Sponsored through Framework Programme Sixth (Call 5) by

Document Information

Version: 1.0
Date : Oct 16, 08
Pages : 14

Owning Partner:
CETIC

Author(s):
Frédéric Fleurial Monfils

Reviewer(s):

Flora Kamseu

To:
European Commission

Purpose of distribution:
Distribution of an intermediate version of the verification and validation of the QualOSS platform deliverable D2.3

The QualOSS Consortium consists of: CETIC (BE), Facultés Notre Dame de la Paix à Namur (BE), Universidad Rey Juan Carlos (ES), Fraunhofer IESE (DE), ZEA Partners (BE), MERIT (NL), AdaCore (FR), PEPITe (BE)

Printed on 10/02/2008

Status:

[ ] Draft
[ ] To be reviewed
[ ] Proposal
[X] Final/Released

Confidentiality:

[X] Public - Intended for public use
[ ] Restricted - Intended for QualOSS consortium only
[ ] Confidential - Intended for individual partner only

Deliverable ID: **D2.3**

Title:

QualOSS - Verification and Validation of the QualOSS Platform

Deliverable: D2.3

Title: QualOSS - Verification and Validation of the QualOSS Platform

Executive Summary:

This document presents the verification and validation that have been conducted on the QualOSS Platform.

As described in the description of work, the QualOSS platform has to be validated in accordance to the requirements elaborated during *Task 2.1*, which includes the specification of acceptance tests (see *Deliverable 2.1 QualOSS Platform Requirements Specifications*). The verification and validation of the QualOSS Platform will be done in 3 phases:

First : set of metrics, tools and user interface
First, the validation focuses on the metrics and their corresponding measuring tools has been made. The validity of the new metrics has not been started yet. Only the implementation and definition of the measures has been started. The verification that a recognized method from measurement theory was well applied when designing these metrics needs to be done based on the *Task 4.2*. Second, we have verified that the measuring tools, that we have selected and or developed, implemented correctly the metrics requirements. Finally, we have verified that the requirements for the user interface matched the users' expectations thanks to the validation of command line options and configuration files. Reviews of the user interface requirements from AdaCore has still to be sought and eventually, from SIG members or other SMEs.

Second : set of metrics, their implementation and integration on the platform
In this second phase, we continued to validate new metrics and their implementation. We tested the first fully integrated version of the QualOSS Platform based on the acceptance test specified during the first iteration of *Task 2.1*.

Third : set of metrics and their implementation
In this third phase, we will validate the last set of the metrics and their implementation. The calibrations of quality models were already partly validated in *Task 1.6* so we will only test the code developed for the QualOSS Platform. This code computes the designed standard QualOSS Quality Model. An example of such a quality model is given in appendix A of the *Deliverable 4.1*. Next, we test the user interface that was adjusted to answer potential users' reviews. Finally, the integrated QualOSS platform goes through the complete acceptance test procedure described during the first iteration of *Task 2.1*.

This version of the *Deliverable 2.3* will evolve iteratively. This first version present earlier Verification and Validation activities. The next version of *Deliverable 2.3, Verification and Validation of the QualOSS platform*. will include the full validation of the new metrics, the validation activities performed to verify the user interface, the outcome of the acceptance tests used to validate the QualOSS platform as well as the test used to validate the measurement tools we developed.

The sections of this document presents the different validations that have been made so far on the current release of the QualOSS Platform according to the planned activities and purposes for this task:

- ***Validate the new metrics***: these metrics should have been designed during WP1 but the list of metrics was too long. It was then decided to focus on the definition of the metrics and their concrete implementation on the platform. This is explained in Section 1;
- ***Validate the tools implementing all metrics***: Some tools were selected to provide the metrics, such as SiSSy or Checkstyle. The integration of these tools on the platform was validated. This is described in Section 2;
- ***Validate requirements related to the user interface***: as decided in WP2.1 there is no user interface on the QualOSS Platform. The platform consists in a set of tools that perform part of the work. The implementation and validation of the QualOSS Platform focused on the correctness of the measures and their collect because as designed, the implemented tools could easily be integrated in a user-

Deliverable: D2.3

friendly user interface. This is described in **Section 3**;

– ***Validate the implementation of the user interface***: this validation consisted in the validation of the options and configuration of the tools available on the platform (because of the choice made and explained in the previous point). This is finally described in **Section 4**.

The QualOSS Platform is part of the Assessment Method of the QualOSS Methodology. The validation of the platform according to the initial version of the *QualOSS Platform Requirements Specifications* (*Deliverable 2.1*) was not possible because of the QualOSS Model was still ongoing.

The implementation of the QualOSS Platform follows the description of the QualOSS Methodology described in *Deliverable 4.1*. This version of the document is also based on this document except for the metrics that are to be detailed in *Deliverable 4.2*.

There are still room for improvement and a second iteration for the Verification and Validation is needed. This document will be updated when the final version of the metrics will be available.

75% of the QualOSS Platform has been validated (except the metrics). The platform showed a good response time, a good accuracy of the computed metrics and a good respect of the declared functionalities. There is however a lack of user-interface that can combine the various tools and ease the job of collecting and interpreting the results.

The next version of this document will take into account the metrics and a textual user-interface.

CHANGE LOG

Ver.	Date	Author	Description
0.1	19/09/2008	Frédéric F. MONFILS	Initial version
0.2	07/10/2008	Frédéric F. MONFILS	Adaptations for the version v0.2 of the QualOSS Platform
0.3	14/10/2008	Flora Kamseu	Review of this deliverable
1.0	15/10/2008	Jean-Christophe Deprez	Sanity Check

GLOSSARY

Term	Definition
Artefact	Element of interest on which measurements are performed
Datasource	Location where artefacts are located
Endeavor	Set of tools, people, rules and workproducts
RDBMS	Relational Database Management System
GQM	Goal-Question-Metric (paradigm for the measurement of data of interest)
SIG	Special Interest Group
SME	Small and Medium Enterprise

TABLE OF CONTENTS

1 Validation of the metrics

The QualOSS Methodology is based on the *Goal-Question-Metric* (GQM) paradigm. The metrics play then a major role. Measurements should be taken and, the QualOSS Platform provides a mean to automate them and store them in a central repository.

The QualOSS Methodology is designed to provide answers related to the *Robustness* and the *Evolvability* of an *FlOSS Endeavor*. A FlOSS Endeavor is composed of:

– People: the set of persons that participate to the endeavor (the community)
– (Work)Products: the set of workproducts that are the result of the activities performed by these people.
– Processes: the processes followed by these people to conduct the activities
– Tools: the tools used by the people to produce the workproducts or to allow people outside the endeavor to access them (these workproducts are placed in *datasources*).

Note: The QualOSS Methodology is described in *Deliverable 4.1*. It defines the terms Robustness, *Evolvability* and *FlOSS Endeavor* in more details.

The metrics that are used to answer the questions from the Goal-Question-Metric are computed by accessing the data present in the *data sources* put in place by the community. Examples of such data sources are a version control repository (*cvs*, *svn*, *bzr*, etc.) or an issue tracking repository (*mantis*, *bugzilla*, etc.).

The *Deliverable 1.3* listed a set of such metrics.

– Some of these metrics were easy to compute, these ones mainly concerned the count of the number of artefacts because they only require the access to these data sources.
– Some others were not easy to compute and they were labelled as 'Advanced' metrics.

The current validation of the metrics was performed on metrics related to the *evolvability* of the workproduct. This work was done by taking a number of metrics related to workproduct (mainly code, as of this writing) as described in Deliverable 1.3 and by validating the interest of each metric for the QualOSS Quality Model. This lead to some renaming in order to be in line with the current QualOSS Model.

In this first version of the Verification & Validation of the QualOSS Platform, we focused on ease to compute metrics and their interaction inside meaningful indicators for the QualOSS Quality Model. The set of validated metrics is detailed below:

– *java_files*: total number of java files
– *java_classes*: total number of classes (also counting anonymous classes, interfaces)
– *java_methods*: total number of methods
– *java_cyclomatic*: sum of the cyclomatic complexity for all methods in java files
– *java_linesofcode*: total number of code lines in java files (counting blank lines and comment lines)
– *java_linesofcomments*: total number of comment lines in java files
– *java_calls*: sum of the distinct calls for all methods defined in the java files

- *java_called_classes*: sum of the fanout (count of distinct called classes) for all methods defined in the java files
- *java_errors*: total number of errors in java files
- *java_committers*: total number of accounts that have committed files
- *java_commits*: total number of commits

Although this first set of validated metrics is small, it is sufficient to use a Quality Model taking into account different aspects of an F/OSS Endeavor. This set includes the workproduct (whith metrics related to the code such as java_files, or java_linesofcode) but also people (with metrics such as java_committers). It also considers 2 datasources, the packaged distribution and the version control system. Finally it uses 3 different tools on the platform: SISSy, Checkstyle and CVSAnalY.

Because the current version of the QualOSS Methodology is only using classical metrics, the validation of those metrics was straightforward. The issue was to find appropriate names so that the name was not misleading. In the next version, if new metrics are introduced, they will be validated via questionnaires and interviews.

2 Validation of the tools implementing the metrics

The metrics listed above are metrics computed directly by the selected QualOSS Tools. The tools that were selected so far are described in detail in Deliverable 1.1 and reproduced here. The description has been updated to reflect the changes occurred in the respective projects since the production of this Deliverable.

From the description of work, QualOSS focuses on Java, C++ and Python FlOSS. The first version of the QualOSS Platform only deals with Java. The validation of the tools implementing the metrics is done for the Java language only.

2.1 SISSy (for Java)

SISSy (for Java)			
General Information			
Version:	0.45 (released: 30/05/2007)	Licenses:	LGPL
Authors:	Adrian Trifu, Mircea Trifu, Olaf Seng, and Peter Sulzman	Maturity:	Stable
URL:	http://sissy.fzi.de/SISSy/CMS/index_html	Dependencies:	ANTLR, JArgs, jTDS, PostgreSQL-jdbc, Recoder (The required libraries are distributed with SISSy)
Description:	(taken from the documentation of the SISSy distribution) The tool for structural investigation of software systems (SISSy) is an open-source platform for the automated detection of structural flaws. It was designed to be integrated in the build process in order to regularly provide reports on the internal quality of the developed system. If in the course of development, problems arise in the structure, they are immediately identified and reported, giving developers the opportunity to fix them before they get unmanageable.		
Constraints:	SISSy works out of the box on MS Windows systems but small changes to the code had to be performed so it ran on Linux. Note: QualOSS contributed to the correction of a few bugs mentioned below.		
Input/Output Information			
Language Analyzed:	C and C++, Java, and Delphi		

Input Types	Source	Input Formats	root directory containing source files
Output Types	Text, Database (MS SQL, PostgreSQL, MySQL). PostgreSQL and MySQL have been tested successfully	Output Formats	database records for miscellaneous source code elements, problematic partner in text files, clone analysis in text files
Other I/O comments	SISSY analyses source code in C++, Java (even Java 1.5) and Delphi. Clone Analysis is run separately from the other analyses. SISSy performs clone analysis as well as a range of pattern analyses looking for bad programming patterns in the code. Bad patterns analysis can be requested via the command line option -queries or the actual SQL queries may also be run on the exported database records.		
Technical Information			
Devel. Lang:	Java	Documentation:	Reference manual
Information Computed:	SISSy performs Clone analysis and Pattern Analysis for 52 poor programming styles. Additional queries to compute traditional metrics could easily be implemented in SQL queries and run against the database of code elements created by SISSy. In the context of QualOSS, the metrics like java_files, java_linesofcode etc have been created based on the provided schema.		
Extensibility:	Additional analysis can easily be implemented by accessing SISSy's result stored in a database. It is also possible to modify SISSy's source code so it performs additional analyses		
Technical Constraints:	In the current version, SISSy has to be run from its installation path because of the hard-coding of resource files.		
Test Performed			
Reliability:	Tests performed on C++ source code: SISSy was tested on four C++ systems. SISSy is based on a CDT parser. Our test shows that it handles C++ for gcc and for MS C++. Tests performed on Java source code: SISSy was tested on 5 Java systems. SISSy for Java is based on ANTLR, a parser generator. Our tests showed that it handles Java 1.4 and also functionalities of Java 1.5. Queries for the java metrics were designed based on the SISSy schema. The Schema allows both PostgreSQL and MySQL databases to be used.		
Performance:	SISSy took from a few seconds to 3 hours to analyze the projects and insert all the data in the database when provided a long list of header files in include.txt. This analysis was performed on a laptop and on the CETIC's cluster. Some of the designed SQL queries for the metrics took a long time on the CETIC's cluster but only a few seconds on the laptop. This issue has been investigated and new versions of Postgresql server and client were installed. This solved the issue.		

As described above, SISSy exports the Java Code Model into a database using JDBC. By default SISSy provides a configuration for PostgreSQL (jdbc.cfg)

```
JDBC_DRIVER="org.postgresql.Driver"
JDBC_URL="jdbc:postgresql://localhost:5432/qualoss"
```

We provided a MySQL configuration

```
JDBC_DRIVER="com.mysql.jdbc.Driver"
JDBC_URL="jdbc:mysql://localhost:3306/qualoss"
```

This creates a connection to the 'qualoss' schema on these two database management systems.

SISSy is responsible for the computation of the following metrics:
– *java_files*
– *java_classes*
– *java_methods*
– *java_cyclomatic*
– *java_linesofcode*
– *java_linesofcomments*
– *java_calls*
– *java_calledclasses*

These metrics were used and must still be validated (by hand) on small projects to check the correctness of the coupling metrics (java_calls and java_calledclasses).

SISSy has been tested on JAVA and C++ projects (no tests have been performed on Delphi as this language is out of scope for QualOSS). For JAVA projects the parameters are:
- the name of the directory containing the sources of the project and
- the name of a configuration file allowing connection to the target DB.

For C++, SISSy expects the same parameters and these two additional parameters:
- the parameter `-cpponly` (meaning that only C and CPP sources are analysed, no header file) and
- a reference to an include file that contains a list of directories used for searching included file.

For C++ projects

We analysed, sources and headers file are mixed in several subdirectories of the source "root". We then had to create the include file using the name of the source directory as single line, all sources and ***referenced*** headers pertaining to the project are auto-magically analysed by SISSy.

Attempts have been made to add more references in the include file in order to find standard header files delivered with the compilers. However, the impact on the resulting metrics is rather low, and as this process is tedious and depends on the production environment, we proposed to use only header files defined in the project sources.

Moreover, tests have been made with the `-def` parameter that contains the macro configuration used in the analysis. Again, defining the used macros cannot be automatic and the impact on the resulting metrics is rather low; so we proposed not use it.

Several features offered by SISSy have not been tested yet (e.g. clone analysis, comment extraction).

During the tests, **several bugs** have been detected by the QualOSS members and they were solved by the members of the SISSy project:
- References to some hard-coded file pathes with Ms Windows syntax
- Problem in the CPP fact extractor due some source code specificities (w.r.t. spaces ans new lines) and the SQL server
- Java StackOverflowError during the analysis of a specific piece of CPP code
- Bug in the ANT buid file w.r.t. dependencies

The following detected issues are still open :
- No correct support of typedef (under investigation by the SISSy project team)
- One erroneous SQL insert statement for a test project (under investigation by the SISSy project team)
- SISSY cannot be launched from "outside" its installation path

The QualOSS Project have then contributed to the correction of bugs present in the SISSy system.

Conclusion: SISSy is a quite stable and mature source code analyzer that parses the source code and exports it in a generic database schema allowing similar checks on the Java, C/C++ and Delphi languages. There are no real metrics provided with this analyzer but the generic schema for representing the source code of a given system allows the writing of measures as SQL queries.

2.2 CHECKSTYLE

Checkstyle			
General Information			
Version:	4.4 (released:)	Licenses:	GNU Library or Lesser General Public License (LGPL)
Authors:	Oliver Burn	Maturity:	Mature
URL:	http://checkstyle.sourceforge.net	Dependencies:	None (beside JVM install)
Description:	Checkstyle is a development tool to help programmers write Java code that adheres to a coding standard. It automates the process of checking Java code to spare humans this boring (but important) task. This makes it ideal for projects that want to enforce a coding standard. Checkstyle is highly configurable and can be made to support almost any coding standard. An example configuration file is supplied supporting the Sun Code Conventions. Other sample configuration files are supplied for other well known conventions.		
Constraints:			
Input/Output Information			
Language Analyzed:	Java (including Java 5)		
Input Types	Source Code	Input Formats	The command line accepts access paths to a file or to a directory (using the -r option)
Output Types	Files	Output Formats	Text and XML
Other I/O comments	Checkstyle can also be invoke from Ant scripts Output files can become quite large, for example, the size of the XML" file generated by Checkstyle when analyzing the source code of Azureus (500 KLOC) is about 100MBytes.		
Technical Information			
Devel. Lang:	Java	Documentation:	User manual available at the URL above
Information Computed:	The checks made by checkstyle are dealing with Javadoc Comments, Naming Conventions, Headers, Imports, Size Violations, Whitespace, Modifiers, Block Checks, Coding, Class Design, Duplicate Code, Metrics, Miscellaneous, J2EE Checks.		
Extensibility:	It is possible to write checks and configuration files. The existing checks can also be modified.		
Technical Constraints:	Checkstyle only performs pattern matching, it does not perform type resolution hence rules cannot check for type information.		

Test Performed	
Reliability:	Currently no bugs are found.
Performance:	For example the Azureus project (about 500 kloc) takes 5 minutes to produce 100MByte size output. The test has been conducted on a Pentium 4 1.5GHz.

As described above, Checkstyle is able to produce its output as XML. The format of this XML document is rather simple: for each file analyzed there is a <file> element, and each error reported for this file is a child element <error>.

Checkstyle is responsible for the computation of the following metric.
– *java_errors*

The checks and validations that have been performed concern Checkstyle version 4.4. This version is the stable one.
Note: A new version of Checkstyle (v5) is in beta state and better designed for Java 1.5 source code.

The installation and execution of Checkstyle was straightforward. The checks performed concern the modification of severity levels for a number of rules. The complete tests of the tool was not possible because of the high number of rules and configurations that is possible. However we used the test cases provided by the Checkstyle distribution (available as ANT tasks “run.gui”, “run.checkstyle”, “compile.tests” and “run.tests”) to check the application.

Conclusion: Checkstyle is easy to use and reports meaningful information about the coding errors found in the Java source code. There is no real issue concerning its use. The reported and opened bugs (as of september 1st, 2008) has been taken into account. The QualOSS project planned to use the new version (version 5) of Checkstyle in the next iteration.

2.3 CVSAnalY

CVSAnalY			
General Information			
Version:	1.0.1	Licenses:	GPL
Authors:	Alvaro Navarro, Gregorio Robles	Maturity:	Stable
URL:	https://forge.morfeo-project.org/projects/libresoft-tools/	Dependencies:	cvs,mysql-server, python, python-mysql, python-mysqldb, python-imaging, gnuplot, ploticus.
Description:	CVSAnalY is a tool that extracts statistical information out of CVS (and recently Subversion) repository logs and transforms it in database SQL formats. It has a web interface - called CVSAnalYweb - where the results can be retrieved and analyzed in an easy way.		
Constraints:	Some features included for CVS are not included yet for SVN.		
Input/Output Information			
Language Analyzed:	Not Applicable		
Input Types	URL (including path on disk)	Input Formats	CVS or Subversion repository
Output Types	Database	Output Formats	SQL Queries
Other I/O comments	CVSAnaly is executed as follows. $ python cvsanaly [-d DBNAME] [CVS-Directory]		

Technical Information			
Devel. Lang:	Python	Documentation:	Reference manual at http://cvsanaly.tigris.org/servlets/ProjectDocumentList and https://forge.morfeo-project.org/projects/libresoft-tools/
Information Computed:	Code repository log data: commits, committers, files, modules.		
Extensibility:	Extension are implemented with plugins		
Technical Constraints:	Current version does not work on Windows platforms because paths to tools does not take windows specificities in the search of commands available in the shell. The version installed with the QualOSS Platform has fixed this issue.		
Test Performed			
Reliability:	Stable		
Performance:	Good, it carries out the analysis quite fast		

The current version of CVSAnalY has been tested by URJC. And it has been tested on Win32 platform by CETIC. As described, CVSAnalY is storing its data in a database and is responsible for the following metrics from the set presented above:

– *java_committers*
– *java_commits*

Conclusion: The checks on CVSAnalY have been light because it was a new version of the tool and development was ongoing. The information provided by the tool was verified on a bunch of projects.

3 Validation of requirements concerning the user-interface

The user interface as described in Deliverable 2.1 is a command line loop that interprets the commands and dispatch them to the appropriate tools.

The validation of the requirements according to end users needs has not yet been performed. The validation will be done as follows: First, the Specification and Requirements for the QualOSS Platform will be validated by AdaCore, responsible for the WP5 Case Study. Then eventually it will be presented to the SMEs forming the Special Interest Group.

4 Validation of the implementation of the user-interface

As explained in section 3, the current implementation of the QualOSS Platform only provided disjoint tools such as Configurator.py, Analyzer.py, Reporter.py and Evaluator.py. The functionalities of these scripts have to be integrated to the Platform.py script that will provide the end-user with a command line loop (like the shell prompt of the MS DOS interpreter, bash or Python interpreter). The validation of the user interface was then not possible.

As of this writing the scripts Configurator.py, Analyzer.py and Reporter.py have been implemented. An attempt to validate the user interface was done by testing the contract using the generated help available with the tools (help is accessible via the -h option at command line).
The logging mechanism used by the QualOSS Platform was also used to check the appropriateness of the reported information on the execution of a given analysis.

There are various roles available for the QualOSS Platform as explained in the Deliverable 2.1, Specifications and Requirements for the QualOSS Platform.

The System Administrator is responsible for the installation of the QualOSS Platform on the machine. Typically, this concerns the installation of the Python environment (at least Python 2.4), the installation of a MySQL RDBMS, eventually the PostGresQL RDBMS, and the Sqlite client. There is so far no user interface for the installation of the platform. This part of the validation has thus not been tested. However a document describing the process that should be followed to ensure the correct installation of the QualOSS Platform is provided as a part of the *Deliverable 2.4 User Guide*.

Verification and Validation performed: The different steps needed for the installation of the QualOSS Platform has been validated. The installation process rely on the EasyInstall facility of Python. EasyInstall, as its names let it suppose, eases the installation of non standard packages on a Python environment. The System Administrator Guide lists the required non standard packages for Python 2.5 (pygresql, mysql-python) and Python 2.4 (lxml, pysqlite, pygresql and mysql-python). The installation of the current version of the QualOSS Platform (v0.2) did not pose any problem.

Tests: Check the reliability of the QualOSS Platform v0.2 by testing the pre-requisites by the QualOSS Platform Administrator:

0. Download the platform (`svn co https://svn.cetic.be/devel/qualoss/tags/v02`): passed
1. Unzip the platform on a directory (referred as `$QUALOSS_HOME`): passed
2. Run `$QUALOSS_HOME/Checker.py`, expect a critical error if Python is not installed: passed
3. Run `$QUALOSS_HOME/Checker.py`, expect a critical error if mysql-python is not installed: passed
4. Run `$QUALOSS_HOME/Checker.py`, expect a warning if pyslite is not installed with Python 2.4: passed
5. Run `$QUALOSS_HOME/Checker.py`, expect a warning if pygresql is not installed: passed
6. Run `$QUALOSS_HOME/Checker.py`, expect a critical error if lxml is not installed: passed
7. Run `$QUALOSS_HOME/Checker.py`, expect a warning if psyco is not installed: passed

As explained in the Deliverable 2.4 'QualOSS User Guide', in the section for the Administration of the platform, for security reason, the System Administrator has to create groups for the different roles. Here are listed the groups:

– Group: QualOSS Administrator
– Group: QualOSS Platform Administrator
– Group: QualOSS Expert User
– Group: QualOSS Advanced User
– Group: QualOSS Basic User

Some of the files, should be available on specific groups only.

The QualOSS Platform Administrator is responsible for the correct configuration of the QualOSS Platform. He is responsible for the configuration of the platform. This configuration is performed by using the Configurator.py script.

As described in the QualOSS User Guide (Deliverable D2.4), the platform is driven by a set of configuration files. The various configuration files needed by the QualOSS Platform Administrator are listed below:

– *Databases.xml*
– *Datasources.xml*
– *Connectors.xml*

As described in the Deliverable 2.4 QualOSS Platform User Guide, the QualOSS Platform Administrator is also responsible for the access rights on various configuration and files. For example, the config/default/databases.xml is not accessible except by accounts in the group *QualOSS Platform Administrator*.

Verification and Validation performed: The verification consisted on providing valid and invalid inputs to the Configurator.py script. Meaningful error messages should be reported. In the next phase, the user interface (the command loop) will be tested for every functionality described in the Specification and Requirements for the QualOSS Platform.

Tests: Check that the script Configurator.py behaves as expected when faced to invalid or valid inputs at command line. All the configuration files are XML documents.

- Run `$QUALOSS_HOME/Configurator.py` without providing a path to a Databases, Datasources or Connectors XML document, expect no error (use of default configuration files respectively:
  - `$QUALOSS_HOME/config/default/databases.xml`,
  - `$QUALOSS_HOME/config/default/datasources.xml`,
  - `$QUALOSS_HOME/config/default/connectors.xml`)
- Run `$QUALOSS_HOME/Configurator.py` with an invalid path to a Databases Datasources or Connectors XML Document, expect an error
- Run `$QUALOSS_HOME/Configurator.py` with a path to a Databases XML Document that corresponds to an existing directory, expect error
- Run `$QUALOSS_HOME/Configurator.py` with a path to non XML documents, or XML document that is not respecting the respective DTD for Databases, Datasources or Connectors, expect error

Note: To ensure that the input is valid for the QualOSS Platform, DTDs have been used (see Deliverable 2.4 QualOSS User Guide)

- *Databases.xml* contains the connection strings to the QualOSS Internal Database. The validity of this connection string is enforced by the `$QUALOSS_HOME/config/default/databases.dtd`.
- *Datasources.xml* describes the contents of datasources and artefacts. The designed DTD (located at `$QUALOSS_HOME/config/default/datasources.dtd`), ensures the validity of the provided XML document.
- *Connectors.xml* contains the needed information to launch and extract data thanks to the selected analyzers and tools. The designed DTD partially ensures the validity of the provided information. Here are the tests that should be performed:
  - Check that the `path` to the tool exists
  - Check that the provided `cmd`; `input` and `output` is sufficient to run the analysis
  - Check that the declared `format` is respected for the output of the tool
  - Check that the given connection string is valid for the connectors whose result is inserted to a database

The QualOSS Expert User is an advanced user (see below) that can additionally (2) create new metrics on the default connectors or (3) implement new connectors and requesting their installation on the platform to the QualOSS Platform Administrator.

Verification and Validation performed: The QualOSS Expert User is often a Python developer because the modification and implementation of both metrics and connectors requires the coding in Python.

Tests: The tests that should be performed on the current available scripts and the current user interface are the same as the ones performed for the QualOSS Advanced User.

As described in the *Deliverable 2.1, Specifications and Requirements for the QualOSS Platform*, the user interface for this user has to make additional checks, such as:

- The interface should allow the listing of the available metrics
- The interface should allow the listing of the installed connectors
- The interface should provide a description for all the installed connectors (name, tool, version, defined metrics, etc.)
- The interface should allow the creation of new simple metrics (if possible)
- The interface should allow the creation of new simple connectors (if possible)

The QualOSS Advanced User can customize the default QualOSS Quality Model by adapting or modifying the quality model used to report the assessment. It has also the permission to run analyses by providing the needed information about a given F/OSS Endeavor.

Verification and Validation performed: The QualOSS Advanced User is using the Reporter.py script that receives the QualityModel file as input. He is also using the Analyzer.py script that requires the Analysis XML document (with the information related to the FlOSS Endeavor to analyse).

Tests: Here are the tests that should be performed for the reporting:
- Run `$QUALOSS_HOME/Reporter.py` without providing a path to a Quality Model, expect no error (use of default configuration: `$QUALOSS_HOME/config/default/qualitymodel.qm`)
- Run `$QUALOSS_HOME/Reporter.py` with a path to a Quality Model that does not follow the correct syntax, expect an error
- Run `$QUALOSS_HOME/Reporter.py` with an invalid path to a Quality Model (i.e., path to a file that does not exists), expect an error
- Run `$QUALOSS_HOME/Reporter.py` with a path to directory instead of a file containing the Quality Model, expect an error

Here are the tests that should be performed for the analysis:
- Run `$QUALOSS_HOME/Analyzer.py` without providing a path to an Analysis XML document, expect an error
- Run `$QUALOSS_HOME/Analyzer.py` with a path to an Analysis XML document that does not follow the correct syntax, expect an error
- Run `$QUALOSS_HOME/Analyzer.py` with an invalid path to an Analysis XML document (i.e., path to a file that does not exists), expect an error
- Run `$QUALOSS_HOME/Analyzer.py` with a path to directory instead of a file containing the Analysis information, expect an error
- Run `$QUALOSS_HOME/Analyzer.py` with a path to an Analysis XML document with unreachable datasource location linked to a metric required by the provided Quality Model, expect an error

There are additional checks that have to be made on the provided user interface for a QualOSS Advanced User. These checks are the same as the checks for a QualOSS Basic User (see below) with additional checks such as:
- The interface allows the user to provide a custom Quality Model
- The interface allows the user to create the configuration for launching a new analysis

The QualOSS Basic User is allowed to access the result of analyses that have been already performed. Its user interface is the QualOSS Platform command loop where he can list the endeavours that have been analysed and that are present on the platform. The basic user can also get the result of applied quality models on a given endeavor. The QualOSS Basic User has also the ability to alter the default QualOSS Quality Model by removing the characteristics that he is not interested in.

Verification and Validation performed: The verification of the user interface will consist in the validation of the functionalities listed above.

Tests: The following tests have to be performed on the user interface designed for the QualOSS Basic User:
- The interface is not accessible without requiring a login/password
- The interface allows the listing of analyses performed
- The interface allows the retrieval of the applied quality model for a given analysis
- The interface allows the retrieval of all the measures computed for a given analysis, as well as all the assessment data (list of analysed datasources, artefacts, scope of the analysis, etc.)