E-vote 2011

SSA-U Appendix 5 Testing and Approval

Project: E-vote 2011

Change log

Version	Date	Author	Description/changes
0.1	26.10.09		First version

CONTENT

1. Introduction

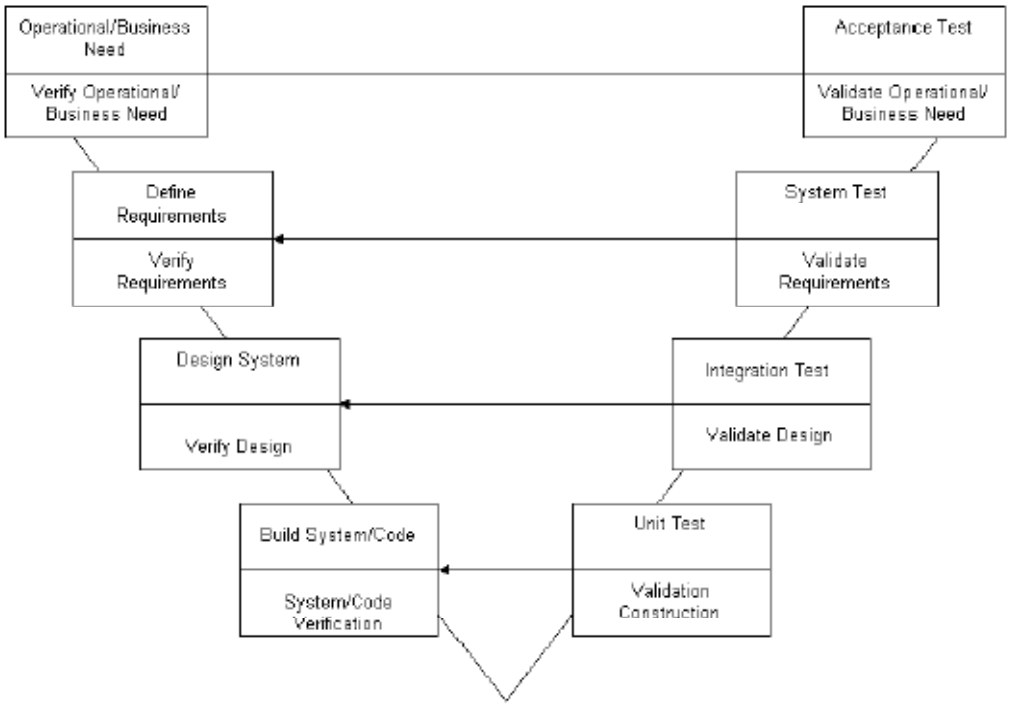
This document elaborates Requirement ST4.4.

The document describes the testing strategy to be applied in this project. It covers the usability testing, unit testing, accessibility testing, system testing, regression testing, volume testing, performance testing and security testing. Acceptance testing is thoroughly described in section 2.5 of the SSA-U Software Development Agreement.

Indra will set up the testing environment to carry out all the tests described below. Indra will also set up the necessary infrastructure to carry out the Customer Acceptance Tests. All testing activities will be documented as agreed with the Customer.

2. Testing Process Overview

The testing process is composed of a set of tasks, related to the development tasks. The usual way of showing these relationships is through the following V-Model:



In this V-Model, the left side shows the development tasks, while the right side displays the corresponding testing tasks.

This V-Model will be used throughout this document in order to describe testing activities and deliverables in detail.

3. Unit Testing

3.1. Purpose

The main purpose of Unit Testing is to validate the construction of the software components/modules.

3.2. Strategy

Common strategies used for unit testing cover Black-box testing and White-box testing.

In Black-box testing, design of the test case is based in exercising the interfaces provided by the module or component, without knowing how the component is built or designed.

In White-box testing, design of the test case is based in exercising the internal mechanisms on which the module or component under test is based. Knowledge of the internal design and implementation of the component is a must.

Independently of the chosen strategy (White-box or Black-box testing), from a operational point of view, unit testing is performed designing a data set that is input to the component/module and its outputs are compared with the expected outputs; if expected and real outputs match, the test-case passes; otherwise, the test fails.

Unit testing is closely related to the building process (its purpose it's validate construction), so it usually is automated in the software building process (compile-test automated cycle), as in a continuous integration environment.

3.3. Tools

Several tools are available for helping in the execution of unit tests, depending on the programming language and technology used in the software component implementation. For example, if Java programming is used in the implementation of the component, these are some of the tools available for unit testing:

- JUnit: This tool allows defining test-cases as Java classes that can be used to invoke the methods provided for the component (java library) under test and make assertions on the values returned by those method invocations.
- DBUnit: This tool is a particularization of the JUnit tool when persistence (database) is involved in testing; this tools allows pre-loading the database with known data in order to make assertions on the test-case, and cleaning the database after each test-case (Java class) is run.
- TestNG: is an alternative functionally equivalent to JUnit

These tools can be integrated in a continuous integration environment. This continuous integration environment can be implemented, for example, using the following tools:

- CruiseControl: This tool is responsible for monitoring the activity (commits) on the software repository (CVS, SVN, …). When certain conditions are met, an automated build is triggered by this tool.
- Maven: This tool is responsible for running the particular building process, including execution of the testing classes (JUnit, …)

All the tools named in this section are freely available, and are the product of open source projects.

4. Integration Testing

4.1. Purpose

The main purpose of the integration testing is validation of the design and architecture chosen to implement the software solution.

4.2. Strategy

Integration testing is mainly focused on Black-box testing, as the focus in this kind of testing is on assembly of the inter-dependent modules/components in which the architecture of the solution is based, in order to check that the whole subsystem complies with the functional and performance requirements.

4.3. Tools

Several tools are available, depending on the technology used in the solution. For example, for web based systems, the following tools can be used:

- JMeter: This is a tool aimed at performance testing. Several technologies can be tested with this tools: web, web-services, database, … The tool provides facilities for load generation, SUT (system under test) monitoring and results reporting.
- Canoo Web Test: This is tool aimed at functional testing. Basically, invokes web pages through HTTP and checks the expected result with the real result. Some automation of the functional testing process can be achieved with this tool, making it very interesting for regression testing.

All the tools named in this section are freely available, and some are the product of open source projects.

5. System Testing

5.1. Purpose

The purpose of system testing is requirements validation, as functional as non-functional.

5.2. Strategy

System testing is mainly twofold:

- End-to-end testing: The use-cases describing the system functionality are taken as a base for the test-case specification. All the layers of the system (user interface, business logic, database, external systems, …) need to be available (built, tested and deployed) to run these kind of test-cases.
- Performance testing: This kind of testing is aimed at validating system behavior (in terms of response time, maximum concurrency, throughput, …) under several load conditions (test scenarios). In this kind of testing, load characterization if a central concept: load characterization means specifying in the test-case the load contribution for every user type (role) or external system (for example, in terms of hits or invocations per time-unit) present in the test scenario, and the way its load changes during the test execution (initial ramp until steady load is reached, steady load level and duration, and final ramp).

  Typically, various kinds of performance testing can be designed; for example:
  - Load testing, aimed at understanding system behavior under a expected load. Used mainly in bottleneck detection.

- Stress testing, aimed at 'breaking' the system; the load offered to the system is well over the expected/specified by the requirements and helps determine the administrators to know the maximum load the system will be able to handle before serious problems arise.
- Endurance/Soak testing, aimed at ensuring the system will be able to sustain the continuous expected load.

5.3. Tools

In this type of testing, the same tools used for integration testing are usually available. Automation of performance tests can be done, usually, through scripting in order to achieve the loads specified in the test-case scenario.

6. Accessibility and Usability Testing

All analysis and solutions referring to the implementation of accessibility will be documented in detail within the group of previously listed deliverables.

Among these descriptions, the section that stands out is dedicated to defining the accessibility tests in all aspects, from the procedures to the tools used to conclude with the analysis of the obtained results. If applicable, the corrective actions will be documented to solve the possible errors detected in the tests.

Specifically, in the correction tests of accessibility we will have the following tasks:

- **Definition of Tests:** once there is a system developed with the objective of fulfilling accessibility rules, the experts in this subject will define in detail a plan of tests based on automatic activities, expert revisions and laboratories with disabled users.

Likewise, the metrics will be defined that will generate the subsequent reports and analysis after executing the tests.

- Execution of Tests: after defining the tests in detail, they will be taken with the participation of accessibility experts. In this way, we can distinguish 3 different test types:

- Automatic Tests: by using automatic accessibility verification tools, such as TAWDIS (Spain) or AChecker (Norway), in a way that the possible shortfalls can be identified that could be found by a computer program.

  This software keeps improving its degree of precision, but it still requires the revision of an accessibility specialist before ruling out the correct testing of some non-testable standards by a computer program.

- Expert Tests: Expert product validations are conducted precisely for those standards that cannot be 100% checked by the automatic validation programs. In this regard, all the specifications of use and system coding are revised in order to verify their complete correction according to all the recommendations for an accessible product. These validations are done by using the current Norwegian rules as reference, and the internationally accepted ones as standard WAI rules, in their specifications 1.0 and 2.0.

- Laboratories for Disabled Users: in which, beyond the expert verifications, the system will be subjected to the tests of real users that present different types of disabilities. This type of test is usually definitive for determining the real degree of the developments' accessibility, since fundamental aspects are shown in these that go beyond the good standards generally covered by the rules.

-Analysis of Results: finally the analytical tasks are done that correspond to the results from the previous tests, in a way that reports can be issued about the accessibility status, identifying the strengths, shortfalls and the necessary corrective actions to carry out.

With these results, an error correction phase is carried out, after which the product is subjected to the same complete tests again, as done in the first iteration.

6.1. User's Tests

In a similar way to that which was explained with the accessibility tests, the execution of tests on users is proposed, which would allow us to verify that the proposal of the prototype and design phase works as expected with real users.

In this case, the following test types are defined:

- **Expert Analysis**: in which the product is subjected to the expert analysis from the usability point of view, to identify the good practices or the cases that are not adequately resolved, so that they facilitate the use of the application.

- **Heuristic Analysis:** the heuristic evaluation or evaluation by criteria is conducted by specialized evaluators based on principles established by the discipline of the HCI. The criteria recommended by authorities in the subject such as Jacob Nielsen (Visibility of the System status, Adaptation between the system and the real world, Control and freedom of the user, Consistency and standards, Preventing errors, Recognizing is better than remembering, Flexibility and efficiency of use, Minimalist aesthetics and design, Helping users recognize, diagnose and solve errors, Help and documentation), is used as the basis for those that add their own criteria based on the experience of our experts.

- **User Laboratories:** finally, in a similar way as the case of accessibility, the product will be subjected to the tests with real objective users, which will prove the reality of the application's use.

  The user conduct, reactions and opinions can become known through the laboratory tests. This way, the detected problems will be analyzed in depth, as well as the expectations and suggestions of these users.

  The **results** include ratios of effectiveness, efficiency, satisfaction, comments, tendencies, opinions, etc. The test results come with recommendations for improvement.

This entire procedure will be properly defined and the results will be recorded. In the same regard, the obtained results will be analyzed, and their conclusions will become feedback in a cyclical process of improvement for the user interfaces from all possible points of view.

7. Acceptance Testing

As described in section 2.5 of the SSA-U Software Development Agreement.

8. Bug Reporting and Change Management

Bug reporting and Change management are the two basic mechanisms to keep the errors detected on the system (through failing tests) identified and controlled.

8.1. Bug Reporting

Bug reporting is usually supported by an application providing functions similar to the following:

- Bug registration: For every bug detected, information describing the bug, detection date, screen captures or log files, or any other information that can be useful for bug-fixing is entered in the application by the reporting user.
- Bug assignment: Allowed software development personnel can receive the new bug registrations and assign them to developers in order to be solved; this change in the status of the bug is visible to the reporting users.
- Bug solving: When the software system is modified by the developer in order to solve the bug, a new change in the state of the bug is visible to the reporting users.
- Bug delivery: When the modification to the software is scheduled for delivery, a new change in the bug state is visible to the reporting users.
- Bug correction/rejection: When the modification to the software is delivered (deployed) as part of a system upgrade, and the reporting user can check the bug is really solved, the result of that check (bug corrected/ not-corrected) can be recorded in the system, thus closing the bug report or starting a new correction cycle.

Depending on the complexity of the bug life-cycle to implement, several tools can be used. Typical, freely available, open source tools include Bugzilla, JTrac and Scarab.

8.2. Change Management

Change management is usually supported by a software repository supporting versioning of the different software artifacts (source code files, configuration files, user documents, models, etc.). Every time a software artifact is changed and committed to the repository, the artifact is stored in the repository with a new version number. The different versions of every artifact can be inspected and compared. A set of artifacts, each in a given version, can be tagged as a release, in order to be exported from the repository as a unit when needed.

One of the reasons a software artifact is changed is bug correction. A set of bug corrections (plus, eventually, new functionality) can be tagged as a release to be delivered and deployed (at that moment, users will be able to check bug correction and report results of the checking to the bug reporting application)