deliverables*: what must be delivered as part of the test process
- I *Incidentals*: introduction, identification and approval
- R *Risks*: risks that may occur
- T *Tasks*: test activities

It is possible to define a Master Test Plan that applies to the entire scope of the testing, however one may prefer to define a test plan for each individual test level, such as a test plan for unit testing, system testing etc. This is entirely a matter of choice.

It goes without saying that irrespective of the choice referred to above, it must be clearly stated what exactly is being tested and at what test level.

4.5.2. Test design specification

The test design specification document describes in a logical manner what must be tested in accordance with the requirements. Those requirements will then be converted into specific test conditions.

That document does not yet contain any test date or any description of how the test conditions can actually be tested. It describes the requirements that apply to that test data and solely forms a link between the requirements and the test cases.

4.5.3. Test case specification

The test case specifications convert the test conditions into specific test cases by supplementing them with test data, pre-conditions and predicted results. The test cases can be created, once the test design is complete. A single test case may consist of one or several test conditions. A test case must consist of:

- The test data that is needed in order to carry out the test. This is made up of a combination of input data, application data and system data
- The predicted result and outcome
- The pre-conditions that determine the starting point of each test
- The staged plan that must be followed in order to carry out the test

4.5.4. Test log

The test log records the details regarding which test cases have been carried out and their results. On the one hand, this may mean that a test case has been successful, which means that he predicted result and the actual result are the same. On the other hand, it may mean that a test case has failed, as a discrepancy has been identified. This will result in the creation of a Test Incident Report, in which the details of the discrepancy will be described. A test log will follow the progress of the tests and will also provide information regarding the cause of the incidents.

4.5.5. Test Incident Report

The Test Incident Report documents each incident that requires more detailed examination. The incidents arise from the Test Log. The Test Incident Report must contain all the details of the incident, such as the predicted and actual result, when it failed and any other information that may help resolve the incident. This report may also possibly contain details of the impact of the testing of the incident.

4.5.6. Test Summary Report

The Test Summary Report contains the results of the test activities and the evaluation of the results obtained. It provides an overall view of the test activities and the quality of the software.

5. Annexes

5.1. ANNEX 1 – TEMPLATES

This policy is accompanied by a number of templates. These are supplied in the form of separate files and have been drawn up in the English language. A list of the templates can be found in Section 5.1.2.

5.1.1. Use of the templates

The intention is for the templates to be used as a check-list, in order to verify that the documents supplied contain correct and sufficient information.

The lighter text that appears in blue in the templates contains instructions (guidance) and must be interpreted pragmatically. Redundancy and excessive wordiness must be avoided in all documents.

If an entire section does not apply, it is sufficient to state "not applicable" or N.A., followed by a brief explanation (1 line or less).

It is permitted, but not compulsory, to use these templates in order to draw up the relevant documents. In many cases (such as incident reports, test case specifications and test log) it is probably more efficient to use adequate tools, rather than handle that information in document form.

5.1.2. List of templates

- Test Case Specification Template
- Test Design Specification Template
- Test Incident Report Template
- Test Log Template
- Test Summary Report Template
- Testplan Template