UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

# Implementation of an integrated testing system for ITSAT

**Author:** Hongzhi Zhu
**Date:** January 15, 2014
**Supervisor:** Xavier Martoll Bofill
**Director:** Antonio Moreno Giménez
**Degree:** Computer Engineering
**Center:** Barcelona School of Informatics
**University:** Polytechnic University of Catalunya

# Index

# 1. Introduction

## 1.1 Context

As technology evolves, computers are becoming increasingly important in the workplace. However, the installation of a computer for a specific work environment is also challenging and labor-intensive. Although readme, help and feedback documents are available, computers are often configured differently if installations are carried out by different people, and multiple installations are generally too much work for a single engineer.

These issues could be addressed by developing a tool that can facilitate the process of installation and guarantee the exact same configuration for all computers. Such a tool must be easy to use and also allow administrators to interact not only with all the computers in the same work environment but also with certain computers under specific circumstances.

## 1.2 Introduction to ITSAT

- What is its purpose?

ITSAT (IT System Administration Tool) is a management tool for IT environments, designed for system administrators to directly increase their work efficiency by making the administration of multiple and diverse computers installed in the same working area much easier. This tool allows a single person to carry out the installation and management of many computers with less effort and in less time, because the process is implemented simultaneously for all the computers. The configurations are also guaranteed to be the same because there is a main computer controlling installation and maintenance.

- Other advantages and benefits

Having a tool which allows interactions with multiple computers can increase the quality of work carried out by a system administrator:

  - **Accessibility**: The tool provides access without the need to remember the username and password for each computer, as well as interactive connections to different computers from a website with no need for an external client to access them at the other end.
  - **Execution of an iterative process**: The tool allows the rapid execution of commands on multiple computers without direct access to each of them. These commands can also be programmed according to a certain schedule, allowing repetitive or periodic commands to be issued at any time.

- Operation modules

The various functions of ITSAT are provided in the form of modules, so that each module controls a specific area or functionality of each computer. The following main modules are currently available:

  - **Audit Module**: A module to view records and past events. This module allows records of previously activities to be viewed in the form of lists or graphically in a timeline. It also comes with the option to filter the result by computer, work

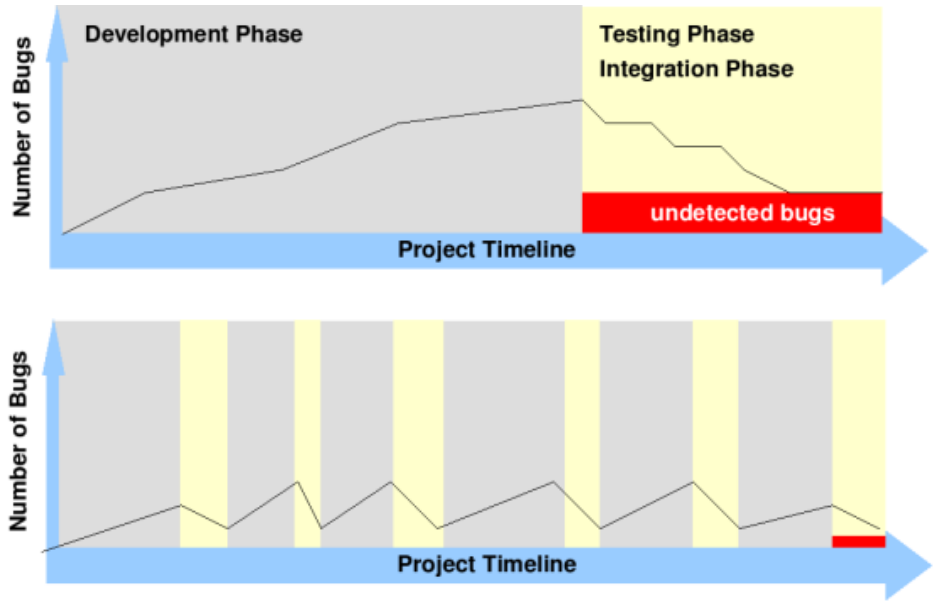
environment, module or user, and also the possibility to limit the timespan of the activities.

- **Basic Operation Module**: A module which allows the execution of commands and services. This module provides an alternative way to execute commands and services, filtering the execution by work environment, production environment, computer or operating system, and in case of services, also filtering by services and instances. This module also shows a list of all the commands and services recorded during execution and whether or not this was successful.
- **Management Module**: A module to manage systems, such as computers and users. This module provides functions such as adding new computers and users, modifying their information and also deleting them, showing general and basic information about ITSAT, configuring different users and computers, privileges and permissions.
- **Remote Connection Proxy Module**: A module which allows connections to different computers. By connecting to a computer, the sessions, activities and logs of all users can be viewed.
- **Task Schedule Module**: A module that commands to be executed as defined by a schedule, e.g. on a specific time and date or periodically.
- **Top Secret Identification Module**: A module that allows users to create, modify and delete credentials from the ITSAT system.

| Module ICON | Module ID | Version | Label | Enabled |
|---|---|---|---|---|
|  | audm | 0.4 | Audit Module | ✓ |
|  | bopm | 0.4 | Basic Operations Module | ✓ |
|  | mngm | 0.8 | Management Module | ✓ |
|  | rcpm | 1.2 | Remote Proxy Control | ✓ |
|  | tsim | 0.8 | Top Secret Identification Module | ✓ |
|  | tsm | 0.2 | Task Scheduler Module | ✓ |

## 1.3 Motivation

ITSAT is one of the pioneers in the area of interactive connections and administration between computers, but it is still under development and needs more features and functions directly related to the level of maintenance of the tool itself. Changes and modifications have to be controlled and maintained continuously to prevent unexpected bugs and the incorrect behavior of previously defined functions. This is a challenging task due to the complexity of ITSAT and a robust testing system is required.



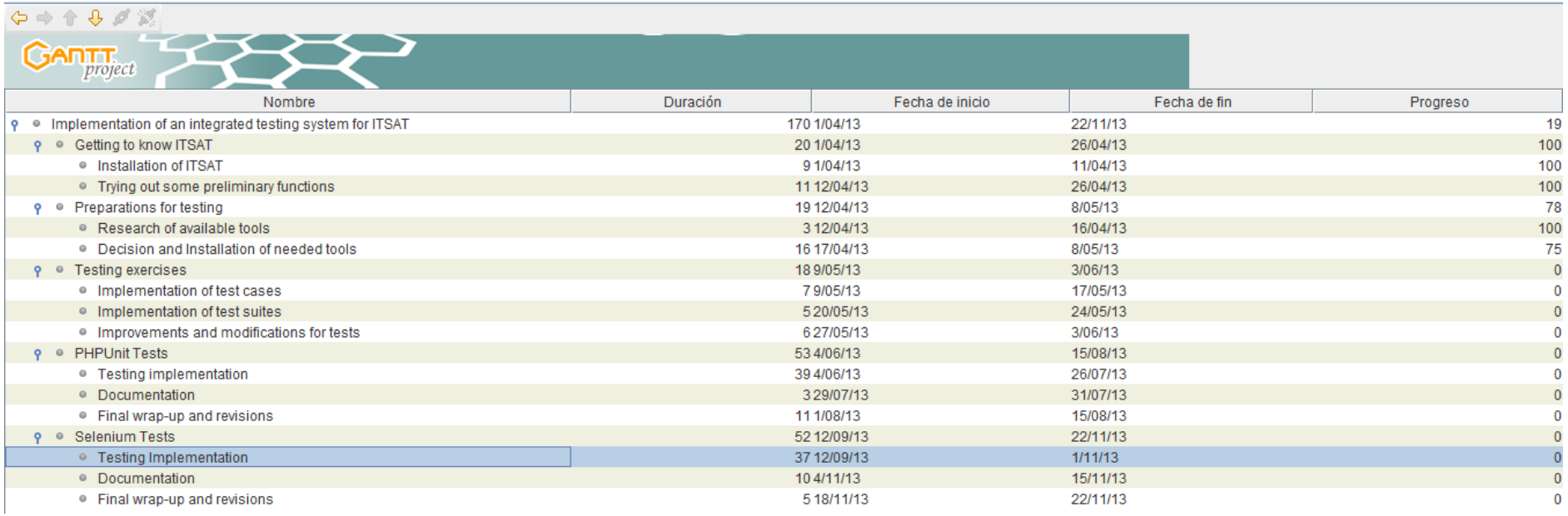

*Comparison between (top) a common process without continuous integration testing and (bottom) a common process with continuous integration testing*

## 1.4 Objectives of the project

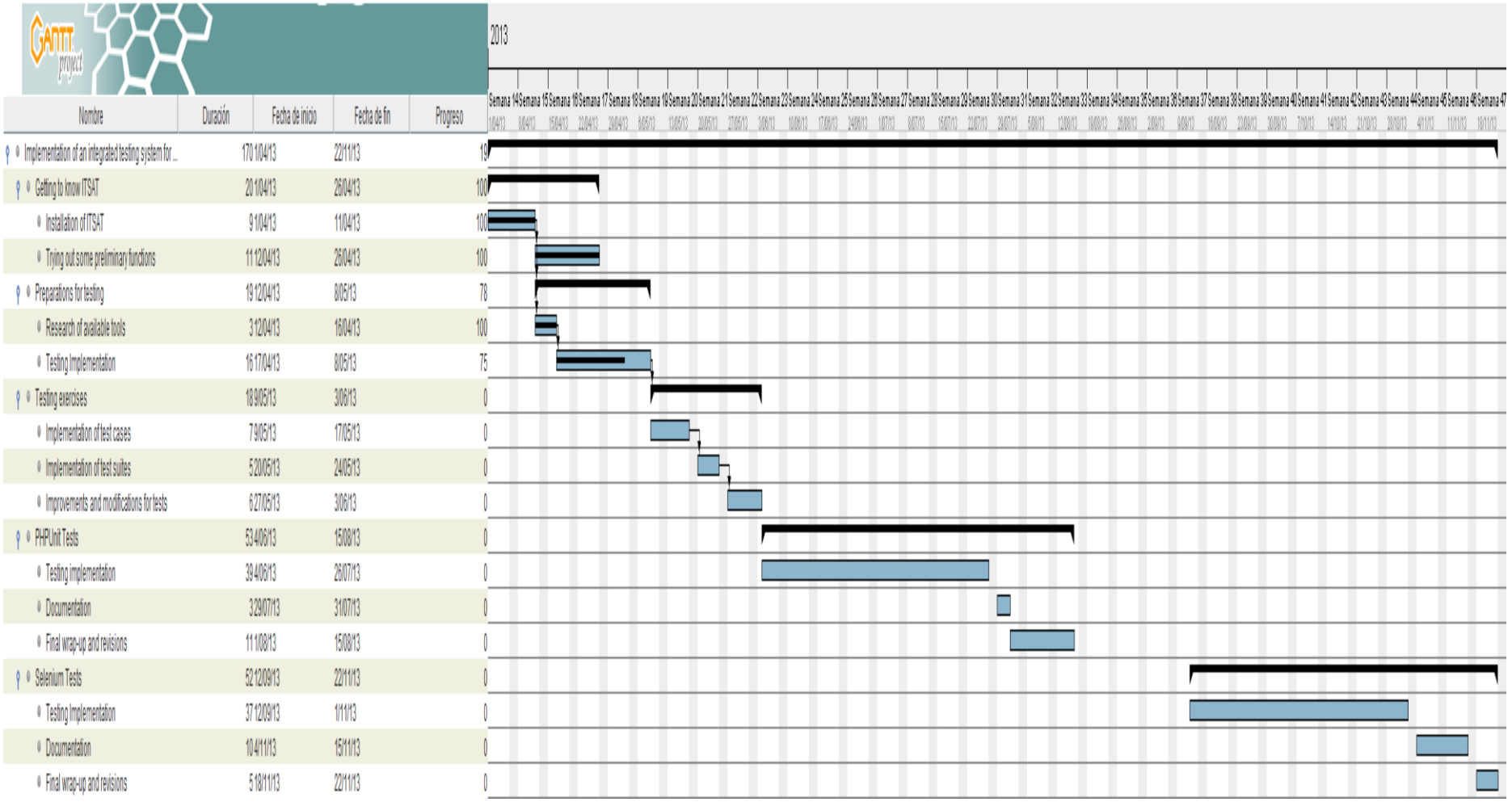
1. **Investigate and select the most appropriate testing systems or applications that can help to maintain ITSAT.**
   The optimal testing system will depend on the correct selection of test parameters, and the choice of programs and applications will depend on how it is tested.
2. **Implementation and maintenance of functional testing**
   This is the main phase of the project, involving the implementation and maintenance of the testing system.
3. **Documentation**
   Once the testing system has been completed, accurate documentation will enable future modifications or further testing.
4. **Revision**
   The final phase of the project is to revise all the tests and check that everything is working perfectly.

## 1.5 Initial Planning

The tasks and the planning of the duration of the project are represented in the following Gantt chart.



| Nombre | Duración | Fecha de inicio | Fecha de fin | Progreso |
|---|---|---|---|---|
| Implementation of an integrated testing system for ITSAT | 170 | 1/04/13 | 22/11/13 | 19 |
| Getting to know ITSAT | 20 | 1/04/13 | 26/04/13 | 100 |
| Installation of ITSAT | 9 | 1/04/13 | 11/04/13 | 100 |
| Trying out some preliminary functions | 11 | 12/04/13 | 26/04/13 | 100 |
| Preparations for testing | 19 | 12/04/13 | 8/05/13 | 78 |
| Research of available tools | 3 | 12/04/13 | 16/04/13 | 100 |
| Decision and Installation of needed tools | 16 | 17/04/13 | 8/05/13 | 75 |
| Testing exercises | 18 | 9/05/13 | 3/06/13 | 0 |
| Implementation of test cases | 7 | 9/05/13 | 17/05/13 | 0 |
| Implementation of test suites | 5 | 20/05/13 | 24/05/13 | 0 |
| Improvements and modifications for tests | 6 | 27/05/13 | 3/06/13 | 0 |
| PHPUnit Tests | 53 | 4/06/13 | 15/08/13 | 0 |
| Testing implementation | 39 | 4/06/13 | 26/07/13 | 0 |
| Documentation | 3 | 29/07/13 | 31/07/13 | 0 |
| Final wrap-up and revisions | 11 | 1/08/13 | 15/08/13 | 0 |
| Selenium Tests | 52 | 12/09/13 | 22/11/13 | 0 |
| Testing Implementation | 37 | 12/09/13 | 1/11/13 | 0 |
| Documentation | 10 | 4/11/13 | 15/11/13 | 0 |
| Final wrap-up and revisions | 5 | 18/11/13 | 22/11/13 | 0 |


GANTT project
2013
Nombre
Duración
Fecha de inicio
Fecha de fin
Progreso
Implementation of an integrated testing system for ... 170 1/04/13 22/11/13 19
Getting to know ITSAT 20 1/04/13 26/04/13 100
Installation of ITSAT 9 1/04/13 11/04/13 100
Trying out some preliminary functions 11 12/04/13 26/04/13 100
Preparations for testing 19 12/04/13 8/05/13 78
Research of available tools 3 12/04/13 16/04/13 100
Testing Implementation 16 17/04/13 8/05/13 75
Testing exercises 18 9/05/13 30/8/13 0
Implementation of test cases 7 9/05/13 17/05/13 0
Implementation of test suites 5 20/05/13 24/05/13 0
Improvements and modifications for tests 6 27/05/13 30/8/13 0
PHPUnit Tests 53 4/06/13 15/08/13 0
Testing implementation 39 4/06/13 26/07/13 0
Documentation 3 29/07/13 31/07/13 0
Final wrap-up and revisions 11 1/08/13 15/08/13 0
Selenium Tests 52 12/09/13 22/11/13 0
Testing Implementation 37 12/09/13 1/11/13 0
Documentation 10 4/11/13 15/11/13 0
Final wrap-up and revisions 5 18/11/13 22/11/13 0

# 2. Testing systems and applications

## 2.1 Characteristics

In order to choose the most suitable testing systems for ITSAT, it is necessary to carry out a thorough analysis of its development and structure.

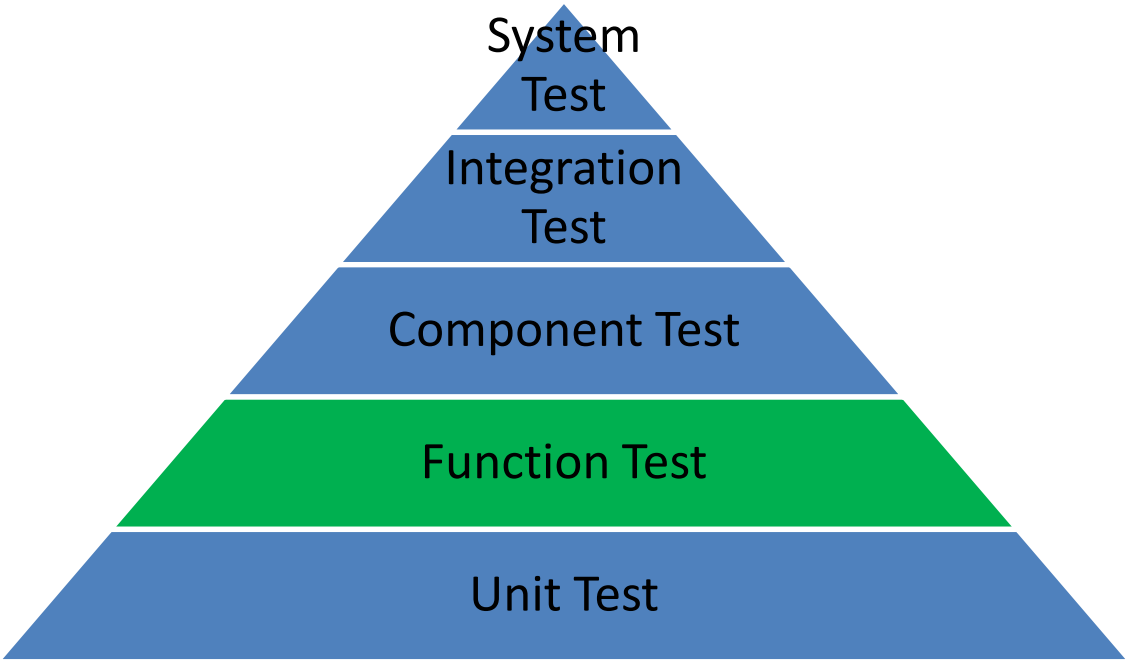
Like most projects under development, ITSAT is stationed in an online repository, where each developer can access the program and make changes and modifications, also commonly known as continuous integration. ITSAT includes a variety of functions that interact with other functions, so modifications to one may cause unexpected negative side-effects on others. Another important characteristic is that ITSAT is not only a code-based project, but also has a web interface, and there is a direct bidirectional relationship between them.

## 2.2 Testing Focus

There are many types of testing and therefore the most appropriate programs, applications and methods must be selected.

- **Unit Test.** The smallest amount of testable code. Often a single method/function, excluding the use of other methods or classes. It is rapid, allowing thousands of unit tests to be run in a few seconds. Unit tests never use:
  - Databases
  - An application server
  - File/network I/O or file system
  - Another application
  - The console (e.g. System out, System err)
  - Logging
- **Function Test.** Larger than a unit but smaller than a full component test. Usually involves several methods/functions/classes working together. Hundreds of function tests can take hours to run.
- **Component Test.** Running one component alone. A component consists of various combined function tests.
- **Integration Test.** Testing two or more components simultaneously.
- **System Test.** All components tested together.

Testing should typically begin with unit tests but ITSAT is such a complex tool that this approach is almost impossible. Because ITSAT is a tool with multiple functionalities, it is much more fitting to focus primarily on function tests. If ITSAT is the whole system, then each module introduced in section 1.2 can be thought of as a component, and each component will have its own functional tests. This does not mean that unit testing can be skipped. Unit testing is just as important as function testing, but function tests are a more realistic starting point. Once the function tests are created, they can be broken down into various unit tests.




*Different types of testing and their relationships. ITSAT currently focuses on function tests.*

## 2.3 Choice of testing programs and applications

From the previous analysis of ITSAT, manual testing each time a new change is applied to the repository is likely to be inefficient and impossible to implement over long periods of time because of the tendency of humans to tire of repetitive tasks and make mistakes. A testing tool with the following characteristics is therefore required to maintain a stable repository:

- **Rapid:** Several developers will be integrating new modifications to the project, so the testing system must be fast enough to complete tests of the previous integration before the next is introduced.
- **Accurate:** The testing must be accurate enough to guarantee that all aspects of ITSAT are functioning properly after each modification.
- **Automatic:** The testing must also be implemented automatically when a new modification is introduced so it can report errors as soon as possible. Modification not only refers to changes added by developers but also actualizations of software that may affect the proper function of ITSAT.
- **Maintainable:** Testing is a support for our project, so it would be illogical if the testing system were more difficult to maintain than the project itself. ITSAT will also incorporate new functions and modify its original functions, so the testing must be flexible and adjustable over time.

Although most of the testing tools already satisfy the properties listed above, we must also take into account other aspects of the tools themselves in order to choose those most suitable for ITSAT.

### 2.3.1 PhpUnderControl


Because most ITSAT codes are written in PHP, it is necessary to find a tool which supports both PHP and continuous integration.

PhpUnderControl is an add-on application for the continuous integration tool CruiseControl, which integrates some of the best PHP development tools. It also provides testing and software metrics such as PHPUnit, and documentation such as PhpDocumentor.

### 2.3.2 Xinc

Xinc (Xinc Is Not CruiseControl) is another tool for testing PHP and is an alternative to phpUnderControl. It is a specific tool for development in PHP, written in PHP and, for example, using Phing (the Ant of PHP) instead of Ant itself. It also supports XML. One of its advantages is that it does not require Java and is developed and orientated towards PHP coding programs. The Xinc developer community is much smaller than that of CruiseControl, but more specific to PHP.

### 2.3.3 Selenium

ITSAT is also a browser-based tool, so a testing framework for interfaces on browsers is also needed. Selenium is a portable software testing framework for web applications. It provides a record/playback tool for authoring tests without learning a test scripting language (Selenium IDE), and also a test domain-specific language (Selenese) to write tests in a number of popular programing languages, including Java, C#, Groovy, Perl, PHP, Python and Ruby. The tests can then be run against most modern web browsers. Selenium deploys on Windows, Linux and Macintosh platforms.

### 2.3.4 Watir

Watir (Web Application Testing In Ruby) is an open-source (BSD) family of Ruby libraries for automating web browsers. It drives Internet Explorer, Firefox, Chrome, Opera and Safari. Watir is one of the most widely-used web browser testing tools, favored by companies such as Facebook, Yahoo!, HP and Oracle. Some of the advantages of Watir include:

- It is a free Open Source tool.
- There is a highly active and growing community behind it.
- It uses Ruby, a fully-featured modern scripting language, rather than a proprietary vendor script.
- It supports web applications regardless of the development platform.
- It supports multiple browsers on different platforms.
- It is powerful and easy to use.

## 2.4 Decision

### 2.4.1 PhpUnderControl vs Xinc

The biggest difference between these two programs is that one is based on CruiseControl and the other is not. However, both support Continuous Integration Testing and PHP and thus satisfy the minimal requirements needed to test ITSAT. The decision was therefore left to the developers, based on which is more comfortable and accessible. Ultimately, phpUnderControl was selected mainly for the following reasons:

- There are more documents, developed extensions and optional packages. Having many more documents and manuals, it is much easier for a developer to get to know the new tool and to use it.
- Learning CruiseControl will not only be useful for testing PHP, but also testing Java and other languages. On the other hand, learning Phing is only useful for testing PHP.
- ITSAT has already incorporated Java as one of its components, so phpUnderControl will be less difficult to install than Xinc.

### 2.4.2 Selenium vs Watir

Having chosen phpUnderControl as the testing tool for ITSAT internal codes, Selenium and Watir were compared to test the browser interface. Both are scriptable, incorporate a recorder program (although in Selenium this only works in Firefox) and support multiple domains. From the view of development, Selenium is much better than Watir for the following reasons:

- Selenium is most suitable for Web GUI Testing whereas Watir is more suitable for Web Services.
- Many useful tools are available in Selenium for Firefox and the new WebDriver, and when this is combined with the PageObjects concept it means that Selenium is a superior cross-browser test tool for the HTML/JavaScript Web, which is the platform of ITSAT.
- Selenium can be coded in multiple languages. Java is the most popular, but others include Perl, PHP, Python and C#. Watir can only be coded in Ruby.

The only drawback of Selenium compared to Watir is that Selenium only has the recorder program for Firefox, and in order to test the other browsers, many new scripts must be created. But considering the many choices of coding languages in Selenium compared to the sole availability of Ruby for Watir, Selenium is the better option without doubt.

# 3. Testing tools

## 3.1 PHPUnit

### 3.1.1 Introduction

PHPUnit is a unit testing software framework for software written in the PHP programming language. Created by Sebastian Bergmann, PHPUnit is one of the xUnit families of frameworks that originated with Kent Beck's SUnit.


PHPUnit was created with the view that the sooner code mistakes can be detected, the quicker they can be fixed. Like all unit testing frameworks, PHPUnit uses assertions to verify that the unit of code being tested behaves as expected.

One of the main goals of PHPUnit is to help developers maintain the basic properties that a test should have:

- **Easy to learn to write**. If it is difficult to learn how to write tests, developers will not learn to write them.
- **Easy to write**. If tests are not easy to write, developers will not write them.
- **Easy to read**. Test code should contain no extraneous overheads so that the test results are not obscured by background noise.
- **Easy to execute**. The tests should run at the touch of a button and present their results in a clear and unambiguous format.
- **Quick to execute**. Tests should fast enough to run hundreds or thousands of times a day.
- **Isolated**. The tests should not affect each other. If the order in which the tests are run changes, the result of the tests should not change.

PHPUnit attempts to achieve these criteria by using PHP as the testing language. Sometimes the full power of PHP is unnecessary for simple straight-line tests, but PHP allows programmers to leverage all the experience and tools already in place. Optional packages are also provided so that simple tests can be created more easily, e.g. *PHPUnit_SkeletonGenerator*, which can generate skeleton test classes from production code classes and vice versa.

PHPUnit errs on the side of isolation rather than rapid execution. Isolated tests are valuable because they provide high-quality feedback. A report containing multiple test failures is not helpful if this is based on the failure of one early test followed by the propagation of the effect to the subsequent tests. Favoring isolated tests encourages designs with a large number of simple objects. Each object can be tested quickly in isolation. The result is better in design and enables faster tests.

The above statement does not contradict the need for ordered sequential tests. For example, in order to determine whether the function that deletes a computer from a work environment is working properly, the computer must first be installed in a work environment, so it is appropriate to test the function of installation before testing the function of deletion. PHPUnit also provides the option to divide complex tests into smaller test suites.

PHPUnit assumes that most tests succeed so the details of successful tests are not reported. When a test fails, that fact is worth noting and reporting. The vast majority of tests should succeed and are not described, although the number of completed tests is reported. This is an assumption that is built into the reporting classes, and not into the core of PHPUnit. When the results of a test run are reported, it will show how many tests were executed and only the details for those that failed.

Tests are expected to be fine-grained, i.e. they should test one aspect of one object. Hence, the first time a test fails, execution of the test halts, and PHPUnit reports the failure. Such fine-grained tests can improve the overall design of the system.

### 3.1.2 Structure of Testing

PHPUnit is a powerful testing tool that contains its own testing functions, such as assertion, validation and exception. In order to implement a simple test, only one file is needed using one of the three functions with its corresponding parameters. However, because it is necessary to test not only output or the result of a function, but also the changes added to databases and the order and parameters of other aspects, it is necessary to associate another file with the testing in order to deal specifically with other aspects of the tests. The structure of testing can therefore be described as follows:

- A file that prepares the testing procedures, including functions such as *construct and teardown*, and validates the correct functions by using the three functions *assert, verify* and *except*. These files are named with the suffix *Test*.
- A file responsible for all the testing that involves other aspects such as database modifications, global values and file directory changes. These files are included with the *Test* files and are named without the suffix.

## 3.2 Selenium

### 3.2.1 Selenium IDE


Selenium IDE is a complete integrated development environment (IDE) for Selenium tests and is implemented as a Firefox extension. It allows recording, editing and debugging tests. Selenium IDE includes the entire Selenium Core, allowing developers to record and play back tests easily and quickly in the realistic environment in which they will run. It is not only recording tool, but a complete IDE.

The features of Selenium IDE include:

- Easy record and playback
- Intelligent field selection using IDs, names or XPath as needed
- Autocomplete for all common Selenium commands
- Walk through tests
- Debug and set breakpoints
- Save tests as HTML, Ruby scripts or any other format
- Support for Selenium user-extension .js files
- Option to insert the title of every page automatically
- Easy customization using plugins

Not only is Selenium IDE easy to work with, it also comes with other useful elements such as *Screenshot on failure*, which takes a screenshot of the browser once an error is detected. The investigator can then determine what happened in the browser when the test was running without having to repeat all the steps of the test.

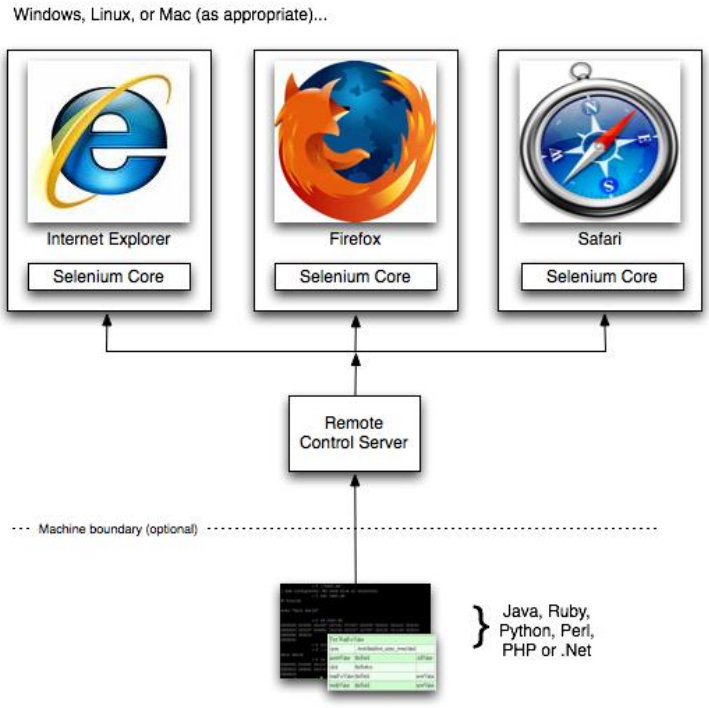
### 3.2.2 Selenium RC

Selenium RC (Remote Control) is a tool that allows developers to write automated web application UI tests in any programming language against any HTTP website using any mainstream JavaScript-enabled browser. It is ideal for testing complex AJAX-based web user interfaces under a Continuous Integration system, as required for ITSAT.

Selenium RC comprises two components:

- The Selenium Server which launches and kills browsers, interprets and runs the Selenese commands passed from the test program, and acts as an *HTTP proxy*, intercepting and verifying HTTP messages passed between the browser and the AUT (Application Under Test).
- Client libraries which provide the interface between each programming language and the Selenium RC Server.

Here is a simplified architectural representation:




The diagram shows that the client libraries communicate with the Server by passing each Selenium command for execution. Then the server passes the Selenium command to the browser using Selenium-Core JavaScript commands. The browser, using its JavaScript interpreter, executes the Selenium command. This runs the Selenese action or verification specified in the test script.

- Selenium Server

Selenium Server receives Selenium commands from the test program, interprets them, and reports the results of those tests.

The RC server bundles Selenium Core and automatically injects it into the browser. This occurs when the test program opens the browser (using a client library API function). Selenium Core is a JavaScript program, a set of JavaScript functions that interpret and execute Selenese commands using the browser's built-in JavaScript interpreter.

The Server receives the Selenese commands from the test program using simple HTTP GET/POST requests. This means developers can use any programming language that can send HTTP requests to automate Selenium tests on the browser.

- Client Libraries

The client libraries provide programming support allowing developers to run Selenium commands from a program of their own design. There is a different client library for each supported language. A Selenium client library provides a programming interface (API) or set of functions that run Selenium commands from a program. Within each interface, there is a programming function that supports each Selenese command.

Each client library takes a Selenese command and passes it to the Selenium Server so that a specific command can be executed against the application under test (AUT). The client library also receives the result of that command and passes it back to the program. The program can receive the result and store it as a program variable and report it as a success or failure, or possibly take corrective action if it was an unexpected error.

To create a test program, the developer simply writes a program that runs a set of Selenium commands using a client library API. Optionally, if the developer already has a Selenese test script created in Selenium IDE, he can also *generate the Selenium RC code*. Selenium IDE can translate Selenium commands into a client-driver's API function calls using its Export menu.

# 4. Analysis of objectives and requirements for testing systems

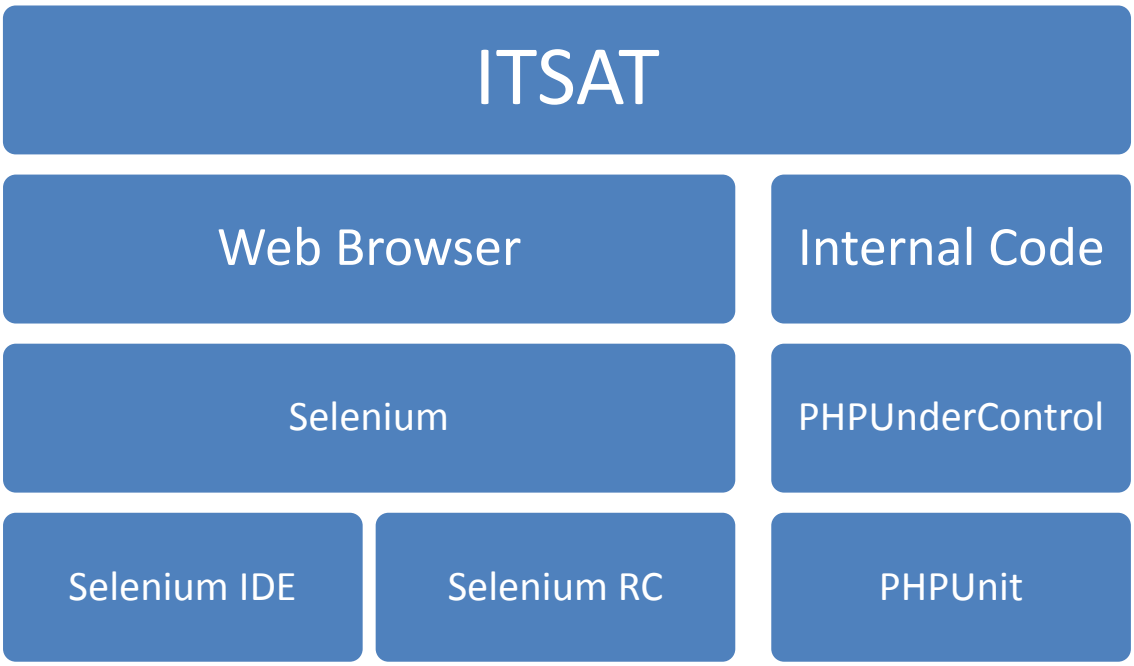
## 4.1 Analysis of objectives

ITSAT comprises three main components depending on whether there is interaction between the client and ITSAT, and the method of interaction. The following chart describes the three components:

| | User interactive | Non-user interactive |
|---|---|---|
| **Web** | • ITSAT website homepage | - |
| **Console** | • Gate One | • sat-remotedo<br>• sat-useradd<br>• sat-machineadd<br>• … |

*The three main components of ITSAT*

The client can with ITSAT through the web or console, the former by navigating to the ITSAT homepage and the latter by applying Gate One, which is included in ITSAT. Because the whole ITSAT website is available to the client, there is no web and non-user interactive aspect. On the other hand, the non-user interactive part of ITSAT can be accessed through the console, which provides all the included functions, including the highly important sat-remotedo. Others include sat-useradd, which adds new users to ITSAT, and sat-machineadd, which adds new computers to ITSAT. The goal of this project is to test the web user interactive aspect of ITSAT using Selenium and the non-user interactive console using PHPUnit.




*Structure of ITSAT and the corresponding testing tools*

## 4.2 Analysis of requirements

Before initiating the tests, the testing system must be useful and effective according to the following criteria:

- **The test must be precise.** One of the most important properties of a testing system is that the test itself must be precise. The test must not include bugs, errors or omissions.
- **The test must be maintainable and flexible.** ITSAT is a project under development, so many previous decisions may need to be reconsidered and changed and the tests must be able to react accordingly. The tests must also be easily maintained so that minor changes to ITSAT will not need the entire test to be redefined.
- **The test must be adaptable to other work environments.** The test must be easily adaptable to other work environments so that it can be applied to computers from different work environments.
- **The test must be easy to execute and must detect errors.** The goal of a test is to detect errors and fix them, so it must be easily executable and must allow testers to detect errors easily.

# 5. Design and optimizations

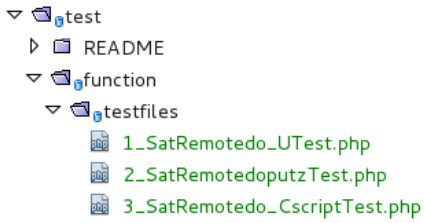
Once the testing system has been created, many optimizations are needed to meet the critera listed in Section 4.2.

## 5.1 Optimizations of PHPUnit

### 5.1.1 Fixing orders for the tests

Initially, all the tests are placed in the directory *test/function/testfiles* and by executing the command *$>phpunit test/function/testfiles* in terminal, the tests are executed in alphabetical order, causing order issues also known as the dependency problem: e.g. test of deleting a computer must not be tested before the test of adding the computer.

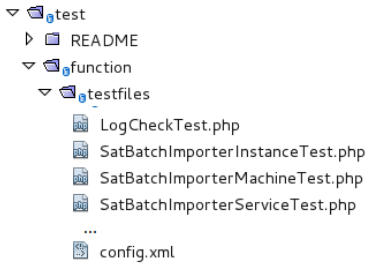
#### *5.1.1.1 First approach: adding prefix*




The first solution is adding a prefix to all the tests so they are ordered preferentially, e.g. adding a number. This will avoid dependency problems. However, this solution was rejected for two main reasons:

- Aesthetically, it is not an appropriate solution because numerical prefixes can be confusing and the file directory must be viewed to see the tests because the numerical prefix is not related to the test.
- During continuous testing, many more testing files will be added and these will need renumbering on a regular basis. For example, if a new test is placed second in the queue then all subsequent tests will need to be renumbered.

#### *5.1.1.2 Second approach: a configuration file*




A fixed order to the tests can be achieved without numerical prefixes if a configuration file is used to determine the order of execution. The configuration file reorder can reorder the tests as preferred and allow developers to divide tests into different groups, also known as test suites. A developer can therefore run specific tests rather than all of them.

```
<testsuite name = "MachineManagement">
    <file>/opt/itsat/develop/test/function/testfiles/SatMachineAddTest.php</file>
    <file>/opt/itsat/develop/test/function/testfiles/SatMachineDelTest.php</file>
    <file>/opt/itsat/develop/test/function/testfiles/SatBatchImporterMachineTest.php</file>
</testsuite>
<testsuite name = "UserManagement">
    <file>/opt/itsat/develop/test/function/testfiles/SatBatchImporterUserTest.php</file>
    <file>/opt/itsat/develop/test/function/testfiles/SatUserAddTest.php</file>
    <file>/opt/itsat/develop/test/function/testfiles/SatUserDelTest.php</file>
</testsuite>
<testsuite name = "ServiceManagement">
    <file>/opt/itsat/develop/test/function/testfiles/SatBatchImporterServiceTest.php</file>
    <file>/opt/itsat/develop/test/function/testfiles/SatBatchImporterInstanceeTest.php</file>
</testsuite>
<testsuite name = "SatRemotedo">
    <file>/opt/itsat/develop/test/function/testfiles/SatRemotedoputzTest.php</file>
    <file>/opt/itsat/develop/test/function/testfiles/SatRemotedo_wTest.php</file>
    <file>/opt/itsat/develop/test/function/testfiles/SatRemotedo_pTest.php</file>
    <file>/opt/itsat/develop/test/function/testfiles/SatRemotedo_STest.php</file>
    <file>/opt/itsat/develop/test/function/testfiles/SatRemotedo_MTest.php</file>
    <file>/opt/itsat/develop/test/function/testfiles/SatRemotedo_ikTest.php</file>
    <file>/opt/itsat/develop/test/function/testfiles/SatRemotedo_cTest.php</file>
    <file>/opt/itsat/develop/test/function/testfiles/SatRemotedo_CscriptTest.php</file>
    <file>/opt/itsat/develop/test/function/testfiles/SatRemotedo_RTest.php</file>
    <file>/opt/itsat/develop/test/function/testfiles/SatRemotedo_UTest.php</file>
    <file>/opt/itsat/develop/test/function/testfiles/SatRemotedoputTest.php</file>
    <file>/opt/itsat/develop/test/function/testfiles/SatRemotedoputxTest.php</file>
    <file>/opt/itsat/develop/test/function/testfiles/SatRemotedogetfTest.php</file>
    <file>/opt/itsat/develop/test/function/testfiles/SatRemotedogetfzTest.php</file>
    <file>/opt/itsat/develop/test/function/testfiles/SatRemotedogetdTest.php</file>
    <file>/opt/itsat/develop/test/function/testfiles/SatRemotedocountTest.php</file>
    <file>/opt/itsat/develop/test/function/testfiles/SatRemotedogetdzTest.php</file>
    <file>/opt/itsat/develop/test/function/testfiles/SatRemotedo_ikxTest.php</file>
    <file>/opt/itsat/develop/test/function/testfiles/SatRemotedo_skTest.php</file>
    <file>/opt/itsat/develop/test/function/testfiles/SatRemotedo_oTest.php</file>
</testsuite>
<testsuite name = "SatFunctions">
    <file>/opt/itsat/develop/test/function/testfiles/SatGenpassTest.php</file>
</testsuite>
<testsuite name = "LogCheck">
    <file>/opt/itsat/develop/test/function/testfiles/LogCheckTest.php</file>
</testsuite>
```

*A fragment of file config.xml.*

The main advantages of configuration files are:

- Whenever a new test is added to the project, it only needs to be added to the configuration file to solve dependency problems (without changing other testing files).
- It provides more debugging alternatives when a test fails. Running a test suite is more efficient than running the whole test, which is time consuming, or running that single test, which might not solve the problem if the bug was produced in previous tests due to dependencies.

### 5.1.2 Change from local testing to generic testing

The tests are implemented on one computer by one person, so testing can only be carried out once the tester has updated his workspace to the latest version of the repository. This creates several development problems:

- If multiple developers modify the repository and a test fails, it will be almost impossible to identify which of the latest modifications are correct and incorrect.
- If a test fails, it means that the repository is unstable, which is against the policy of continuous integration.

In order to solve these problems, all developers must carry out the testing themselves before committing any new changes to the remote repository.

### 5.1.2.1 First approach: documentation and manual modifications

The first approach is to write manuals and readmes about testing, so that developers gain an overall idea about how testing works. Because testing files are implemented and configured locally, much of the information about the computer is hard-coded, i.e. written in the code itself. In order for other developers to conduct testing, minor changes and modifications are needed, and these should also be included in the documentation.

Although this allows other developers to do the testing, the testing code can be so long and complex that it takes a long time to do the modifications necessary to make it to work in the workspace of other developers. Also, the changes are implemented manually so human errors are inevitable.

### 5.1.2.2 Second approach: test configuration files

The alternative approach is to let the test change the necessary values by including a configuration file. The developers then only need to change the values of a small configuration file from which the test will gain the necessary information.

In order to give as provide information as possible to the developers, the configuration file also allows comments by starting any line with the "#" symbol. This feature was added so that the manuals and readmes discussed in the first approach can be included in the configuration file.

```
#The variable of the first line must be written between two quotation marks
#The input must be successful when executing the command:
#   sat-remotedo -w <work environment identifier> -c <command>
#   NOTE: you can add more than just one command by using the semicolon as separator
#The number of lines following the <number of expected outputs> must match with the indicated number
#The list contains <name of machine> with each <expected outputs>, separated only by a coma
#Indicate with l (literal) or r (regular expression) at the beginning of each expected output to specify the filter

#FORMAT:
#<work environment identifier> <command>
#<number of expected outputs>
#list of <name of machine> <expected outputs

#PROCEDURE:
#Executes: sat-remotedo -w <work environemnt> -c <commands>
#Checks for the expected output in the corresponding machine in the output.
#If the expected output doesn't exist, or a machine wasn't present in the output, the test reports FAILED.
#The detection of work ERROR/Error/error is done throughout the test. If the word is found, the test reports FAILED.

#EXAMPLE:
#"0" "echo /etc/hostname;id"
#8
#srvitsat,l,/etc/hostname
#srvitsat,r,uid.*\(rcpadm\)
#itsat-sg01,l,/etc/hostname
#itsat-sg01,l,uid=1002(rcpadm) gid=1002(itsatadm) grupos=1002(itsatadm)
#itsat-sg02,l,/etc/hostname
#itsat-sg02,l,uid=1002(rcpadm) gid=1002(itsatadm) grupos=1002(itsatadm)
#itsat-sg03,l,/etc/hostname
#itsat-sg03,l,uid=1002(rcpadm) gid=1002(itsatadm) grupos=1002(itsatadm)
```

*A fragment of the configuration file for SatRemotedo_cTest.php*

The first two paragraphs provide an overall summary of the configuration file format for each test, followed by a third paragraph which describes the procedure of the test so that if the test fails, the developer can try to debug it. The final paragraph provides an example of a correct configuration file.

### 5.1.2.3 Third approach: test configuration file templates

The second approach allows generic testing, but improvements are required to achieve repository updates. Because the repository ensures that all computers have identical configuration files, any changes introduced by a tester will propagate to other developers when they update their own workspace from the repository.

In order to avoid this problem, a template for each test configuration file can be created. The repository will only contain the templates instead of the actual configuration files so that any updates will only affect the local templates. This method is beneficial when minor changes are applied to a configuration file, e.g. adding a value and maintaining all the other values. Instead of changing all the values again, the developers only need to add the new value.

```
-rw-r--r-- 1 root root  102 Oct 16 09:43 machinedel.txt
-rw-r--r-- 1 root root  534 Oct 28 17:48 machinedel.txt.template
-rw-r--r-- 1 root root  250 Oct 16 09:43 satremotedoc.conf
-rw-r--r-- 1 root root 1865 Oct 11 11:34 satremotedoc.conf.template
-rw-r--r-- 1 root root    7 Oct 15 08:42 satremotedocount.conf
-rw-r--r-- 1 root root    7 Oct 11 11:34 satremotedocount.conf.template
-rw-r--r-- 1 root root  765 Oct 16 09:43 satremotedocscript.conf
-rw-r--r-- 1 root root 2488 Oct 11 11:34 satremotedocscript.conf.template
-rw-r--r-- 1 root root   53 Oct 16 09:43 satremotedogetd.conf
-rw-r--r-- 1 root root  684 Oct 11 11:34 satremotedogetd.conf.template
-rw-r--r-- 1 root root  620 Oct 15 08:42 satremotedom.conf
-rw-r--r-- 1 root root  620 Oct 11 11:34 satremotedom.conf.template
-rw-r--r-- 1 root root   46 Oct 16 09:43 satremotedoo.conf
-rw-r--r-- 1 root root 1239 Oct 11 11:34 satremotedoo.conf.template
-rw-r--r-- 1 root root  727 Oct 15 08:43 satremotedop.conf
-rw-r--r-- 1 root root  727 Oct 11 11:34 satremotedop.conf.template
-rw-r--r-- 1 root root  149 Oct 16 09:43 satremotedoput.conf
-rw-r--r-- 1 root root  803 Oct 11 11:34 satremotedoput.conf.template
-rw-r--r-- 1 root root  149 Oct 16 09:43 satremotedoputx.conf
-rw-r--r-- 1 root root 1017 Oct 11 11:34 satremotedoputx.conf.template
-rw-r--r-- 1 root root   53 Oct 16 09:43 satremotedoputz.conf
-rw-r--r-- 1 root root  636 Oct 11 11:34 satremotedoputz.conf.template
-rw-r--r-- 1 root root  242 Oct 16 09:43 satremotedor.conf
-rw-r--r-- 1 root root 1787 Oct 11 11:34 satremotedor.conf.template
-rw-r--r-- 1 root root  734 Oct 15 08:44 satremotedos.conf
-rw-r--r-- 1 root root  734 Oct 11 11:34 satremotedos.conf.template
-rw-r--r-- 1 root root  269 Oct 16 09:43 satremotedou.conf
-rw-r--r-- 1 root root 1959 Oct 11 11:34 satremotedou.conf.template
```

*Each configuration file has its own template.*

### *5.1.2.4 Fourth approach: auto-generating configuration files*

Although the use of configuration file approach avoids manual editing and many errors, it does not change the fact that for a file must be edited for each test. However, because most of the changes are the same, it is beneficial to use a file that can autogenerate all the test configuration files.

```
#main server
#machine_name hostname
debian1 debian1 /opt/itsat/home/rcpadm/
#the machines you want to add
#machine_name ip_address OS work_environment username password connection_mode hostname
itsat-sg01 192.168.80.202 Linux 1 root root sshsftp itsat-sg01 /opt/itsat/home/rcpadm/
itsat-sg02 192.168.80.203 Linux 1 root root sshsftp itsat-sg02 /opt/itsat/home/rcpadm/
itsat-sg03 192.168.80.204 Linux 1 root root sshsftp itsat-sg03 /opt/itsat/home/rcpadm/
```

*The configuration file for the TestAutoGenerator.php*

The file for the TestAutoGenerator.php is similar to those of the original configuration files, providing information about the format requirements and also an example. On execution, it also provides the developer with an option to back up all the existing configuration files. All the back-up files have the extension of *.backup* and new configuration files will replace the original ones. This approach is the best thus far for the following reasons:

- The developer does not have to understand any testing codes written by others.
- The developer needs to only modify one file so that all the tests work. This speeds up the testing process by avoiding problems associated with the configuration files.

- The template provides developers with the minimum necessary information concerning the configuration files, e.g. issues that can cause a test to fail, and the format required for a certain test.

```
[ITSAT][srvitsat].root:/opt/itsat/develop/test/function/testfiles/includes > php TestAutoGenerator.php
Reading autogenerate.conf...
Reading autogenerate.conf... OK!

Do you want to create backup files for all the configuration files that will be generated? (Y/N)
Y
Creating machinebatch.txt...
Creating machinebatch.txt... OK!

Creating machineadd.txt...
Creating machineadd.txt... OK!
```

*The TestAutoGenerator also provides an option to back-up all the existing configuration files.*

### 5.1.3 Test error outputs

When testing is applied to computers belonging to other developers, the testing results must be modified. PHPUnit has an output by default: a dot (.) indicating a pass and an F indicating a fail.

```
PHPUnit 3.7.27 by Sebastian Bergmann.

.

Time: 1.18 seconds, Memory: 8.25Mb

OK (1 test, 1 assertion)
```

```
PHPUnit 3.7.27 by Sebastian Bergmann.

F

Time: 130 ms, Memory: 7.50Mb

There was 1 failure:

1) SatBatchImporterServiceTest::testSat_batchimporter_sevice
Failed asserting that 1 matches expected 0.

/opt/itsat/develop/test/function/testfiles/SatBatchImporterServiceTest.php:65

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

*The output of a successful test (top) and a failed test (bottom).*

Such binary outputs are uninformative because the test itself can be implemented incorrectly, thus giving false negatives and positives. The following chart describes the relationship between test subjects and results:

| | **The testing subject works** | **The testing subject fails** |
|---|---|---|
| **Test passed** | | "False Positive" |
| **Test failed** | "False Negative" | |

*Test result and tested subject relationship chart*

- The green field is expected from the tests: a pass for a successful test and a fail for an unsuccessful test.
- The yellow field is a **False Negative**, which occurs when a successful test is reported as a failure. This is a minor problem because the testing procedure, implementation and subject are revised and the error will eventually be found.

- The red field is a **False Positive**, which occurs when an unsuccessful test is given a pass. False positives must be taken into account during test implementation because they are much more difficult to detect and is dangerous because the test subject and the error are often ignored.

The goal of testing is not only to detect errors but also to find solutions, so we need to modify the default output by adding more information.

### *5.1.3.1 First approach: outputting the error with more information*

In order to provide developers with more information about each test, documentation about each test can be added to the testing system, but instead of stating all the possible causes of test failure it is much more effective if the test itself can identify the problem. Thus, if a test fails, the output will explain the cause of failure.

```
[ITSAT][srvitsat].root:/opt/itsat > phpunit --colors /opt/itsat/develop/test/function/te
PHPUnit 3.7.27 by Sebastian Bergmann.

F

Time: 1.05 seconds, Memory: 7.50Mb

There was 1 failure:

1) SatBatchImporterServiceTest::testSat_batchimporter_sevice
Failed asserting that '

                  There was an error message in the output!
' matches expected 0.

/opt/itsat/develop/test/function/testfiles/SatBatchImporterServiceTest.php:65

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

*The test failed because an error message was found during execution.*

By knowing the cause of the failure immediately, it will take less time to debug and correct the error, because errors can only be produced in a certain area of the code and it is not necessary to revise the whole code. But if the test or the function is complex or linked in chain where one function leads to another, then an error produced in the final execution cannot identify the original source. This approach achieves rapid error identification, but another approach is needed to minimize the amount of revision required.

### *5.1.3.2 Second approach: adding the failed command line*

An error is usually produced the execution of a single command line, so if the test can detect the command line responsible for the error, then revision can be minimized to only one command line or a specific aspect of the code related to that command line.

```
[ITSAT][srvitsat].root:/opt/itsat > phpunit --colors /opt/itsat/develop/test/function/te
PHPUnit 3.7.27 by Sebastian Bergmann.

F

Time: 537 ms, Memory: 7.50Mb

There was 1 failure:

1) SatBatchImporterServiceTest::testSat_batchimporter_sevice
Failed asserting that '

                    There was an error message in the output!

                    Error:  /etc/init.d/apache2 status already exists in DB.

                    The command was:

                    sat-batchimporter -s -f /opt/itsat/develop/test/function/testfiles/in
' matches expected 0.

/opt/itsat/develop/test/function/testfiles/SatBatchImporterServiceTest.php:65

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

*The test output contains not only the error but also the exact error output and the command line which caused it.*

As shown above, the test fails because the service Apache has already been inserted in Database, which does not fulfill the pre-condition necessary to run the test. The command line that provoked the error is the command used to batch import the service into Database.

This approach is the most effective way to detect and solve errors, but another possible issue must be taken into account, i.e. the test configuration files. Although developers are provided with documentations and examples, it is still possible to write incorrect configuration files that induce test failures because the information needed for the test has not been provided correctly.

#### *5.1.3.3 Third approach: outputting received output*

Since the test requires data from a configuration file, it is not possible for the test itself to determine whether the data provided is correct or not, so although the tested subject is correct, the test will always fail if the configuration file is not correctly set up. To address this issue, the test during execution will output the result from running the test subject, thus eliminating false negatives by checking manually if the result is the same as expected.

```
The following command failed:
SATPROCEXEC_FORCE_SEQ=TRUE sat-remotedo -M "*sat*"
With the following output:
START -->

#ID             WorkEnv(#id)      Op.System       Prod.Env                  IP   Machine Name
================================================================================================
  1             Default(#1)           Linux            PRO       192.168.80.88       srvitsat
 18             Default(#1)           Linux            PRO      192.168.80.136     itsat-sg01
 19             Default(#1)           Linux            PRO      192.168.80.137     itsat-sg02
 20             Default(#1)           Linux            PRO      192.168.80.138     itsat-sg03
<---END
```

*The test also gives information about the result when it fails.*

If the result is as expected given the command, then it means the test subject is working and there is an error in the implementation or configuration files of the test.

### 5.1.4 Test execution issues

A problem can also exist during the test execution process, e.g. missing configuration files or incorrect test implementation which can generate loops that cannot output any results. These cases must be taken into consideration because the test subject may be functional but the test itself may be faulty.

#### *5.1.4.1 First approach: timeout implementation*

The timeout implementation is the first approach to detect loops. Each test is given a specific amount of time to complete the test, and once this runs out the test will automatically output a timeout exception together with the last command executed by the test. However, the execution time depends not only on the test but also on other factors such as the environment (the processor, other hardware and RAM) and the number of times the same test has been run. Many variants of this approach have been reported, but none control the individual contribution of each factor to the execution time.

```
[ITSAT][srvitsat].root:/opt/itsat > phpunit --colors /opt/itsat/develop/test/function/testfiles/SatRemotedo_MTest.php
PHPUnit 3.7.27 by Sebastian Bergmann.

..

Time: 619 ms, Memory: 2.75Mb

OK (2 tests, 2 assertions)
[ITSAT][srvitsat].root:/opt/itsat > phpunit --colors /opt/itsat/develop/test/function/testfiles/SatRemotedo_MTest.php
PHPUnit 3.7.27 by Sebastian Bergmann.

..

Time: 832 ms, Memory: 2.75Mb

OK (2 tests, 2 assertions)
[ITSAT][srvitsat].root:/opt/itsat > phpunit --colors /opt/itsat/develop/test/function/testfiles/SatRemotedo_MTest.php
PHPUnit 3.7.27 by Sebastian Bergmann.

..

Time: 1.09 seconds, Memory: 2.75Mb

OK (2 tests, 2 assertions)
[ITSAT][srvitsat].root:/opt/itsat > phpunit --colors /opt/itsat/develop/test/function/testfiles/SatRemotedo_MTest.php
PHPUnit 3.7.27 by Sebastian Bergmann.

..

Time: 1.12 seconds, Memory: 2.75Mb

OK (2 tests, 2 assertions)
[ITSAT][srvitsat].root:/opt/itsat > phpunit --colors /opt/itsat/develop/test/function/testfiles/SatRemotedo_MTest.php
PHPUnit 3.7.27 by Sebastian Bergmann.

..

Time: 909 ms, Memory: 2.75Mb

OK (2 tests, 2 assertions)
[ITSAT][srvitsat].root:/opt/itsat > phpunit --colors /opt/itsat/develop/test/function/testfiles/SatRemotedo_MTest.php
PHPUnit 3.7.27 by Sebastian Bergmann.

..

Time: 1.03 seconds, Memory: 2.75Mb

OK (2 tests, 2 assertions)
```

*Multiple executions of the same test.*

#### *5.1.4.2 Second approach: outputting the configuration files for each test*

Because test files updated to the repository must be working (requirement for stability), the most likely reason for a test to execute incorrectly is its configuration files. The second approach provides information about which configuration files are being used during the test.

```
[ITSAT][srvitsat].root:/opt/itsat > phpunit --colors --debug /opt/itsat/develo
SatRemotedo_MTest.php executes with config files: [satremotedom.conf]
SatRemotedo_MTest.php executes with config files: [satremotedom.conf]
PHPUnit 3.7.27 by Sebastian Bergmann.


Starting test 'SatRemotedo_MTest::testRemotedo_M'.
.SatRemotedo_MTest.php was executed with config files: [satremotedom.conf]

Starting test 'SatRemotedo_MTest::testRemotedo_M2'.
.SatRemotedo_MTest.php was executed with config files: [satremotedom.conf]


Time: 1.05 seconds, Memory: 2.75Mb

OK (2 tests, 2 assertions)
```

*The configuration file is shown before and after the test.*

In order to identify the configuration file, the information is shown before and after the test. If the test fails to terminate and generate a result, the earlier configuration file can be used, whereas if the test reports a failure then the later configuration file can be revised.

### 5.1.5 MockObject

#### *5.1.5.1 What is MockObject?*

In object-orientated programming, mock objects are simulated objects that mimic the behavior of real objects in controlled ways. A programmer typically creates a mock object to test the behavior of another object, in much the same way that a car designer uses a crash test dummy to simulate the dynamic behavior of a human in vehicle impacts. This is one of the optimizations that can be used to isolate each test, thus eliminating their dependencies.

#### *5.1.5.2 Advantages and Disadvantages*

Before applying Mock Object to the testing system, it is necessary to compare its advantages and disadvantages. Mock Object has many advantages:

- **Create tests in advance – TDD**. This is one of the strongest benefits of Mock Object. Using a Mock, a Service Test can be created before the actual service exists, allowing the tests to be added to an automation environment in the development process. This is also known as TDD, Test Driven Development.
- **Write tests for resources that are not accessible**. This is a crucial advantage because some tests cannot be implemented due the presence of a firewall. In this situation, the mock of the test subject can be created in an accessible environment, including a local computer.
- **Isolate systems**. A test that only addresses part of a system without including other parts affecting it can be difficult to handle and maintain. This is because the other systems will add noise to the test data and make it harder to make firm conclusions e.g. from Database data. Using mocks, all dependencies can be removed by mocking all systems except the system to be tested. Mocks for isolation can be made extremely simple but reliable, rapid and predictable.

On the other hand, there are also disadvantages such as:

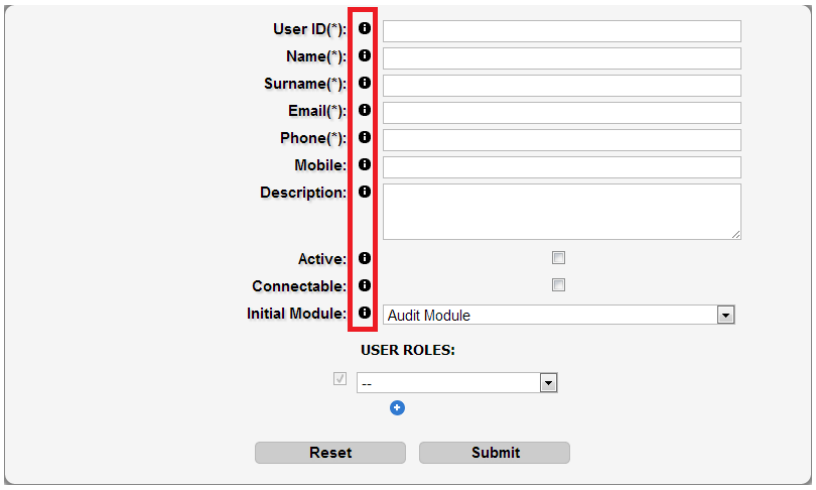
- **Redundancy of work.** The mock is often discarded after use thus it only has a short-term value. It is difficult to decide whether it is worth the effort to create a mock object.
- **Deployment constraints.** Creating a mock can be an obstacle to changing from local testing to generic testing (see Section 3.3.2). If all developers want to test the same Mock Object, it must be placed where all users have access to it.
- **Even Mocks get bugs.** Since mocks are also implemented, they are also a subject to bugs. One solution is to carry out extensive testing of the mocks, but this leads to complications, including the loss of confidence in tests based on mocks (see Section 3.3.3).
- **Differences between actual and mocked environments.** The mock is not the same as the genuine environment and is liable to misinterpretation in the same way that a photograph can misrepresent a person.

#### *5.1.5.3 Decision*

Ultimately, Mock Object was not applied to the ITSAT testing system because its disadvantages were deemed to outweigh its advantages, it contradicted our previously stated objectives and it is orientated towards unit testing instead of functional testing. ITSAT has many functions, and the main objective of this project was to test the functionalities of ITSAT instead of each unit and class.

## 5.2 Optimizations of Selenium

Once the Selenium tests were created, they were optimized to offer more flexibility. Because the ITSAT website is undergoing development, any minor change such as adding tooltips would be capable of breaking down the entire Selenium testing system.




*The form for the creation of a new user with the tooltips (inside the red rectangle) added.*

This was a major obstacle in the development of Selenium tests because each time a minor change was made to the website, the Selenium tests needed to be modified accordingly. To avoid this, the precision of Selenium test was lowered so that instead of checking that a space for input must be followed after a certain keyword, the program will only check the existence

of both elements regardless of their relative positions. Selenium will therefore report a pass regardless of how many spaces are added between tootltips or where and how they are added to the webpage. Although this solution is strictly and aesthetically acceptable, it is less than optimal because the keyword and the space for input can be so far away from each other that they lose their original meaning, yet the test will still report as passed.

Another optimization applied to Selenium was the use and distinction between verification and assertion. Initially, all the Selenium tests used the command assertion in order to check if website representation was correct. However, if one assertion is reported as false, the whole test stops running, leaving the rest of the test unchecked even though the asserted aspect is irrelevant to these aspects of the test.

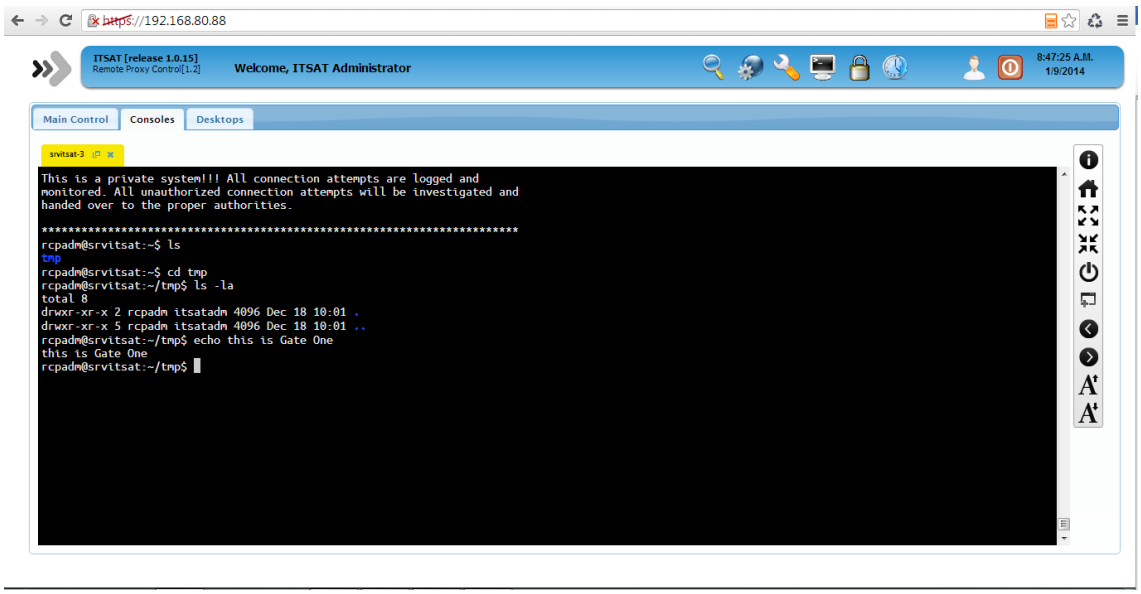
A simple example of this phenomenon is a misspelling. If the keyword from the picture above “Email” is spelled “E-mail”, the Selenium test will stop and report a fail even though the main goal of the test is to check that the form can be saved correctly into the database, regardless of keyword spelling. A verification method has therefore been introduced. The only difference between these methods is that verification can be reported as passed or failed but the test will continue to run, whereas with assertion the test halts. The optimization consists of checking the less important aspects of a website with verification and the more important ones with assertion in order to increase the effectiveness of the Selenium test.

# 6. Limitations

Although PHPUnit and Selenium test most parts of ITSAT, there are still limitations, some of which have been solved by creating a temporary workaround or bypass so that it will not hinder the execution of a test, whereas others are unavoidable.

## 6.1 Gate One, plugins and flash components

The greatest limitation of all is being unable to test one of the most important features of ITSAT – Gate One. This is an open source, web-based terminal emulator with a powerful plugin system. It comes bundled with a plugin that turns Gate One into an SSH client but Gate One can actually be used to run *any* terminal application. It can be embedded into other applications to provide an interface into serial consoles and virtual servers. Selenium is unable to test such plugins and flash components, and instead of recording the interaction that has happened inside the plugins, Selenium detects all plugins as a static picture.




*Selenium is unable to report any interactions that occur inside a certain plugin.*

Future expectations and possible solutions:

- Selenium is working on a new component, FlexUISelenium, which can detect flash and plugin interactions. This is a new project within Selenium and is still not officially released, but offers hope for future Selenium flash testing projects.
- Gate One is not a plugin like any other because it involves many interactions with terminals and other remote computer systems. One future expectation is to communicate with the developers of Gate One and ask them to consider ways of automating the tests on Gate One or any other helpful tools that can do it.

## 6.2 Concurrence testing

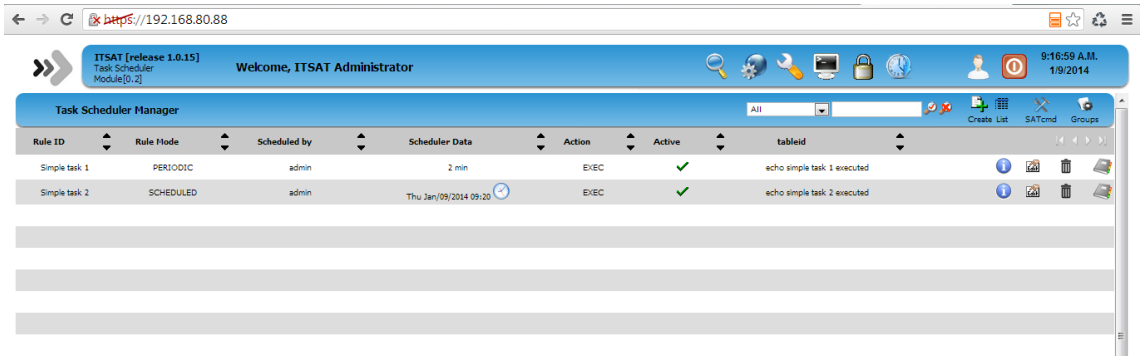
Another problem of the testing system this far is that they are installed in a single computer, so there is no way to test how two computers interact with each other. The problem is detected manually, communicated to the developers and a fix patch is issued in the next version of Gate One. However, there is no guarantee that the problem will not recur.

Future expectations and possible solutions:

- Manual testing remains the best approach thus far and it is uncertain what progress there will be in the future. The concurrence problem involves operations on two or more computers simultaneously to see whether they function correctly.

## 6.3 Conditional and loop testing

Unlike other testing tools, Selenium does not have functions such as if-clauses, while loops or parameterization. This prevents the testing of special cases, repetitive interactions and conditional testing.



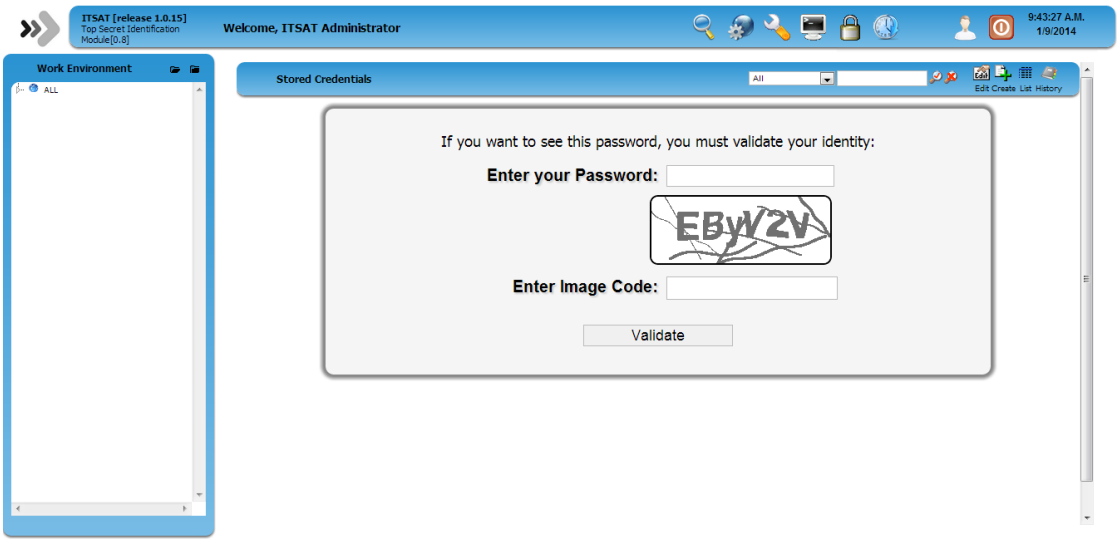

*This limitation posed the biggest obstacle during the testing of a TSM Module becaise it cannot determine periodic tasks or a scheduled task at a certain time.*

Future expectations and possible solutions:

- A possible solution is to incorporate Selenium into CruiseControl. This is possible because Selenium allows its code to be exported as PHPUnit testing files, which can be added directly to the CruiseControl testing suite under phpUnderControl. CruiseControl allows loop executions and scheduled time executions which address the rigidness of Selenium.

## 6.4 Image capture verification

In the TSIM module of ITSAT, an image capture detector is used to prevent third party programs accessing the database of saved credential files. Currently, there is no web browsing testing tools that can identify the image because the sole purpose of the image capture detector is to prevent any non-manual access.




*In order to view the credentials, a password and an image code is needed*

Future expectations and possible solutions:

- A workaround to this problem has been developed which involves creating a script that instead of verifying the image code, verifies a certain key string as the correct password, in this case “Selenium Test”. This script will be run before the Selenium test against the TSIM module. Whenever Selenium accesses this page, the credentials will only be shown when inputting “Selenium Test” into the image code area.

## 6.5 Compatibility with other web browsers

Currently, all Selenium tests are run in the Firefox browser, but it is also important to be able to test ITSAT’s functions in other browsers.

Future expectations and possible solutions:

- Although Selenium supports other browsers like IE, Mozilla and Safari, it is orientated mainly towards Firefox so there are many issues and bugs when running Selenium in other web browsers. A possible solution is to incorporate other web testing tools which are more widely orientated like TestComplete or Ranorex Studio. Because Selenium allows the export of its code in many different languages, the process of incorporation should not be too difficult.

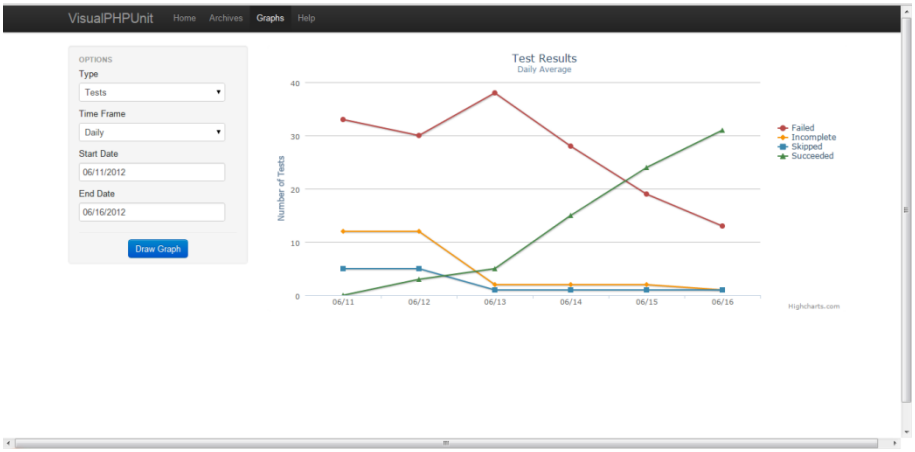
## 6.6 Representation of Results

Neither Selenium nor PHPUnit has an effective representation method and showing numbers of passed and failed tests is not enough to evaluate whether or not a project is successful.

Future expectations and possible solutions:

- There are many tools that can be used for the graphic representation of PHPUnit test results. For example, VisualPHPUnit is a visual front-end for PHPUnit. It offers features such as the ability to view the progress of tests in the form of graphs, an option to maintain a history of test results through the use of snapshots, enumeration of

PHPUnit statistics and messages. As a future expectation, it can be useful to have a graphic representation of test results.




*A sample of VisualPHPUnit at representing test results*

# 7. Test Implementation

## 7.1 Internal code testing files with PHPUnit

### 7.1.1 LogCheckTest.php

- Purpose:
  - This file exclusively checks if any log files have been modified during the execution of tests. ITSAT provides a list of log files to detect any errors or warnings during execution.
- Pre-condition:
  - All log files must be blank before running the tests.
- Post-condition:
  - The test fails if any log file has registered any errors or warnings.
- Configuration file:
  - logfilter.conf

### 7.1.2 SatBatchImporterInstanceTest.php

- Purpose:
  - Tests the correct function of batch importing a file of new instances for a service into a database. An instance is the link between a service and a computer, i.e. an instance grants permission for a computer to run a service.
- Pre-condition:
  - The services of the instance must exist in the database.
  - The instances to be added do not exist in the database.
- Post-condition:
  - Test fails if the instances are not correctly added to the database.
  - Test fails if the instances cannot be executed correctly.
- Configuration file:
  - instancebatch.txt

### 7.1.3 SatBatchImporterMachineTest.php

- Purpose:
  - Tests the correct function of batch importing a file of computers into database. This test adds computers to a work environment and also checks if there is a connection between the main computer and the added computer using a ping.
- Pre-condition:
  - The indicated work environment must exist for the new computers.
  - The computers must be turned on.
  - The computers are not in the database.
- Post-condition:
  - Test fails if the computers have not been added correctly due to a database error.
  - There was no response (ping) from the computers after execution.
- Configuration file:
  - machinebatch.txt

### 7.1.4 SatBatchImporterServiceTest.php

- Purpose:
  - Tests the correct function of batch importing a file of new services into the database.
- Pre-condition:
  - The services do not exist in the database.
- Post-condition:
  - Test fails if the services cannot be added to database.
- Configuration file:
  - servicebatch.txt

### 7.1.5 SatBatchImporterUserTest.php

- Purpose:
  - Tests the correct function of batch importing a file of new users into a database. The users can be added with corresponding roles, which is a list of permissions for each of them.
- Pre-condition:
  - The users do not exist in the database.
  - The role must exist in the database.
- Post-condition:
  - Test fails if the users are not added correctly into the database.
  - Test fails if the users are not given the correct roles.
- Configuration file:
  - userbatch.txt

### 7.1.6 SatGenpassTest.php

- Purpose:
  - Checks the correct function of the ITSAT pass generator. An ITSAT user can create a specific account password or use the random password generator which creates the password according to the TSIM policy available on the website (and once created, encodes it into a file).
- Pre-condition: -
- Post-condition:
  - Test fails if the created password pattern does not match with the TSIM policy.
  - Test fails if the file for storing passwords is not correctly created or the password is not correctly added to the file.
  - Test fails if the encoding of the passwords does not work properly.
- Configuration file:
  - genpass.conf

### 7.1.7 SatMachineAddTest.php

- Purpose:
  - Checks the correct function of adding a computer to a work environment. It is similar to SatBatchImporterMachineTest.php but instead of using a batch file, this test obtains the information from a computer manually by input.
- Pre-condition:

    - The indicated work environment must exist for the new computers.
    - The computers must be turned on.
    - The computers are not in a database.
- Post-condition:
    - Test fails if the computers have not been added correctly due to database error.
    - There was no response (ping) from the computers after the execution.
- Configuration file:
    - machineadd.txt

### 7.1.8 SatMachineDelTest.php

- Purpose:
    - Checks the correct function of deleting a computer.
- Pre-condition:
    - The computers must exist. *
- Post-condition:
    - Test fails if the computers cannot be deleted from the database
- Configuration file:
    - machinedel.txt

*NOTE: Test does not fail if the computer does not exist previously in the database, so it is very important to make sure that the computer DOES EXIST in the database beforehand.

### 7.1.9 SatRemotedo_CscriptTest.php

- Purpose:
    - Checks the correct function of sat-remotedo with flag “-C”, which executes a script in an indicated computer.
- Pre-condition:
    - The computer which executes the script must be turned on.
- Post-condition:
    - Test fails if the script was not correctly executed.
- Configuration file:
    - satremotedoscript.conf

### 7.1.10 SatRemotedo_MTest.php

- Purpose:
    - Checks the correct function of sat-remotedo with flag “-M”, which shows a list of the computers installed.
- Pre-condition: -
- Post-condition:
    - Test fails if the output list is not correct.
- Configuration file:
    - satremotedom.conf

### 7.1.11 SatRemotedo_RTest.php

- Purpose:

    - Checks the correct function of sat-remotedo with flag “-R”, which executes a certain command remotely in a computer with root as user.
- Pre-condition:
    - The specified computer must exist and be turned on.
- Post-condition:
    - Test fails if the command output does not match with the expected output.
- Configuration file:
    - satremotedor.conf

### 7.1.12 SatRemotedo_STest.php

- Purpose:
    - Checks the correct function of sat-remotedo with flag “-S”, which shows a list of the services installed.
- Pre-condition: -
- Post-condition:
    - Test fails if the output list is not correct.
- Configuration file:
    - satremotedos.conf

### 7.1.13 SatRemotedo_UTest.php

- Purpose:
    - Checks the correct function of sat-remotedo with flag “-U”, which executes a certain command remotely in a computer being a specific user.
- Pre-condition:
    - The specified computer must exist and be turned on.
- Post-condition:
    - Test fails if the command output does not match the expected output.
- Configuration file:
    - satremotedou.conf

### 7.1.14 SatRemotedo_cTest.php

- Purpose:
    - Checks the correct function of sat-remotedo with flag “-c”, which executes a certain command remotely.
- Pre-condition:
    - The specified computer must exist and be turned on.
- Post-condition:
    - Test fails if the command output does not match with the expected output.
- Configuration file:
    - satremotedoc.conf

### 7.1.15 SatRemotedo_ikTest.php

- Purpose:
    - Checks the correct function of sat-remotedo with flag “-l”, which shows a list of the instances installed, and “-k”, which shows a list of commands of that same instance installed.

- Pre-condition: -
- Post-condition:
    - Test fails if either the instance list is incorrect or the command list is incorrect.
- Configuration file:
    - instancebatch.txt
    - servicebatch.txt

### 7.1.16 SatRemotedo_ikxTest.php

- Purpose:
    - Checks the correct function of sat-remotedo with the flag "-l", "-k" and "-x". Unlike the previous test, this checks the execution of a specified instance and a specified command.
- Pre-condition:
    - The instance and the command must exist in the database.
- Post-condition:
    - Test fails if the command was not correctly executed.
- Configuration file:
    - instancebatch.txt

### 7.1.17 SatRemotedo_oTest.php

- Purpose:
    - Checks the correct function of sat-remotedo with the flag "-o", which saves the output of any other execution of sat-remotedo function into a file.
- Pre-condition: -
- Post-condition:
    - Test fails if the file is not correctly created.
- Configuration file:
    - satremotedoo.conf

### 7.1.18 SatRemotedo_pTest.php

- Purpose:
    - Checks the correct function of sat-remotedo with the flag "-p", which shows a list of the production environment.
- Pre-condition: -
- Post-condition:
    - Test fails if the list is incorrect.
- Configuration file:
    - satremotedop.conf

### 7.1.19 SatRemotedo_skTest.php

- Purpose:
    - Checks the correct function of sat-remotedo with the flag "-s", which shows a list of the services installed, and "-k", which shows a list of commands installed in that service.
- Pre-condition:
    - The service and the related command both exist in the database.

- Post-condition:
    - Test fails if the output of the command does not match with the expected output.
- Configuration file:
    - instancebatch.txt
    - servicebatch.txt

### 7.1.20 SatRemotedo_wTest.php

- Purpose:
    - Checks the correct function of sat-remotedo with the flag "-w", which shows a list of the work environments.
- Pre-condition: -
- Post-condition:
    - Test fails if the list shown is incorrect.
- Configuration file:
    - satremotedow.conf

### 7.1.21 SatRemotedocountTest.php

- Purpose:
    - Checks the correct function of sat-remotedo with the count option. As its name suggests, it counts the number of computers that meet the filter requirements.
- Pre-condition:-
- Post-condition:
    - Test fails if the result is incorrect.
- Configuration file:
    - satremotedocount.conf

### 7.1.22 SatRemotedogetdTest.php

- Purpose:
    - Checks the correct function of sat-remotedo with the getd option. It puts all the indicated files into a .tar extension file in the remote computer and transfers it to the current computer.
- Pre-condition:
    - The origin computer must be turned on.
    - The indicated files must exist in the origin computer.
- Post-condition:
    - Test fails if the .tar file cannot be transferred correctly.
    - Test fails if the content of the transferred file does not match with that expected.
- Configuration file:
    - satremotedogetd.conf

### 7.1.23 SatRemotedogetdzTest-php

- Purpose:
  - Checks the correct function of sat-remotedo with the getdz option. This function compresses the .tar extension file, which is described in the previous test, in the remote computer. If no compression function can be found in the remote computer, it will first transfer the tar file and then compress it in the current computer.
- Pre-condition:
  - The origin computer must be turned on.
  - The indicated files must exist in the origin computer.
- Post-condition:
  - Test fails if the compressed/tar file cannot be transferred correctly.
  - Test fails if the content of the transferred file does not match with that expected.
- Configuration file:
  - satremotedogetd.conf

### 7.1.24 SatRemotedogetfTest.php

- Purpose:
  - Checks the correct function of sat-remotedo with the getf option. This function gets the entire indicated directory and transfers it to the local computer.
- Pre-condition:
  - The origin computer must be turned on.
  - The indicated directory must exist in the origin computer.
- Post-condition:
  - Test fails if the directory cannot be transferred correctly.
  - Test fails if the content of the transferred directory does not function properly or the content is different from the original.
- Configuration file:
  - satremotedoputx.conf

### 7.1.25 SatRemotedogetfzTest.php

- Purpose:
  - Checks the correct function of sat-remotedo with the getfz function. Similar to getf, this function gets the entire indicated directory and compresses it. Then it transfers the compressed version to the local computer.
- Pre-condition:
  - The origin computer must be turned on.
  - The indicated directory must exist in the origin computer.
- Post- condition:
  - Test fails if the directory cannot be transferred correctly.
  - Test fails if compression fails in the remote computer.
- Configuration file:
  - satremotedoputx.conf

### 7.1.26 SatRemotedoputTest.php

- Purpose:
  - Checks the correct function of sat-remotedo with the put and putf function, since both are the same. This function distributes a file or directory in the remote computers.
- Pre-condition:
  - The file or directory must exist in the local computer.
  - The remote computers must be turned on.
- Post-condition:
  - Test fails if the file cannot be transferred correctly.
- Configuration file:
  - satremotedoput.conf

### 7.1.27 SatRemotedoputxTest.php

- Purpose:
  - Checks the correct function of sat-remotedo with the putx function. This function distributes a compressed file in all the remote computers.
- Pre-condition:
  - The compressed file must exist.
  - The remote computers must be turned on.
  - The destination directory must exist.
- Post-condition:
  - Test fails if the compressed file cannot be transferred correctly.
- Configuration file:
  - satremotedoputx.conf

### 7.1.28 SatRemotedoputzTest.php

- Purpose:
  - Checks the correct function of sat-remotedo with the putz function. This function is similar to the putx function, but it decompresses the sent file at the remote directory.
- Pre-condition:
  - The compressed file must exist.
  - The remote computers must be turned on.
  - The destination directory must exist.
- Post-condition:
  - Test fails if the compressed file cannot be transferred correctly.
  - Test fails if the file cannot be decompressed in the remote directory.
- Configuration file:
  - satremotedoputz.conf

### 7.1.29 SatUserAddTest.php

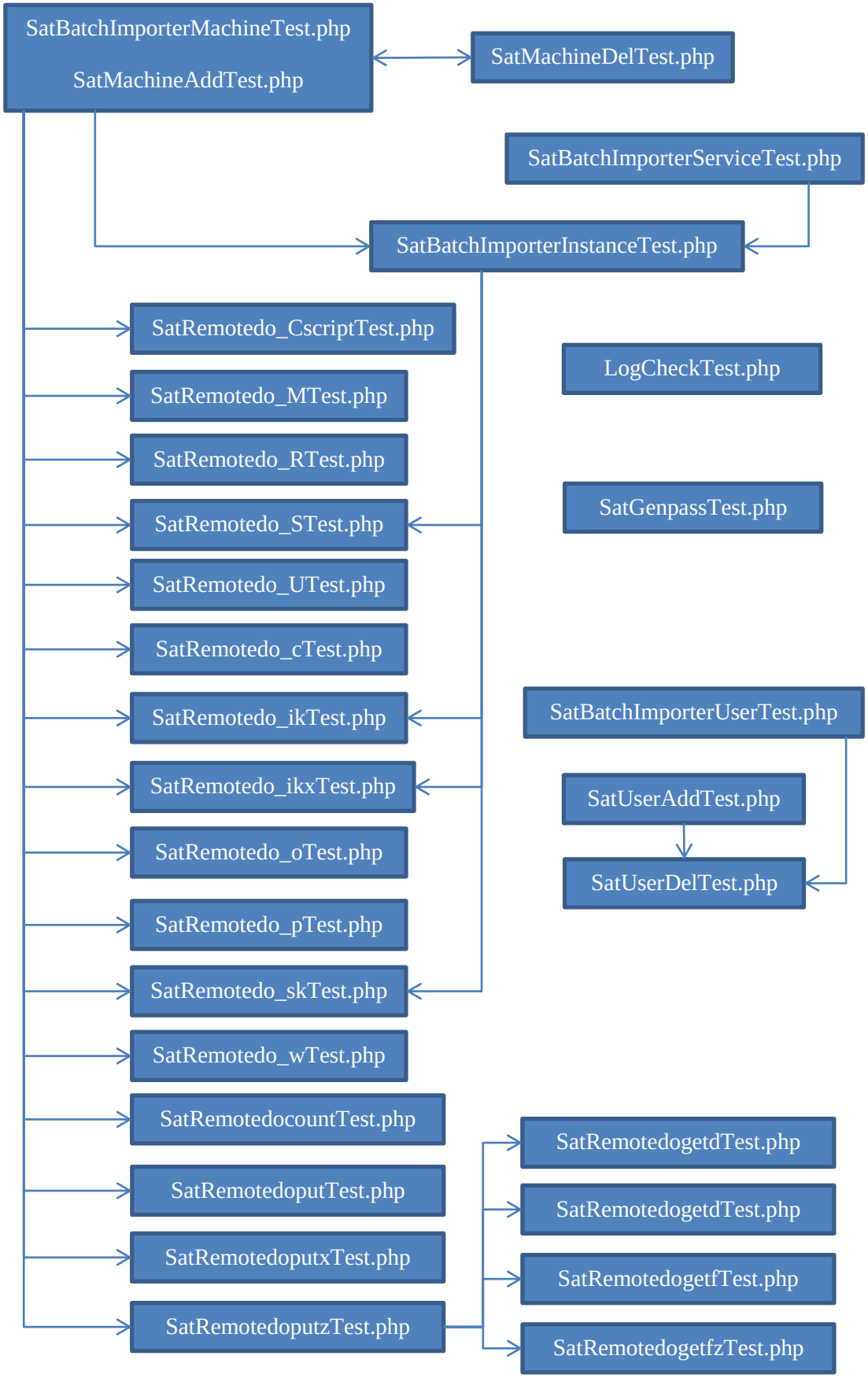
- Purpose:
  - Checks the correct function of adding a user to the database similar to SatBatchImporterUser, but instead of using a file, it adds one user at a time through command.

- Pre-condition:
    - The user must not exist previously in the database.
    - The role of the user must exist.
- Post-condition:
    - Test fails if the users are not added correctly into database.
    - Test fails if the users are not given the correct roles.
- Configuration file:
    - useradd.txt

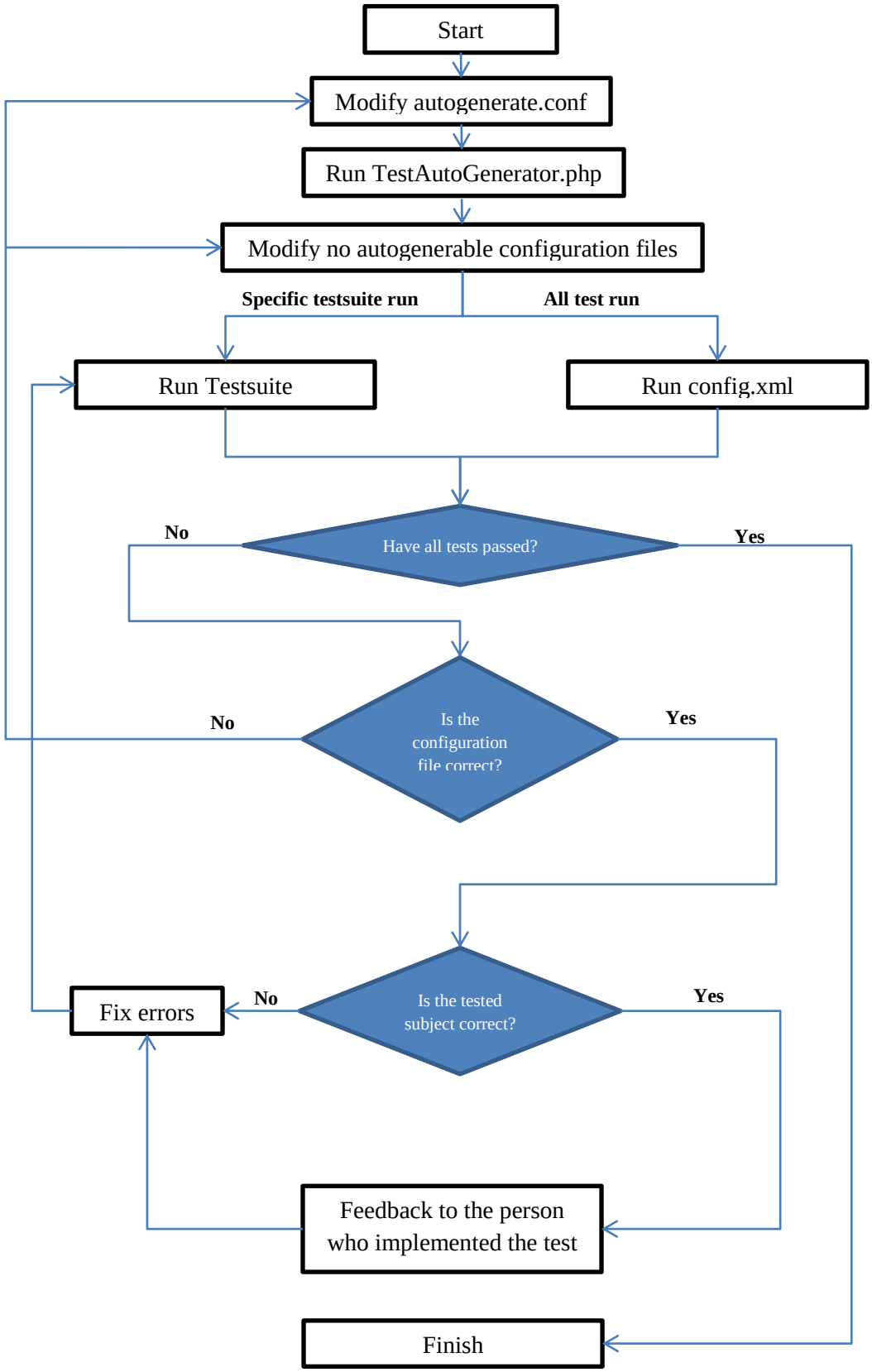
#### 7.1.30 SatUserDelTest.php

- Purpose:
    - Checks the correct function of deleting a user from database. This test deletes the users which have been added by SatBatchImporterUserTest and SatUserAddTest.
- Pre-condition:
    - Must be run after SatBatchImporterUserTest and SatUserAddTest.
- Post-condition:
    - Test fails if the users are not correctly removed from the database.
- Configuration file:
    - userbtach.txt
    - useradd.txt

### 7.1.31 Test dependency diagram

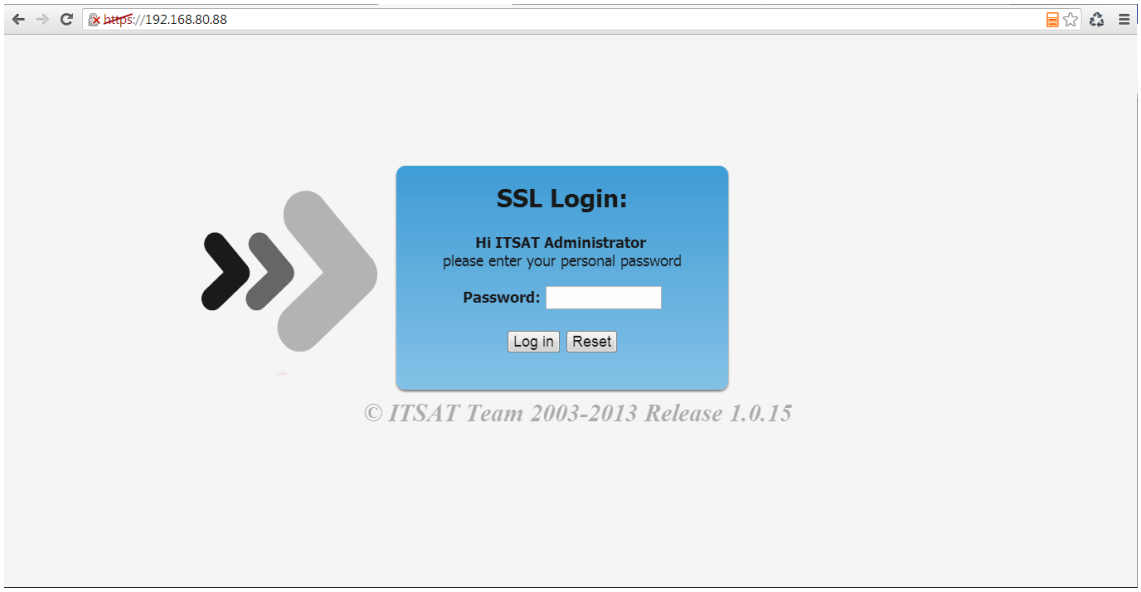


### 7.1.32 Testing procedure

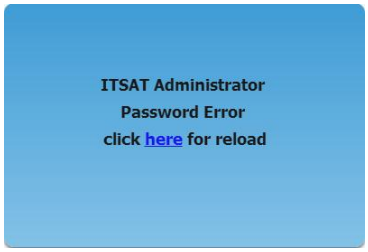


## 7.2 Web testing files with Selenium IDE/RC

### 7.2.1 Login

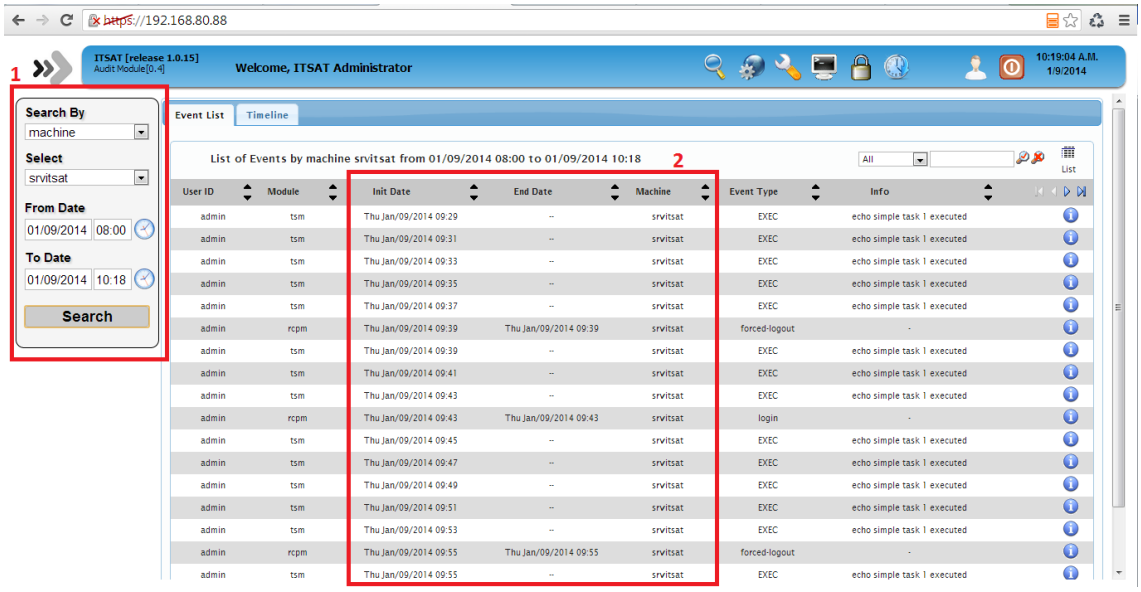



- Test with a wrong password input.

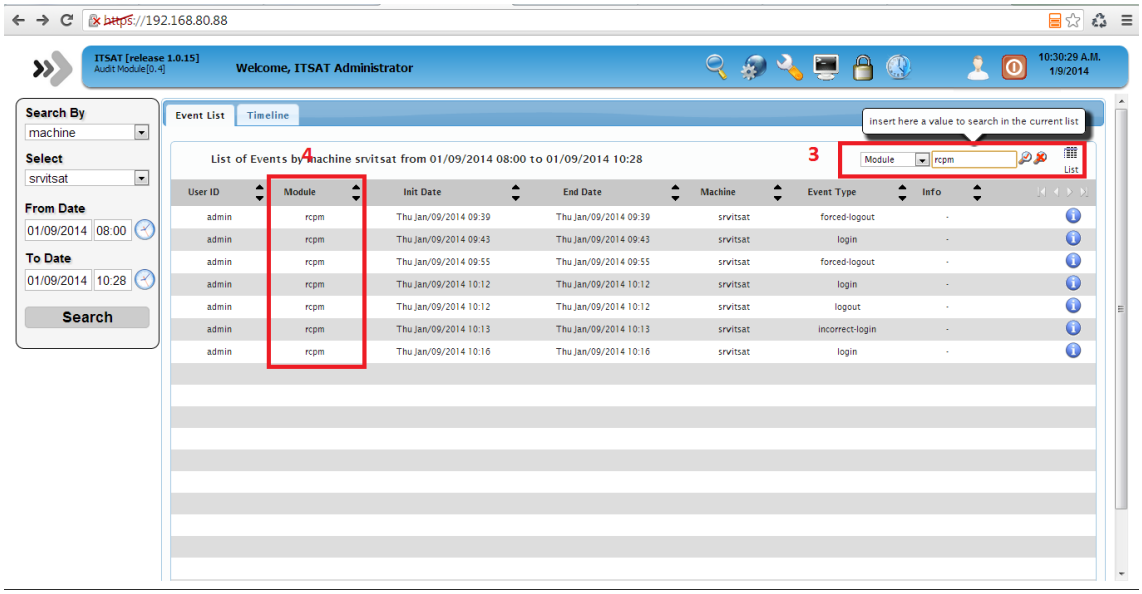



- Test the link works properly.
- Test with the correct password input.

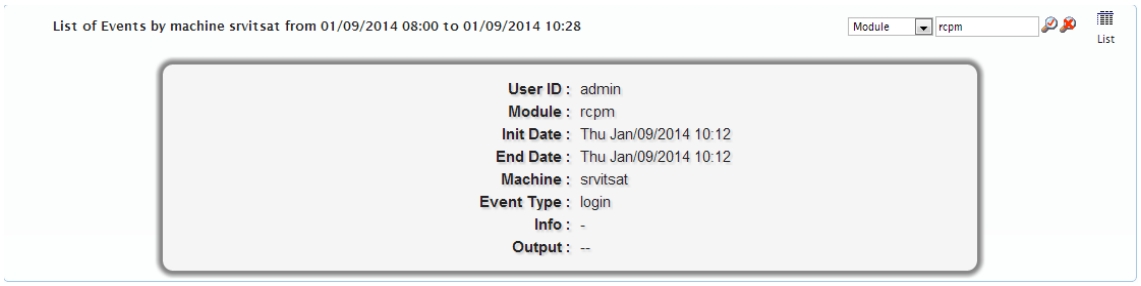
## 7.2.2 AUDM Module

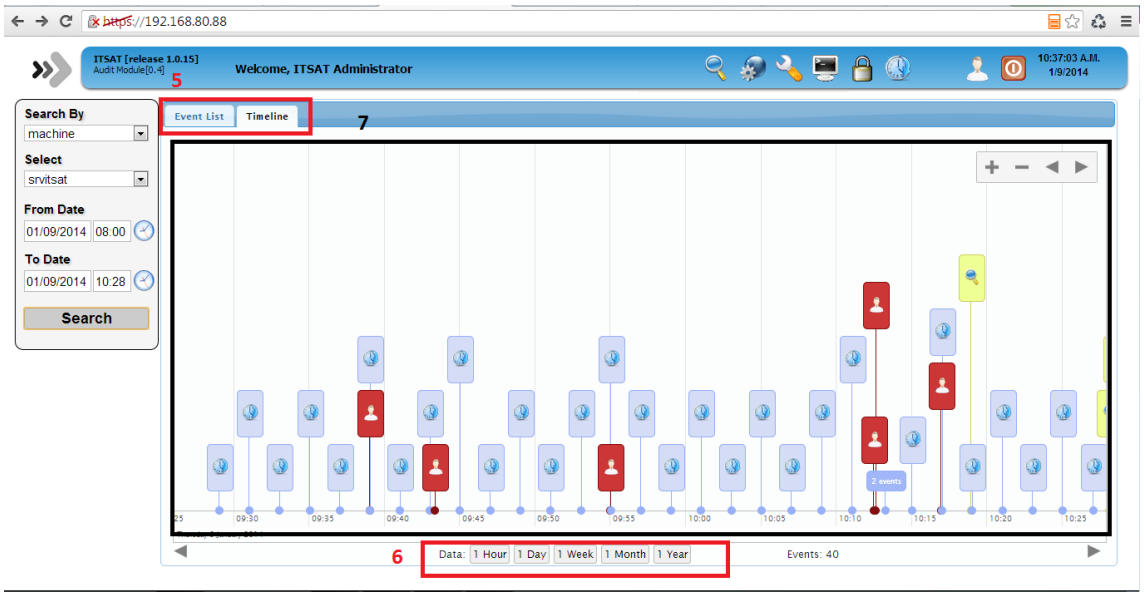



- Test the filter allows all possible combinations. [1]
- Test the result representation in the table corresponds to the filter information. [1][2]




- Test the filter of the results is functioning properly. [3][4]
- Test the info key is functioning properly and can show the corresponding information.

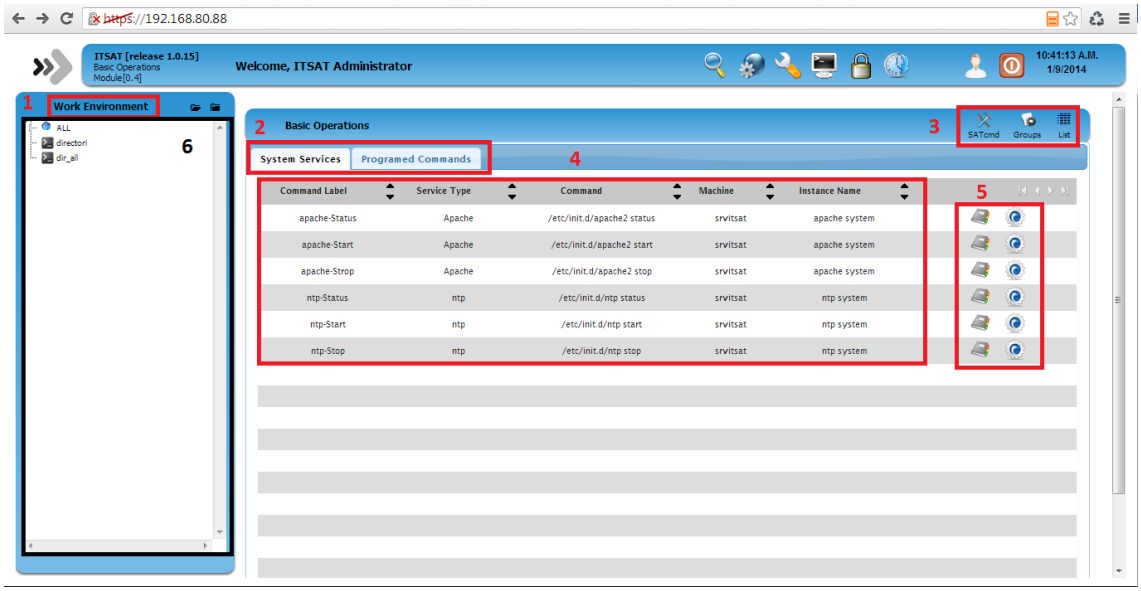





- Test the tabs can switch from one another. [5]
- Test the timeline buttons can be clicked. [6]

[7] The timeline itself cannot be tested.

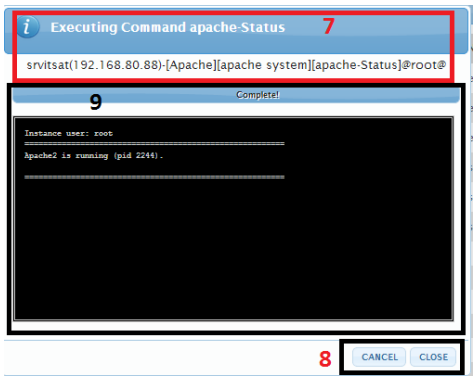
### 7.2.3 BOPM Module




- Check the keywords [1], the tabs [2], the BOPM option icons [3] and the corresponding history and execution icons of each operation [5] exist and function properly.
- Check the inserted basic options are present in the list [4].

[6] The tree cannot be tested.

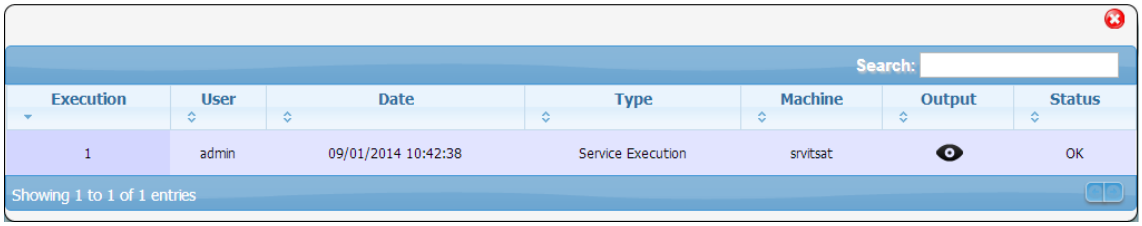
- Check the execution icon  functions properly and shows the execution pop-up.



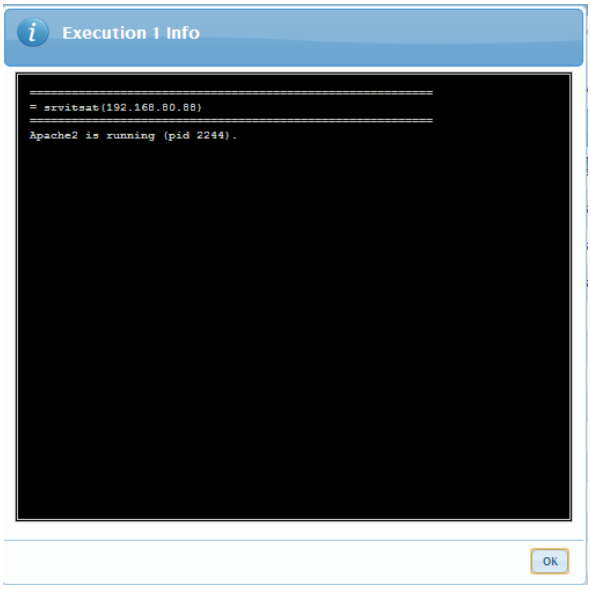

- Check that the pop-up shows the corresponding information. [7]
- Check that the pop-up has the “Cancel” and “Close” buttons and function properly. [8]

[9] The result of the execution cannot be tested.

- Check the history  icon functions properly and shows the history pop-up.




- Check that the information shown is correct.
- Check the view  icon can be clicked and shows the pop-up.




- Check that the pop-up contains the “OK” button.

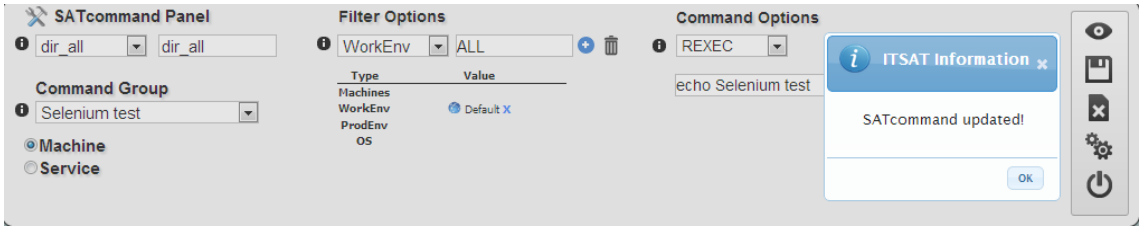
The log information itself cannot be tested.

- Check that the SATgroup edition panel can be accessed by clicking its icon. 

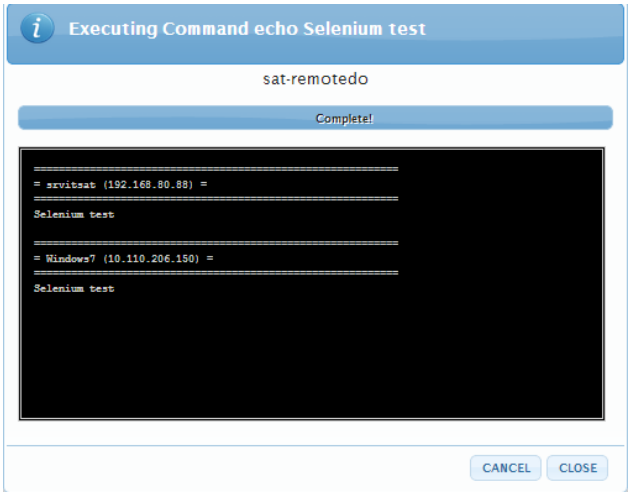


- Create a “Selenium test” group and check that it can be saved by clicking the save icon  and shows the pop-up with “OK” button and message “Group updated”.
- Check that the exit icon  closes the panel.

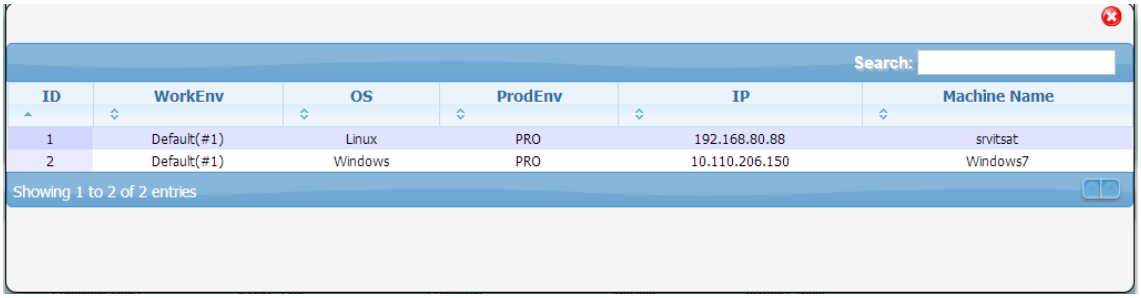
- Check that the SATcmd edition panel can be accessed by clicking its icon. 

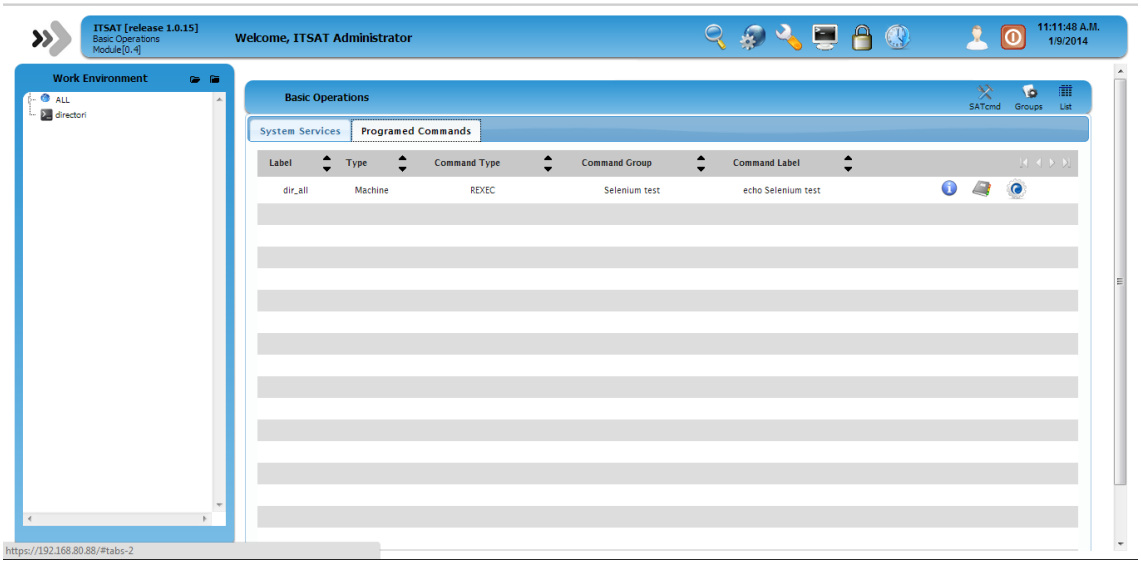

- Check that the filters allow all possible combinations.
- Check that the recently created group “Selenium test” is now available.
- Check that the choice between “Machine” and “Service” can change the “Filter Options” on the right.
- Check that after saving, the pop-up is correct.
- Check that the execution icon  functions properly and shows the pop-up.




- Check that with the view icon  the filtered results shown in the pop-up are correct.

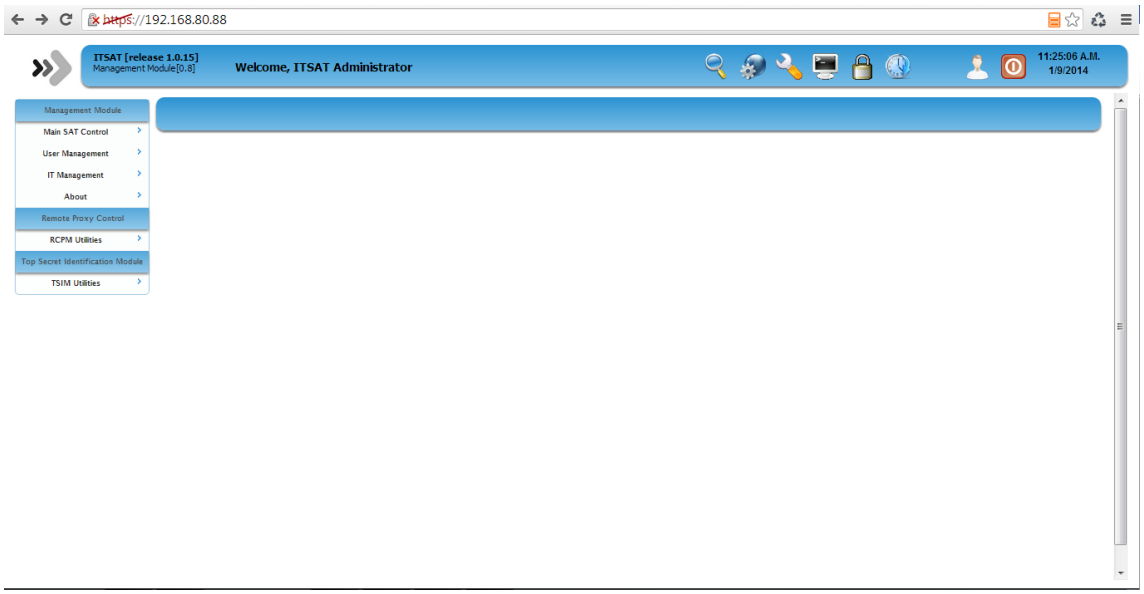



- Check that by switching to the tab “Programed Commands”, the newly saved command is available with its information.




- Check the three buttons (information, history and execution respectively) function properly.

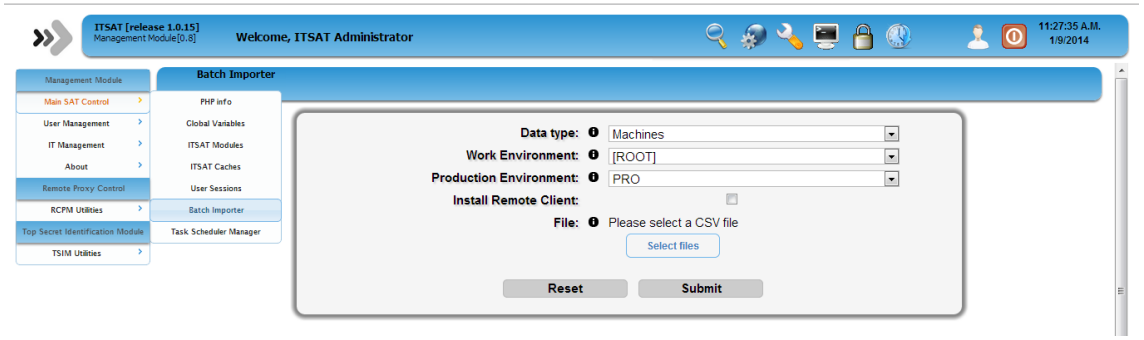
### 7.2.4 MNGM Module




- Check that all the basic information and basic operations provided on the left side are functioning properly.

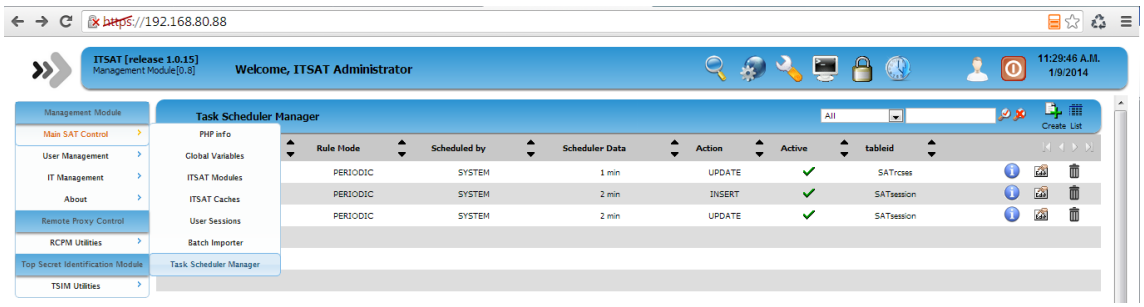
In particular:

**1. Main SAT Control -> Batch Importer:**



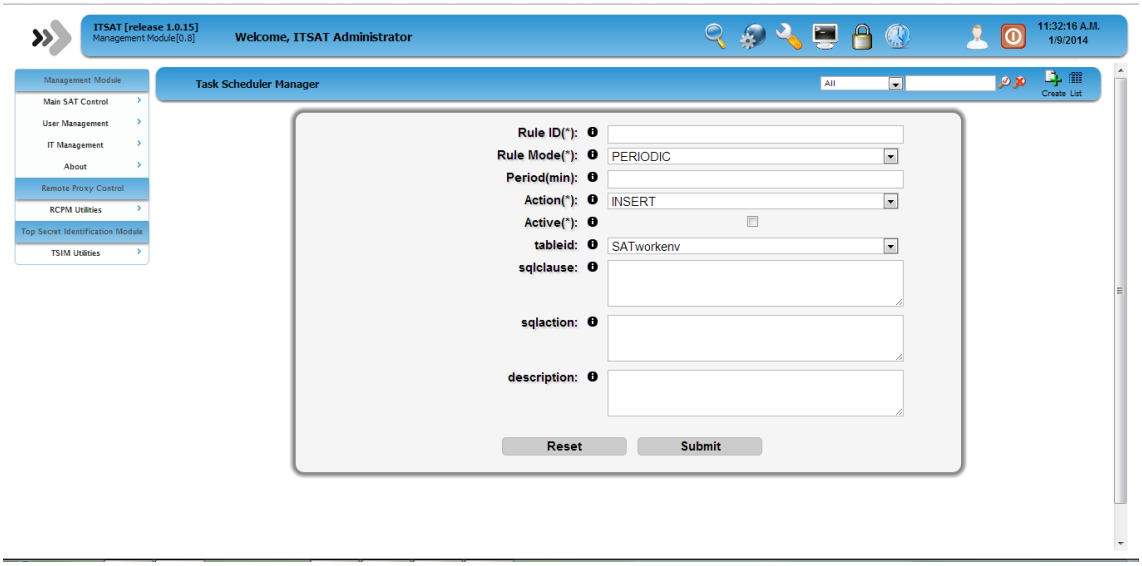

- Check that the filters can be correctly selected with their corresponding information.

**2. Main SAT Control -> Task Scheduler Manager:**



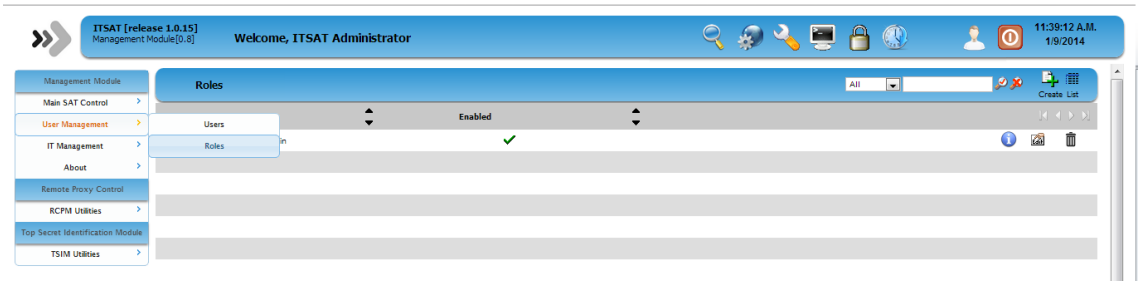

- Check that the three basic tasks are shown correctly.
- Create a new task 

and check that the filter is functioning properly.

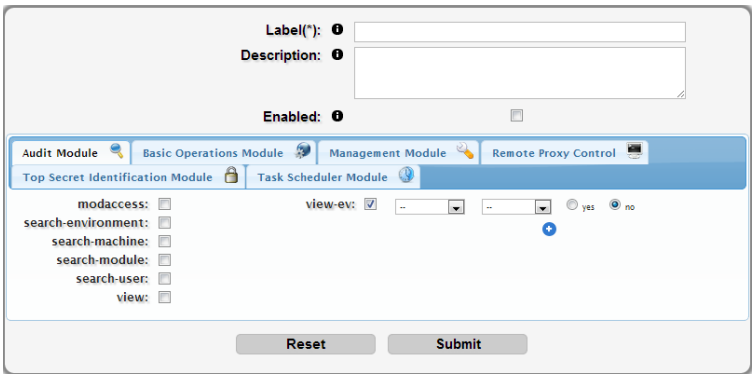



- Check a variety of combinations of filters and check that the corresponding options are shown and the others are hidden.
- Check the three options beside each task (information, modification and deletion respectively) are functioning properly.

### 3. User Management -> Roles:

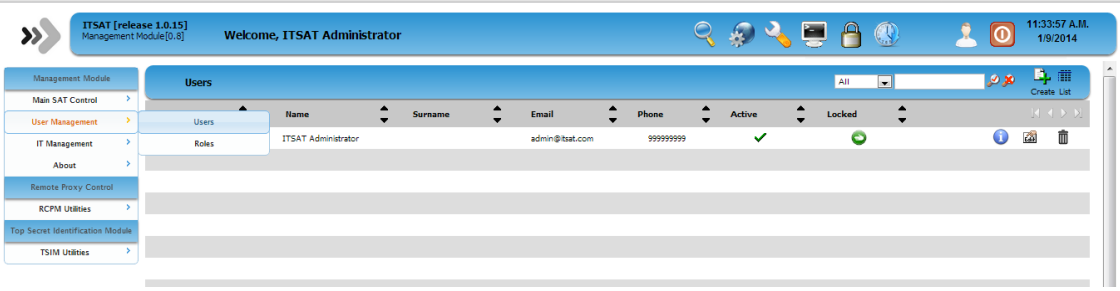



- Check the same procedure as Task Scheduler Manager with its corresponding creation form.

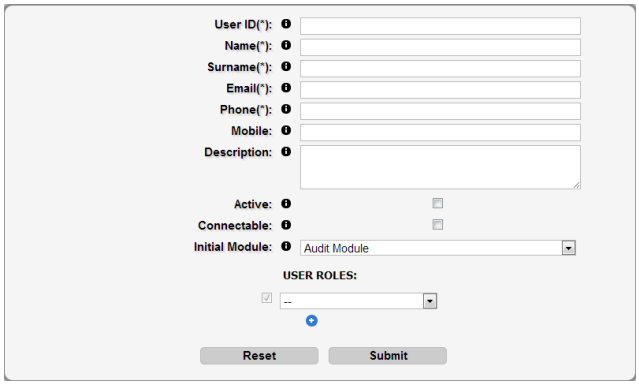



- Check that all tabs function properly and all permissions can be selected and edited.

**4. User Management -> Users:**

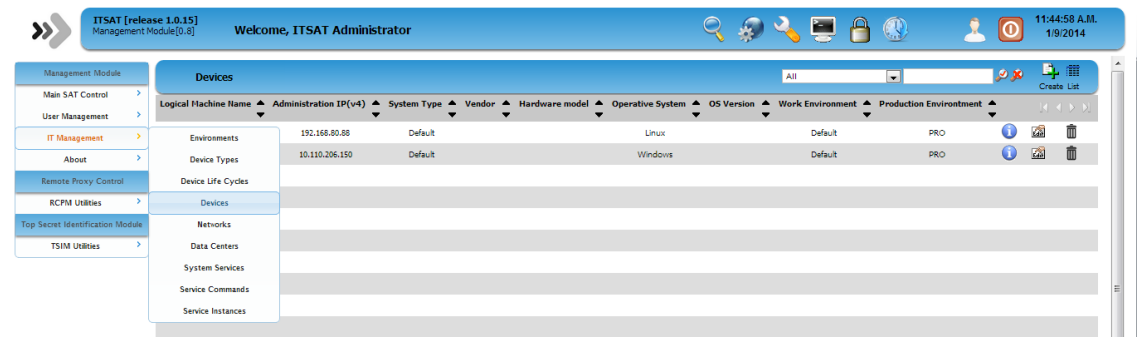



- Check the same procedure as the Task Scheduler Manager with its corresponding creation form.

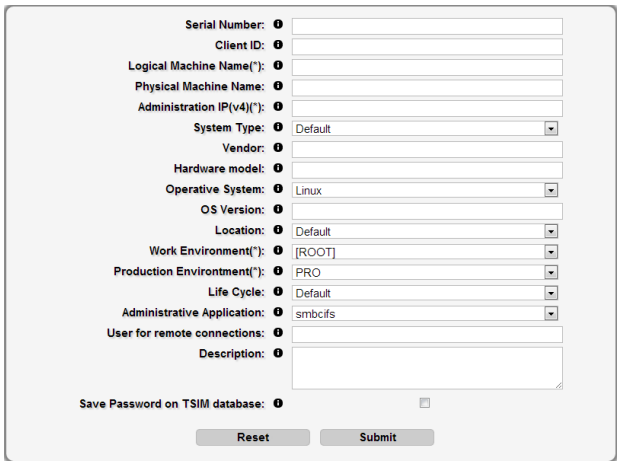



- Check that the recently created role can now be selected from the scrollbar of “User Roles:”
  The user’s login itself and its corresponding permission cannot be tested.

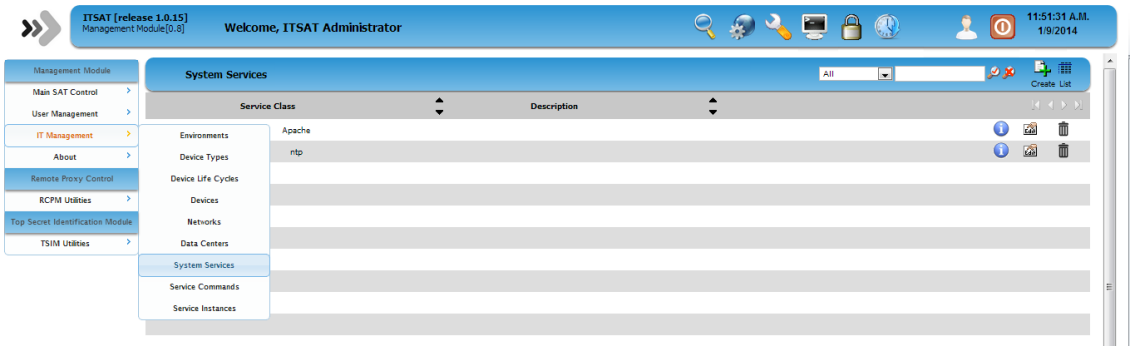
**5. IT Management -> Devices:**



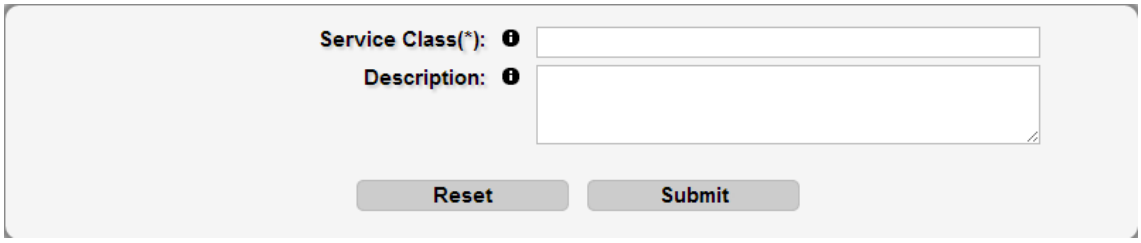

- The same check procedure as Task Scheduler Manager with the creation form.



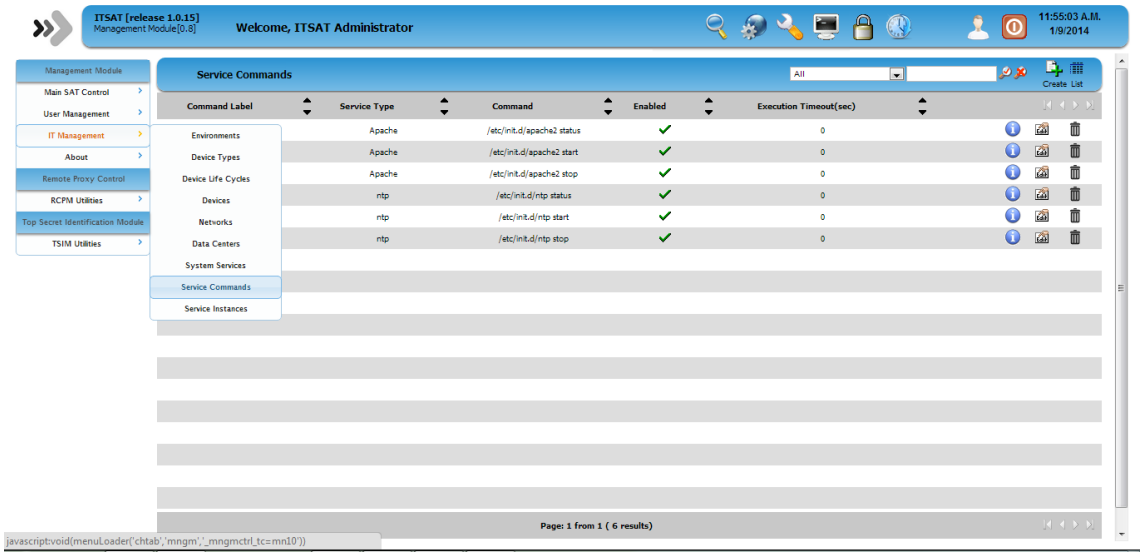

### 6. IT Management -> System Service:



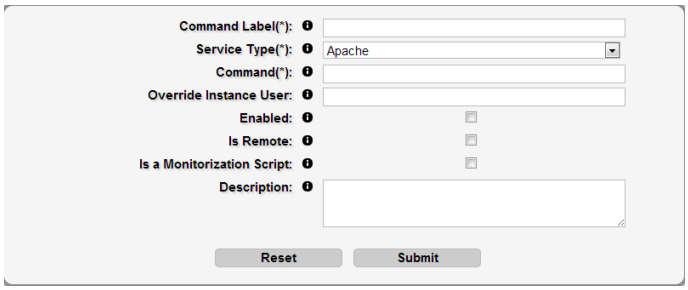

- The same check procedure as Task Scheduler Manager with the creation form.



### 7. IT Management -> Service Commands:

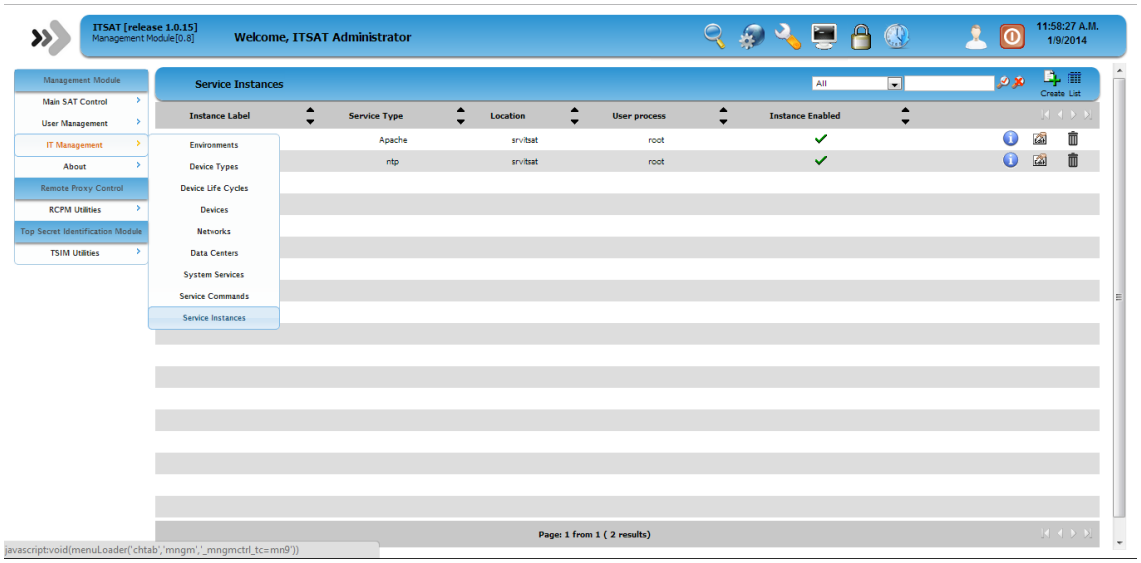



- The same check procedure as Task Scheduler Manager with the creation form.

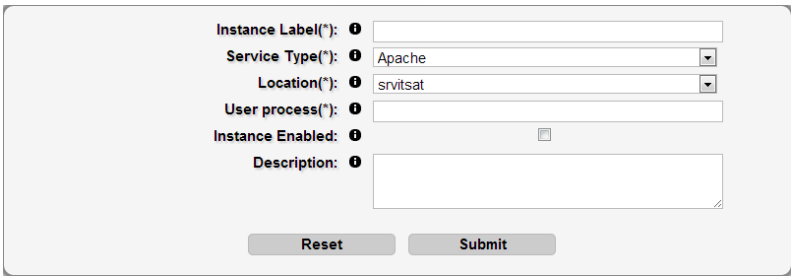



- Check that the recently created System Service and be chosen from the scrollbar of "Service Type".

### 8. IT Management -> Service Instances:

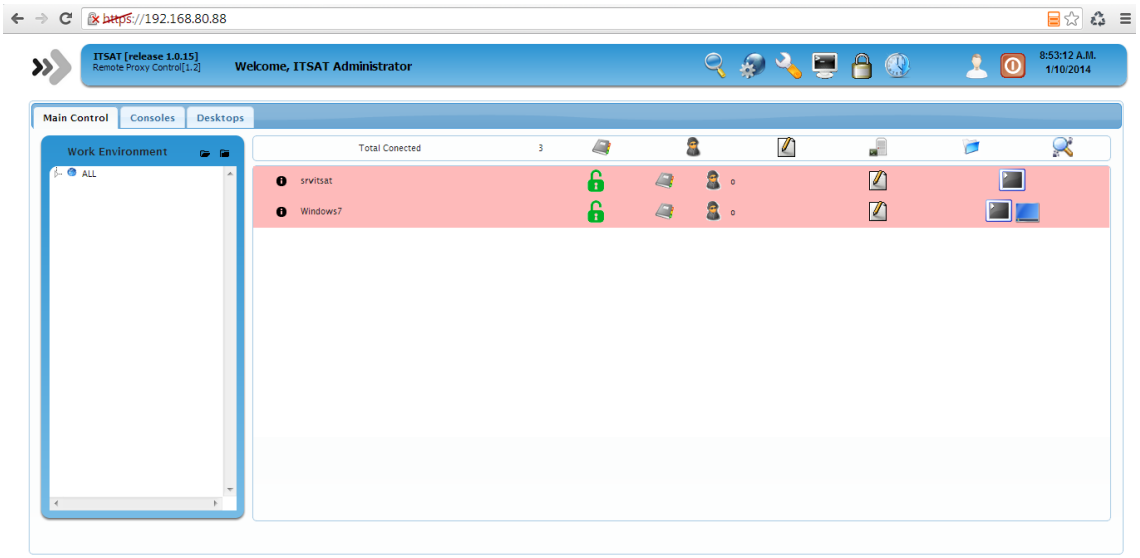


- The same check procedure as Task Scheduler Manager with the creation form.

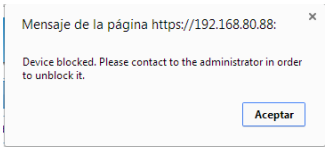



- Check that the recently created System Service and be chosen from the scrollbar of "Service Type" and the recently created computers in Devices can be chosen from the scrollbar of "Location".

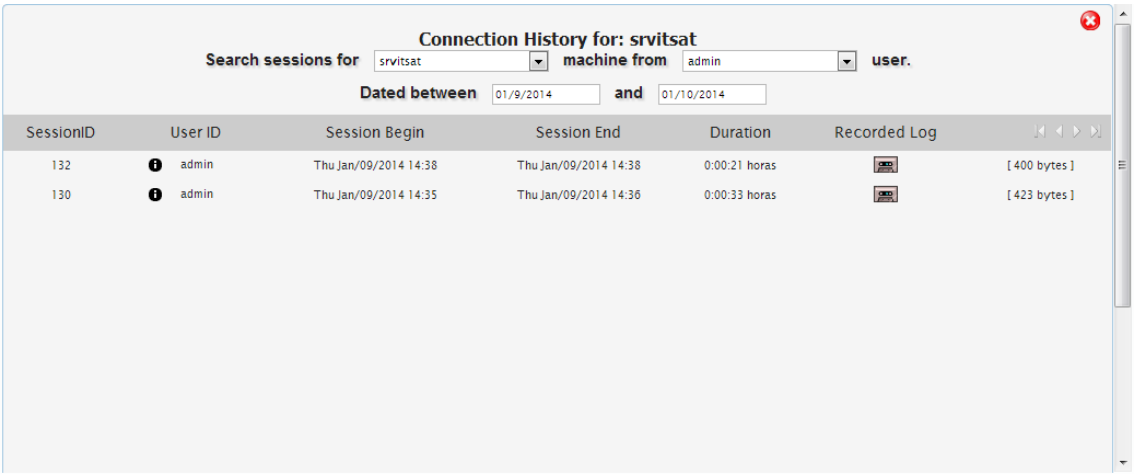
### 7.2.5 RCPM Module 

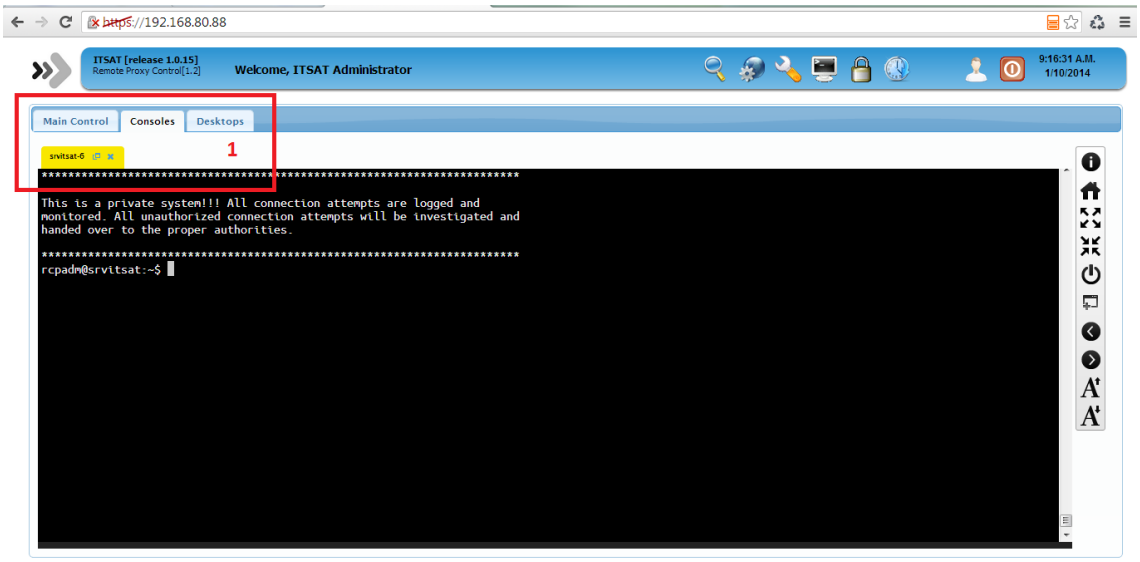



- Check all installed computers exist in the table.
- Check that all Linux computers only have a console icon  whereas the Windows computers can have another desktop icon 
- Check that for each computer, when the lock is locked  ->  , the connection to console or desktop is forbidden with this information on a pop-up.

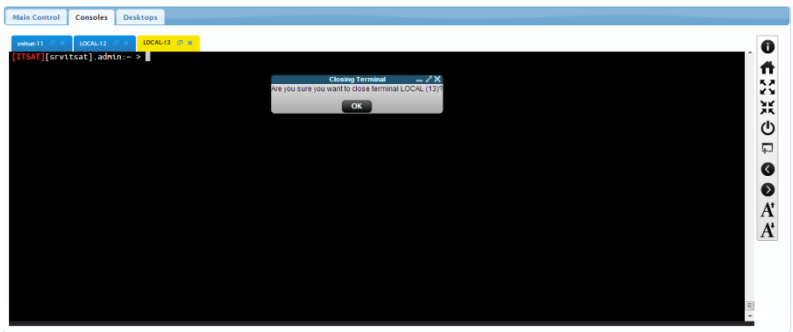



- Check that for each three options of computers (history of connected users  , connected people  and history of computer  ), the pop-up is correct.
- Check that the Finder icon  can show up a pop-up.
- Check that the session search  can be done successfully with different filter possibilities.

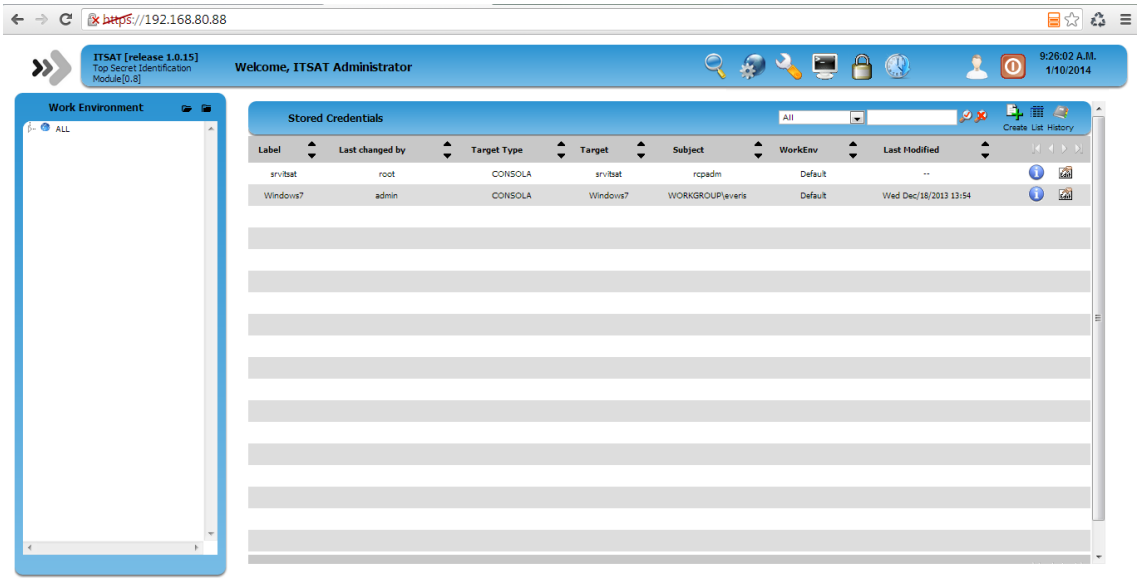



- Check that by clicking any console icon ( is for local console) and the tabs switch from “Main Control” to “Consoles” with a newly opened tab. [1]
- Check that by clicking either srvitsat-6 or or , a new tab is opened.
- Check that by clicking and , the switching between each tab is enabled.
- Check that by clicking either srvitsat-6 or , a pop-up of closing terminal will show and by clicking “OK”, the tab is closed.



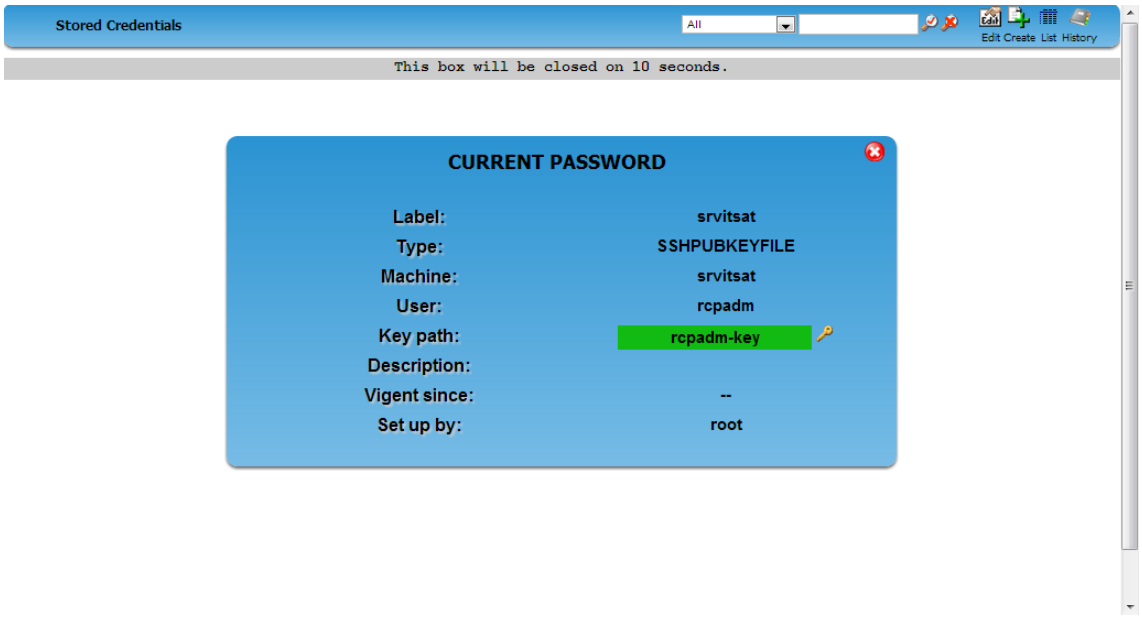
## 7.2.6 TSIM Module



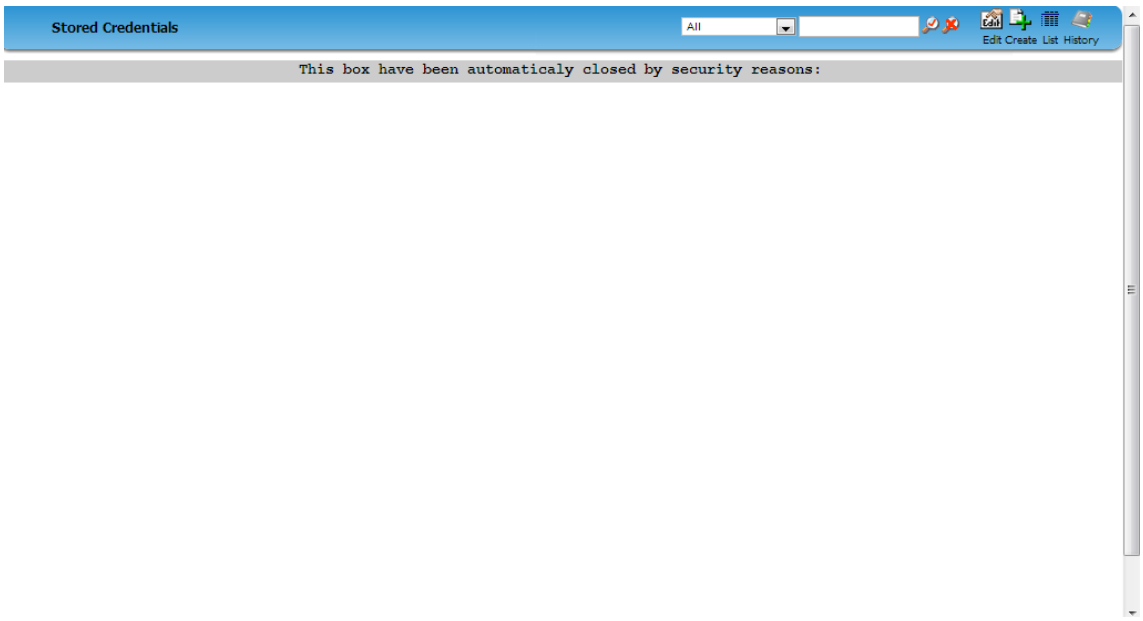

Before starting any testing procedures on this module, the script for changing the image verification detector must be run to fix the password detection issue for Selenium.

- Check that the filter works properly, similar to the process of MNGM.
- Check that by clicking the info icon , in order to view the credential, two passwords are needed.

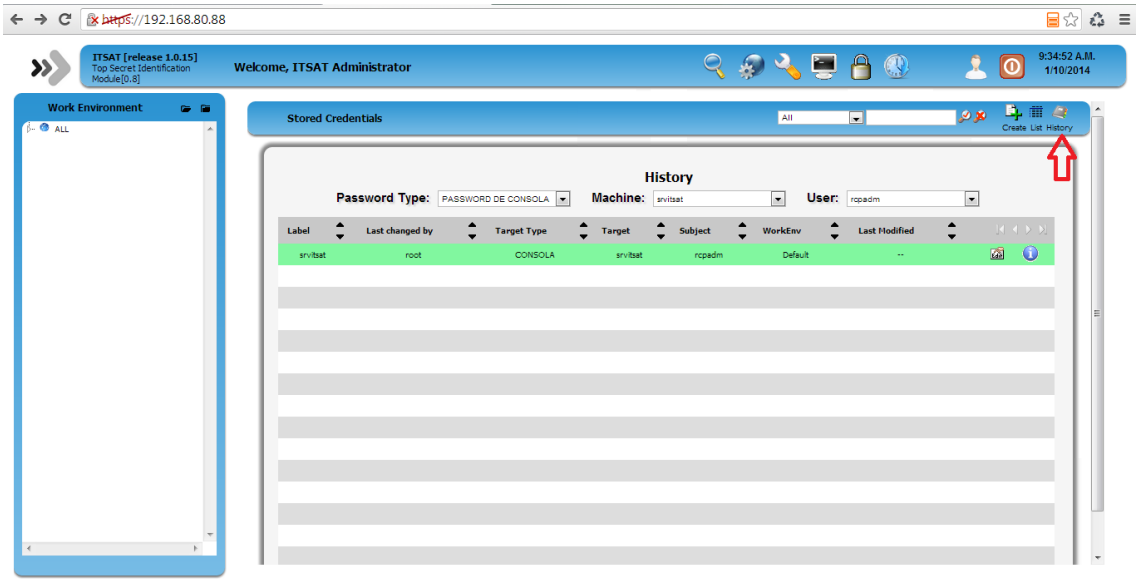

- Check that by inputting the password and then the word “Selenium”, the credential is shown. And if not or inputting with wrong passwords, a pop-up with password error is shown.



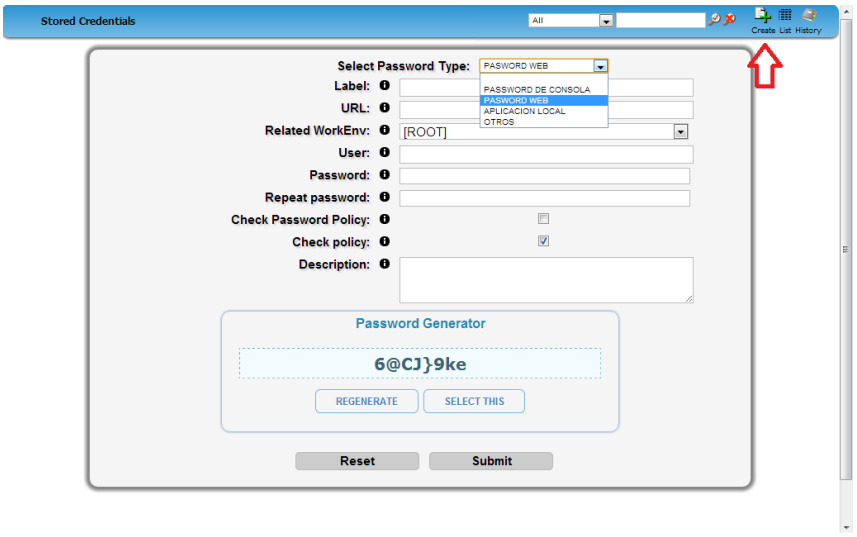

- Pause Selenium for 10 seconds and check that the table is hidden afterwards.




- Check that the history filter works properly with any combination of filter possibilities.

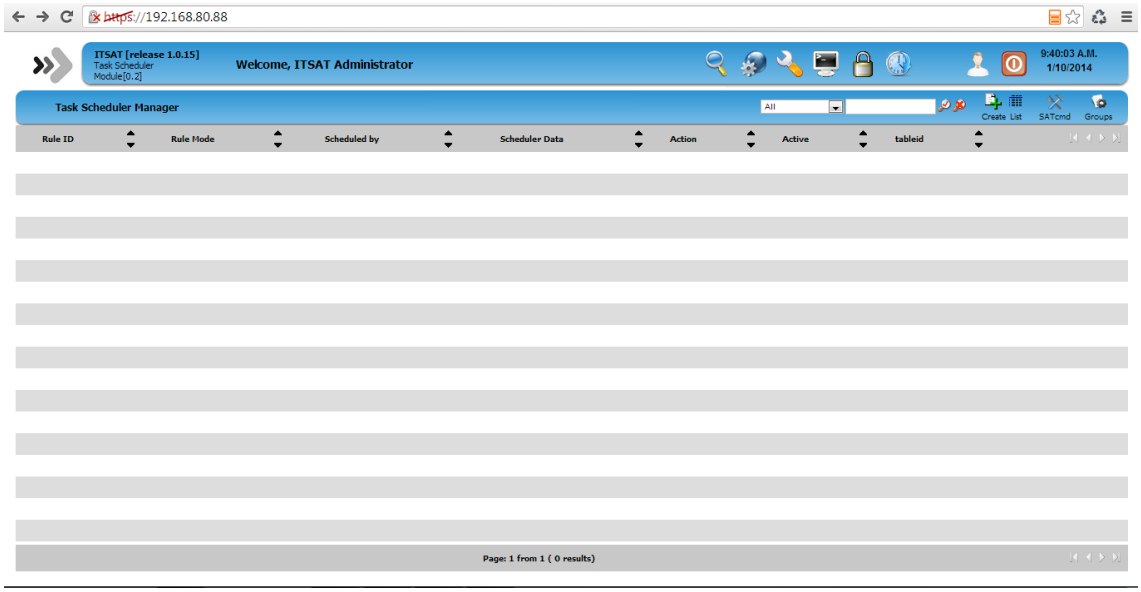



- Check that at creation of any credential, four options are given and for each option, check that the forms are different and show the required fields while hiding the irrelevant ones.



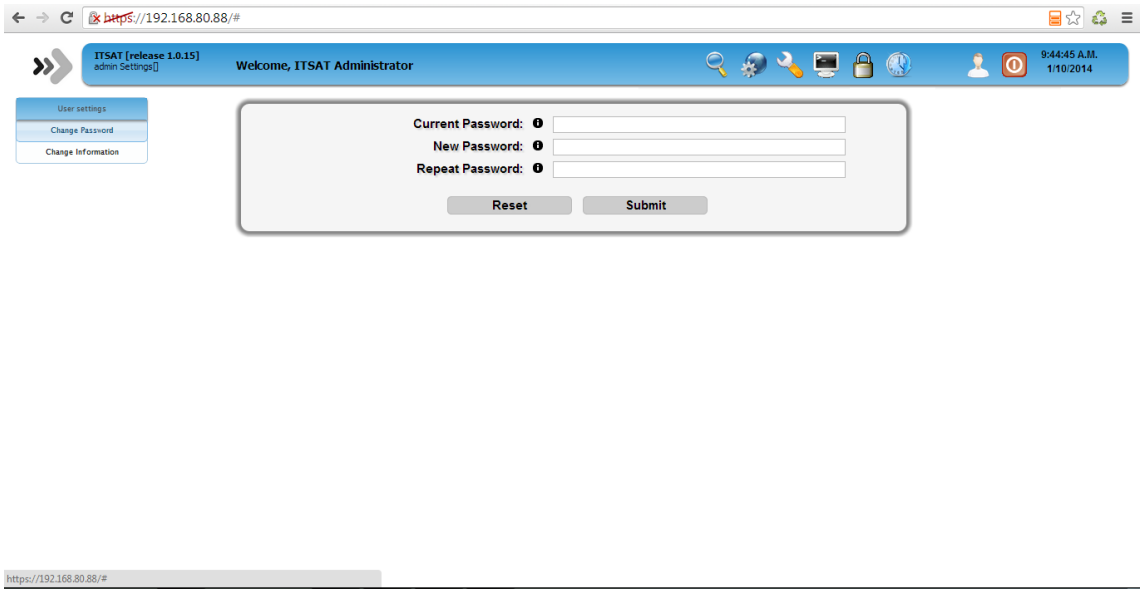

- Check that by clicking Submit, the new credential is visible from the main page table.

### 7.2.7 TSM Module

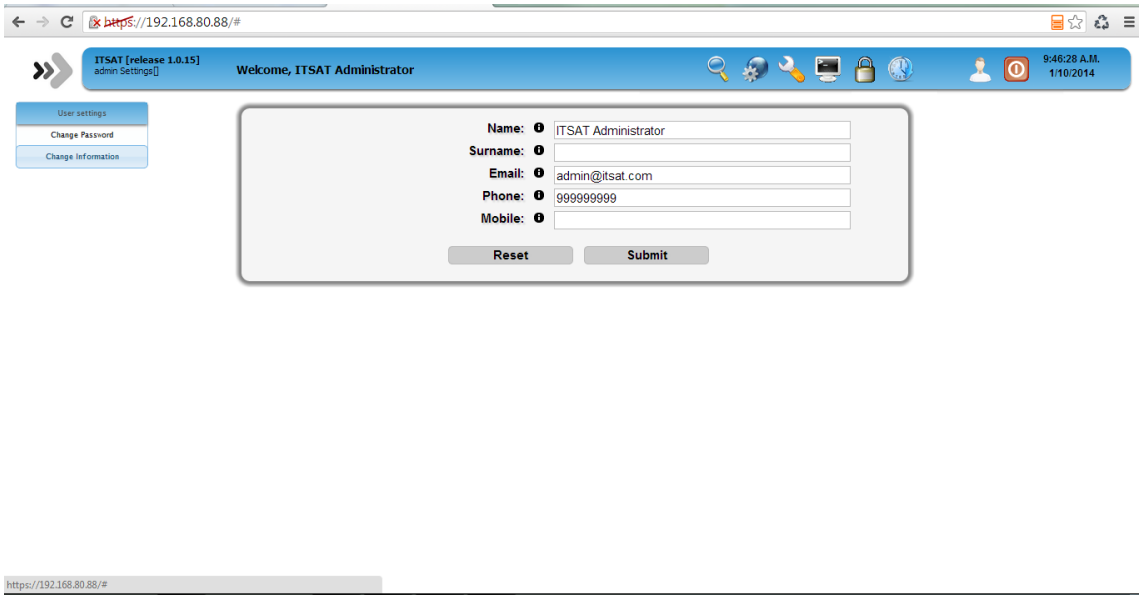



This module is a combination of “Task Scheduler Manager” from MNGM module and the control panel from BOPM module. The checking procedure is the same.

## 7.2.8 User Settings




- Check that the password can be changed and check the correct function by doing Logoff and Login.
  The Login will be another file with the changed password instead of the Login mentioned in section 7.2.1.



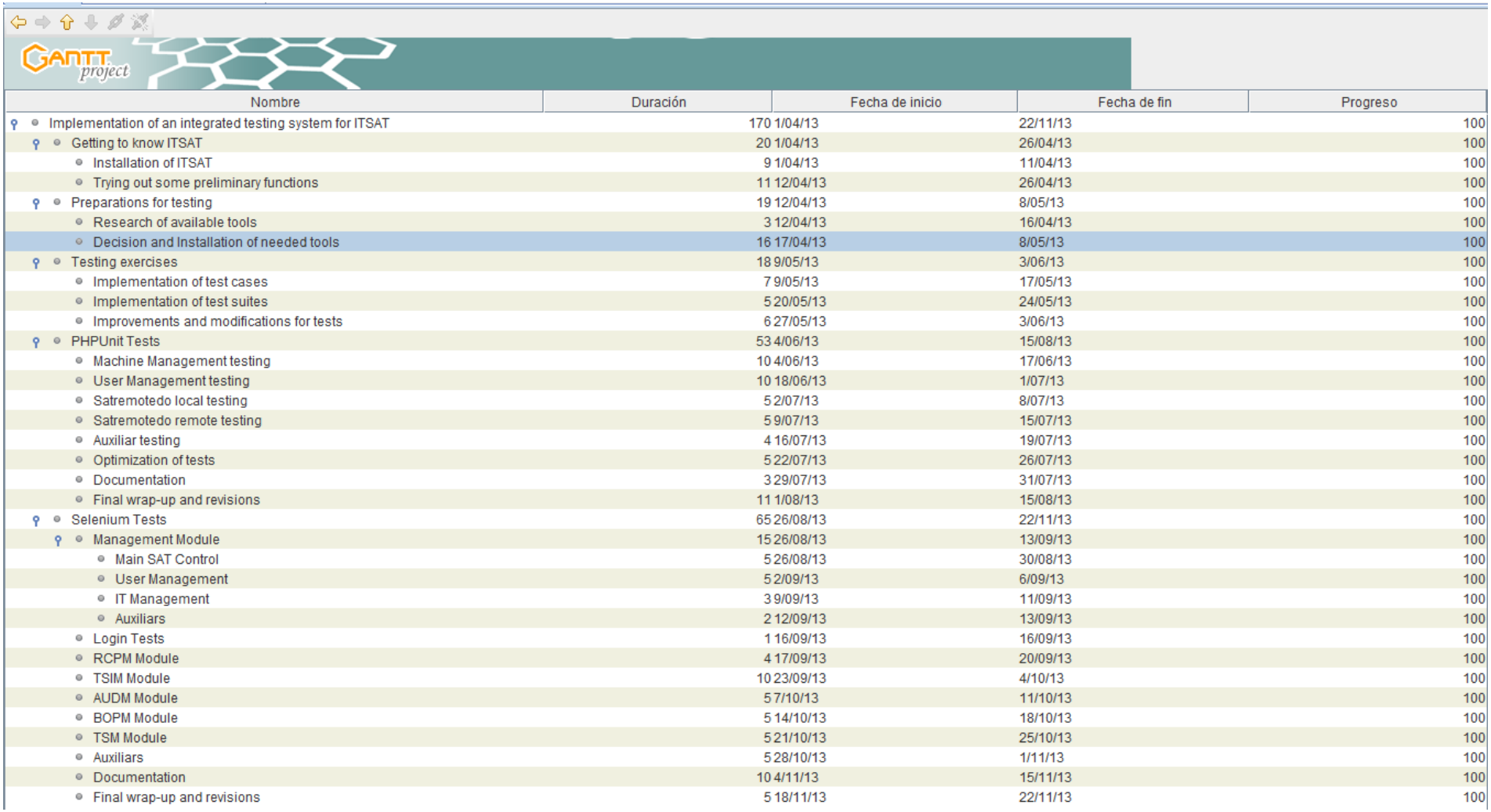

- Check that the user information can be changed correctly by looking at MNGM module through **Management Module -> User management -> Users**.

## 7.2.9 Logout

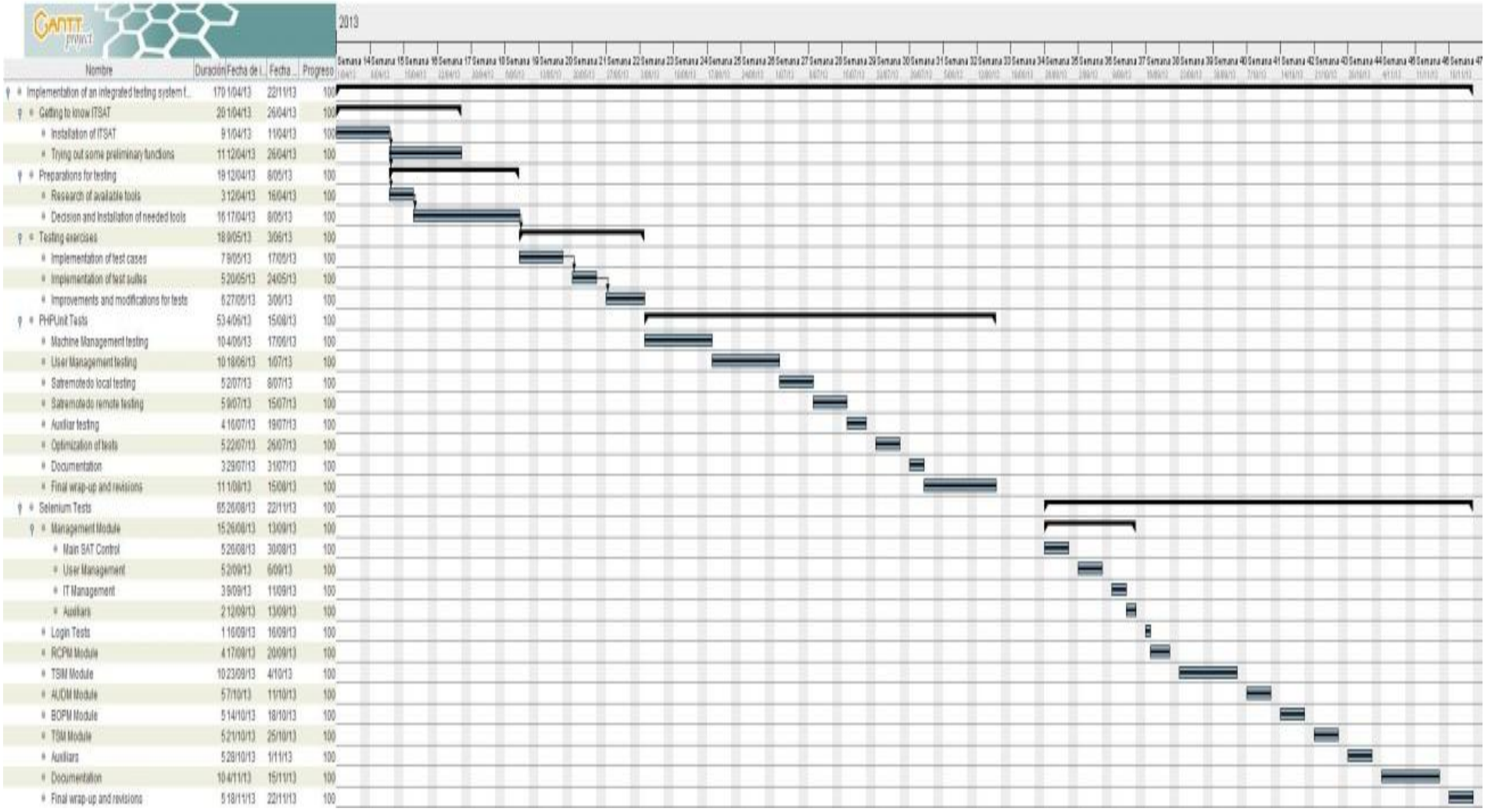
- Check that by simply clicking on the icon, the login page will be show again.

# 8. Final Planning

The tasks and duration of the whole project are shown below as a Gantt chart. Although the duration of the project has not changed compared to the initial plan, the tasks have been further divided into smaller and more specific subtasks. There have been minor changes to the priority of the tests so that the important ones were carried out first, but the overall schedule and timing were maintained so there were no changes to the deadline stated in the initial plan.



| Nombre | Duración | Fecha de inicio | Fecha de fin | Progreso |
|---|---|---|---|---|
| Implementation of an integrated testing system for ITSAT | 170 | 1/04/13 | 22/11/13 | 100 |
| Getting to know ITSAT | 20 | 1/04/13 | 26/04/13 | 100 |
| Installation of ITSAT | 9 | 1/04/13 | 11/04/13 | 100 |
| Trying out some preliminary functions | 11 | 12/04/13 | 26/04/13 | 100 |
| Preparations for testing | 19 | 12/04/13 | 8/05/13 | 100 |
| Research of available tools | 3 | 12/04/13 | 16/04/13 | 100 |
| Decision and Installation of needed tools | 16 | 17/04/13 | 8/05/13 | 100 |
| Testing exercises | 18 | 9/05/13 | 3/06/13 | 100 |
| Implementation of test cases | 7 | 9/05/13 | 17/05/13 | 100 |
| Implementation of test suites | 5 | 20/05/13 | 24/05/13 | 100 |
| Improvements and modifications for tests | 6 | 27/05/13 | 3/06/13 | 100 |
| PHPUnit Tests | 53 | 4/06/13 | 15/08/13 | 100 |
| Machine Management testing | 10 | 4/06/13 | 17/06/13 | 100 |
| User Management testing | 10 | 18/06/13 | 1/07/13 | 100 |
| Satremotedo local testing | 5 | 2/07/13 | 8/07/13 | 100 |
| Satremotedo remote testing | 5 | 9/07/13 | 15/07/13 | 100 |
| Auxiliar testing | 4 | 16/07/13 | 19/07/13 | 100 |
| Optimization of tests | 5 | 22/07/13 | 26/07/13 | 100 |
| Documentation | 3 | 29/07/13 | 31/07/13 | 100 |
| Final wrap-up and revisions | 11 | 1/08/13 | 15/08/13 | 100 |
| Selenium Tests | 65 | 26/08/13 | 22/11/13 | 100 |
| Management Module | 15 | 26/08/13 | 13/09/13 | 100 |
| Main SAT Control | 5 | 26/08/13 | 30/08/13 | 100 |
| User Management | 5 | 2/09/13 | 6/09/13 | 100 |
| IT Management | 3 | 9/09/13 | 11/09/13 | 100 |
| Auxiliars | 2 | 12/09/13 | 13/09/13 | 100 |
| Login Tests | 1 | 16/09/13 | 16/09/13 | 100 |
| RCPM Module | 4 | 17/09/13 | 20/09/13 | 100 |
| TSIM Module | 10 | 23/09/13 | 4/10/13 | 100 |
| AUDM Module | 5 | 7/10/13 | 11/10/13 | 100 |
| BOPM Module | 5 | 14/10/13 | 18/10/13 | 100 |
| TSM Module | 5 | 21/10/13 | 25/10/13 | 100 |
| Auxiliars | 5 | 28/10/13 | 1/11/13 | 100 |
| Documentation | 10 | 4/11/13 | 15/11/13 | 100 |
| Final wrap-up and revisions | 5 | 18/11/13 | 22/11/13 | 100 |


2013
Nombre
Duración
Fecha de i...
Fecha ...
Progreso
Implementation of an integrated testing system f... 170 1/04/13 22/11/13 100
Getting to know ITSAT 20 1/04/13 26/04/13 100
Installation of ITSAT 9 1/04/13 11/04/13 100
Trying out some preliminary functions 11 12/04/13 26/04/13 100
Preparations for testing 19 12/04/13 8/05/13 100
Research of available tools 3 12/04/13 16/04/13 100
Decision and installation of needed tools 16 17/04/13 8/05/13 100
Testing exercises 18 9/05/13 3/06/13 100
Implementation of test cases 7 9/05/13 17/05/13 100
Implementation of test suites 5 20/05/13 24/05/13 100
Improvements and modifications for tests 6 27/05/13 3/06/13 100
PHPUnit Tests 53 4/06/13 15/08/13 100
Machine Management testing 10 4/06/13 17/06/13 100
User Management testing 10 18/06/13 1/07/13 100
Satremotedo local testing 5 2/07/13 8/07/13 100
Satremotedo remote testing 5 9/07/13 15/07/13 100
Auxiliar testing 4 16/07/13 19/07/13 100
Optimization of tests 5 22/07/13 26/07/13 100
Documentation 3 29/07/13 31/07/13 100
Final wrap-up and revisions 11 1/08/13 15/08/13 100
Selenium Tests 65 26/08/13 22/11/13 100
Management Module 15 26/08/13 13/09/13 100
Main SAT Control 5 26/08/13 30/08/13 100
User Management 5 2/09/13 6/09/13 100
IT Management 3 9/09/13 11/09/13 100
Auxiliars 2 12/09/13 13/09/13 100
Login Tests 1 16/09/13 16/09/13 100
RCPM Module 4 17/09/13 20/09/13 100
TSM Module 10 23/09/13 4/10/13 100
AUDM Module 5 7/10/13 11/10/13 100
BOPM Module 5 14/10/13 18/10/13 100
TBM Module 5 21/10/13 25/10/13 100
Auxiliars 5 28/10/13 1/11/13 100
Documentation 10 4/11/13 15/11/13 100
Final wrap-up and revisions 5 18/11/13 22/11/13 100

# 9. Cost

The calculation of the project costs can be divided into three parts: hardware, software and personnel.

## 9.1 Hardware

The project was developed on one computer but a second one was required to provide the additional disk space and RAM necessary to maintain multiple virtual machines. The external computer is also used for other projects, therefore the cost of the external computer for this project of testing is reduced to 100 euros.

| | **Cost** |
|---|---|
| **Lenovo Thinkpad SL510** | 300€ |
| **External Computer** | 100€ |
| **Total** | **400€** |

## 9.2 Software

PHPUnit and Selenium are open source projects with no associated costs. The only chargeable software used in this project was VMWare Workstation, which emulates virtual machines. Similar to the case of external computer, the cost of its license is reduced to 225 euros due to the distribution with other projects.

| | **Cost** |
|---|---|
| **VMWare Workstation** | 225€ |
| **Total** | **225€** |

## 9.3 Personnel

Using a cost of 45 euros/hour of the Programmer profile, and considering the project being developed at part time (4 hours/day), the overall personnel costs are calculated as follows:

| | **Cost per Hour (€/hr)** | **Hours** | **Cost** |
|---|---|---|---|
| **Programmer** | **45** | **680** | 30,600€ |
| **Total** | | | **30,600€** |

## 9.4 Total

The total cost of the project is **31,225€** (the sum of the hardware, software and personnel costs).

# 10. Conclusion

This project successfully accomplished all predefined objectives and is now integrated into ITSAT to facilitate program management. The tests that have been integrated can detect many problems and have allowed the developers to avoid destabilizing the repository.

The PHPUnit that tests the internal code has on one occasion detected a problem when a developer tried to modify the default path to a directory for Windows. In Windows, the path directory uses "\" from directory to subdirectory but "/" is used in Linux. The abovementioned modification was only suitable for Windows, and the test soon detected that many functionalities for Linux had started to fail. The problem was solved by using conditional clauses. Many issues have also been solved with Selenium, including filters not showing all available possibilities, the revelation of hidden parameters and the hiding of parameters that should remain visible.

Personally, I have learned a lot through the work on this project. Not only I have learned how to use two new technologies, PHPUnit and Selenium, but the project also showed me the importance of testing in a developing project. Although testing might not be as interesting as developing a new module or as challenging as implementing new functions, it plays a highly important role and provides a critical support system. The only way to judge whether a new module is valid or a new function is correct is through testing.

Another conclusion I have drawn from this project is that there is no "perfect" automated testing system. No-one can guarantee that the tests we have implemented in PHPUnit are bug-free because Selenium cannot guarantee that the visual effect on the layout of a website is acceptable. The system of testing is merely a tool which can help developers to detect errors more quickly, but manual testing is also needed at all times for ultimate confirmation.

# 11. Annex A: Development of new tests using PHPUnit

## 11.1 Installation of PHPUnit and PHPUnit_SkeletonGenerator

Note: PHPUnit 3.7 requires PHP 5.3.3 (or later), but PHP 5.5.1 (or later) is highly recommended.

The easiest way to obtain PHPUnit is to download a PHP Archive (PHAR) that has all the necessary (as well as some optional) dependencies of PHPUnit bundled in a single file:

```
wget https://phar.phpunit.de/phpunit.phar
chmod +x phpunit.phar
mv phpunit.phar /usr/local/bin/phpunit
```

The PHAR can be used immediately after download:

```
wget https://phar.phpunit.de/phpunit.phar
php phpunit.phar
```

Once the PHPUnit is installed, it is also useful to have the PHPUnit_SkeletonGenerator, a tool that can generate skeleton test classes form production code classes and vice versa. This package can be installed using the following command:

```
pear install phpunit/PHPUnit_SkeletonGenerator
```

## 11.2 Implementation of the test

A PHPUnit test consists of two files: a file containing the function that will be tested and another which prepares and runs the function in the first file for checking.

For example, the following code can be used to conduct a test for the add function, and it can be saved with the file name *Add.php*.

```
<?php
class Addition
{
        public function add($a, $b)
        {
                Return $a + $b;
        }
}
?>
```

Next, the other file can be created using Skeleton_Generator by running the following command:

```
phpunit-skelgen -test Add
```

Note that this will create a new file named *AddTest.php* and the main skeleton for the test is then provided. Now any test can be developed by adding a function to it as shown below:

```
/**
 * We want to check 0+0 = 0
 */
public function testAdd()
{
        $o = new Addition;
        $this->assertEquals(0, $o->add(0,0));
}

/**
 * We want to check 1+2 = 4
 */
public function testAdd2()
{
        $o = new Addition;
        $this->assertEquals(4, $o->add(1,2));
}
```

## 11.3 Execution of the test

Any PHPUnit test can be initiated by placing the command *$>phpunit* before it, e.g. to run the AddTest implemented as above, the following command is used:

```
phpunit AddTest.php
```

Alternatively, to run all tests saved in a directory, the test file name can be replaced with the directory name. Here are some useful flags that can be used while running a PHPUnit test:

```
--colors:                    Use colors in output.
--debug:                     Display debugging information during test execution.
-c|--configuration <file>    Read configuration from XML file. Often used to fix the
                             order of test execution.

Example:
phpunit --debug --colors -c MyFunctions.xml
```

# 12. Annex B: Development of new tests using Selenium

## 12.1 Installation of Selenium IDE

Selenium IDE is a plug-in for Firefox, so it is only necessary to enable the plugin by adding it to the Firefox browser.

## 12.2 Implementation of the test

Selenium IDE has the ability to register and record all the interactions that have been carried out with the browser (for limitations, see Section 6). If Selenium IDE is open during browser navigation it will start recording automatically.

## 12.3 Execution of the test

### 12.3.1 Execution using Selenium IDE

This type of execution is suited for beginners and for a recently implemented test because it provides live analysis of how a website is changing while running the Selenium test. Open any file saved with Selenium IDE and simply click play to execute the test.

### 12.3.2 Execution using Selenium RC + PHPUnit

This type of execution is better suited for experienced testers. A Selenium RC server can be downloaded from the following URL: docs.seleniumhq.org/download/.

Then it is necessary to open a command and start the selenium RC server as follows:

```
java -jar selenium_rc_server
```

Now that the server is in run mode, another command can be opened and the Selenium test can be conducted with phpunit:

```
phpunit LoginTest.php
```

Note that in order to run the above command, the Selenium test must be exported in PHPUnit format.