QUALITY ASSURANCE AND TESTING IN LCG

M. Gallas [*], J.T. Moscicki [†], A. Aimar, L. Mancera, M. Lammana
CERN, Geneva, Switzerland

Abstract

Software Quality Assurance is an integral part of the software development process of the LCG project. It includes several activities such as automatic testing, test coverage reports, static software metrics reports, bug tracker, usage statistics and compliance to build, to code and release policies.

Automatic software testing is integrated into the quality assurance activity. The SPI project delivers a general test-framework solution based on open source software together with test document templates and software testing policies. The test-framework solution is built on QMTest, Oval and the X-Unit family (CppUnit, PyUnit, JUnit). The specific languages testing features are covered at the unit-testing level with the X-Unit family, the validation testing activity can be done through Oval. Moreover QMTest offers a way to integrate all the tests and write custom python tests, having a nice web interface for running and browse the test results.

The test coverage reports allow to understand to which extent software products are tested. They are based on the approach used by the Linux Testing Project. The code size and the development effort of the software is estimated using “sloccount utility” based on standard development models. Statistics are automatically extracted from the Savannah bug tracker which enables to analyze the evolution of the quality, amount of feedback from the users, etc. Finally the compliance with the standard LCG policies is verified. It includes the build and CVS repository structure and the standard release procedure.

INTRODUCTION

Quality Assurance (QA) and Software Testing (Sw-Testing) activities in the LCG Applications Area[1] were addressed since the very beginning to support the software development process in projects like POOL, SEAL, PI, and SIMULATION [2]. Both activities are part of the development infrastructure which is provided by the Software Process Infrastructure (SPI) project [1].

The SPI infrastructure promotes consistency and homogeneity of software development in terms of coding styles, build systems, software testing and quality assurance. Additionally SPI provides software project management services such as task management, Web project portals and collaborative tools (Savannah), mailing lists, CVS repository and CVS browsing tools, external software libraries, and software products distribution-kits. The strategy adopted by the SPI project is based first on the breakdown of the whole SPI project in different services that can be operated independently or in conjunction. Second, it tries to use and integrate the available free source code tools. And third, SPI works with the users, starting from the development related work instead from the requirements or design that is leaved to the projects themselves.

In this spirit the QA and Sw-Testing are two, closely related SPI sections that use other SPI services like the SPI External Tools service [2], which offers centralized access to the common external software used by the LCG projects and LHC experiments, or Savannah [3] Project Portal to track the software bugs and analyze the their evolution.

QUALITY ASSURANCE IN LCG APPLICATIONS AREA

Software Development in large scientific environment such as High Energy Physics is different from industrial environment for several reasons:

- there are not easy ways of measuring the compliance of project deliverables with product specification because such specifications *a priori* do not exist;
- management of user requirements is particularly difficult because they are very often discovered and introduced at very late phases of projects;
- software procedures and tools are not standardized and they are defined by agreement of scientific groups rather then by central decision of the management.

Therefore special approach is required to successfully integrate QA into the software development process. It should focus on monitoring and improving information flow between the development teams and users (which are very often other development teams) and enhance the quality of software artifacts such as tests and documentation.

[*] *mgallas@mail.cern.ch*

[†] *jakub.moscicki@cern.ch*

[1] Applications is one of the four activity areas in the LHC Computing GRID Project (LCG) that develops and maintains that part of the physics applications software and infrastructure shared among the LHC experiments (see more in http://lcgapp.cern.ch/project/)

[2] The four development projects within the LCG Applications are:

- POOL: http://lcgapp.cern.ch/project/pool
- PI: http://lcgapp.cern.ch/project/pi
- SEAL: http://lcgapp.cern.ch/project/seal
- SIMULATION: http://lcgapp.cern.ch/project/simu

This would indirectly improve the software quality both from the purely technical perspective as well as the interaction with user community. Organization of QA in LCG provides a concrete example.

Testing and Bug Reporting are central concepts in the LCG QA. *Test Inventory* allows to estimate the amount of support for testing in a project. *Bug Tracker Statistics* gives the defect summary in the function of time. Bug reports (and following bug fixes) may be correlated with corresponding test cases, to make sure that every reported bug is covered in automatic testing and minimize the risk that it will be silently reintroduced in subsequent releases of a project. Bug Tracker Statistics may be generated for arbitrary periods of time including the time covering the release

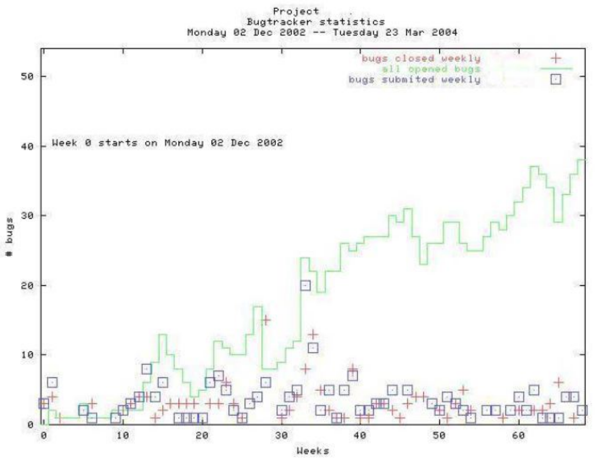

Figure 1: Bugtracker Statistics example

Test Metrics not only give detailed information about the results of the automatic test execution. They also enable to check the test coverage and spot the memory leaks. *Test coverage* is particularly important because it allows to estimate how much of the source code is covered by the tests. It helps to improve the test suite to provide a maximal possible coverage. Current test coverage implementation is very lightweight and uses the built-in features of gcc3.2 compiler and helper tools from Linux Testing Project [11].

Software Complexity Metrics are estimated at the level of packages and subsystems (e.g. [10]). Source code complexity measures as proposed by commercial tools (e.g. [5]) did not gain wide usage and have not been implemented.

Documentation Inventory allows to assess the quality of documentation artifacts. Currently it focuses on automatically generated references (i.e. [4]) and examples provided by the projects.

Source Code Metrics provide a simple estimate of the size of the project measured in SLOC ([7]) units. Coding Conventions are checked automatically ([8]) [3]

Summary of Quality Assurance activities:

- Test Inventory: number of unit, integration and validation tests, number of tests in automatic test system
- Test Metrics: automatic test execution, test coverage, memory leaks.
- Bug Tracker Statistics: number of spotted defects (in time), efficiency of bug fixing (in time), number of users (individuals who spotted defects).
- Software Complexity Metrics: package dependency metrics (e.g. NCCD, see [9]), source code complexity metrics (Logiscope, see [5]).
- Documentation Inventory: number of (meaningful) warnings from automatic refdoc tool per package, number of packages without user documentation, number of examples per package an per project
- Source Code Metrics: SLOC number (number of physical lines of code), Coding Conventions

LCG Applications Area decided to introduce the Software Development Policies to provide a minimal required homogeneity of the project setup, CVS structure, build system, packaging and installation procedures [12]. LCG QA provides tools to ensure that configuration of a build system and CVS directory structure policies are respected.

Every release of one of LCG Applications Area projects is followed by automatically generated QA report. QA reports are published on a public website of SPI project. Additionally any member of the project may generate additional reports with customized options settings. Web interface for the tools is under development to further facilitate the usage of QA tools by all project members. Existing QA infrastructure will be reused in the context of EGEE project [6].

SOFTWARE TESTING IN LCG APPLICATIONS AREA

The main goal of the Sw-Testing SPI component [13] was to offer all the infrastructure (tools, polices, documentation, examples, support) needed to guarantee that within the LCG Applications Area the Sw-Testing can be an integral part of the software development process, all level of software testing can be run as part of an automatic process as many times as needed (by developer, by release managers, at the developing time, at nightly builds or at releases time) and the Sw-Testing infrastructure offers a common entry point for the QA activity in order to inspect the test results and test coverage.

The initial scope within the LCG Applications Area was enlarged by the fact that the acquired experience is now shared among other software projects like the Enabling Grids for E-science in Europe (EGEE) [14] and the offline software for the ATLAS [15] CERN-LHC experiment.

The survey phase of the Sw-Testing SPI component has shown that the software testing activity is not a very common activity in the HEP community and probably this is due to the lack a well defined software infrastructure with the appropriate test frameworks and policies. The intention

[3] at the time of writing, Coding Conventions checks have not been implemented

of the SPI Sw-Testing component was the creation of the needed infrastructure for testing and reinforce in this way the testing culture, working inside the LCG Applications Area projects and learning from their needs.

The three main deliverables from the SPI Sw-Testing component are the test-framework environment, the test policies compilation and the test documentation within the LCG Application Area.

Test-FrameWorks

The test-framework environment adopted is based on open source and existing test-frameworks. Totally independent of the build system it can be run in different platforms (Linux-Unix, Windows and MacOsX platforms), and it covers from unit-testing to validation-testing. It combines QMTest [16], Oval [17], and the X-Unit [18] test-frameworks family (JUnit, CppUnit, PyUnit, etc) in order to fulfill all the LCG Applications Area needs. It offers a structured and coherent automatic software testing approach in which the different selected test-frameworks are totally integrated.

The QMTest tool works as the master piece (see Fig. 2) of the adopted test-framework environment in the sense that it can run all the tests in automatic way offering also a nice web interface (accessible remotely if needed) to setup, to browse and run the tests, easy browse-able log execution file, easy interface with the nightly building system (NICOS [15]) and valuable information for the QA activity. The tests can be organized hierarchically by components or packages, QMTest records the dependences among tests and expected results (pass, fail, untest). It can be operated in batch or through the Web based GUI interface. The tests can be run in parallel, all or one of them at once if needed. QMTest is written in Python and all the configuration files and any relevant information for the reports use the standard Extensible Markup Language (XML), it is modular and new functionality can be added in an easy way as well as other test frameworks.

QMTest was selected among other tested possibilities like the Software Testing Automation Framework (STAF) [19] and DART [20]. QMTest is a light tool that offers a web interface (in comparison with STAF) and it does not depend on any tool to build the test executables (DART depends on CMake).

QMTest as top level tool integrates the other proposed tools as well as the test-scripts or validation examples. The rest of the proposed test-frameworks are adopted due to two main reasons:

1. They offer at the level of sw-testing, tools written for and in the specific programing language (Java, C++, Python...) as it is the case of Junit, CppUnit, PyUnit or any other member of the X-Unit family. This will make a comfortable and more adapted test environment for the developer, in special in case of the unit-testing.

2. To allow two ways of testing: "test inside the testing-code" (X-Unit style) and "check the output of the test code" (Oval style). The later is a quite extended way to work within HEP. The developers normally check the output of their tests and examples on the std-out. The use of Oval represents the minimal effort to convert this habit in something that can be ran automati-

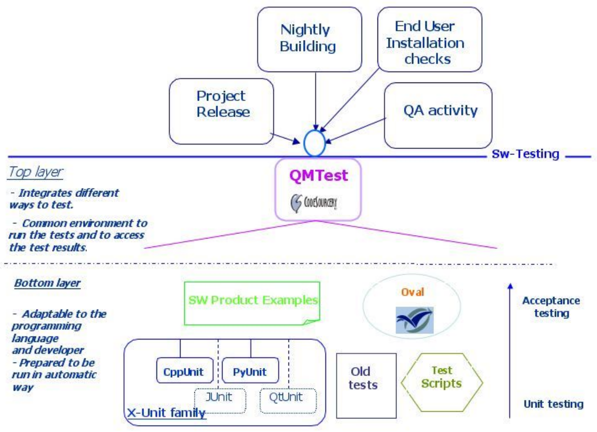
Nightly Building
End User Installation checks
Project Release
QA activity
Sw-Testing
Top layer
- Integrates different ways to test.
- Common environment to run the tests and to access the test results.
QMTest
Bottom layer
- Adaptable to the programming language and developer
- Prepared to be run in automatic way
SW Product Examples
Oval
Acceptance testing
CppUnit
PyUnit
JUnit
QtUnit
X-Unit family
Old tests
Test Scripts
Unit testing

Figure 2: General overview schema of the software testing frameworks used in LCG Application Area.

CppUnit, unit-testing framework for C++, in the philosophy of the extreme programing, is one of the members of the X-Unit family (Junit, PerlUnit, PyUnit...) and it was adopted at the very beginning of the SPI Sw-Testing component as it is out of the box. The work done on it was the creation of a common CppUnit test driver for all the projects to make possible the integration of this tool in the OVAL and QMTest test-frameworks. CppUnit was selected instead other C++ test-unit frameworks (like for example the test library of Boost) for its independence and because it belongs to the X-Unit family (less training and common approach in different programming languages: Java, C++, Python). The scope of CppUnit is the unit-testing in C++ and it provides the name of the test case that fails, the name of the source file that contains the test and the line number where the failure occurred together with the expected value. The use of PyUnit (now part of the standard Python distribution) within LCG Application area was decided base on the same reasons as CppUnit.

OVAL [17] is a testing validation tool developed originally within the CERN-CMS experiment that was adopted as one of the testing frameworks at the very beginning of the SPI Sw-Testing component. The main reason was the easy testing approach in which the tester only needs to validate the output of the test program and create a reference output. Later, the "smart" oval diff command can compare automatically the output of the test program and the reference output. Looking for the specific oval-tags (that must be instrumented in the test code), oval can point the differences between the output of the program at any time and the expected output saved in the reference output. Oval does not provide help to write the test code, it only compares the output of the test code against a reference. Oval has not a browse-able GUI to record the test results and does not offer the extra features that QMTest offers and in LCG Applications Area is always reporting to QMTest tool.

The test-frameworks described above were integrated and made available through the SPI External Library Software service [2] (in different platforms and compilers) with minors modifications in respect to the original sources. The use of open source and not project-built specific tools and solutions allow as to contribute to the general open source projects with a minimal load of work to keep the tools working and updated.

As part of the integration effort and to make easy the software testing to the LCG Application Area projects, specific Python scripts were built to set test-clases examples, test examples and to automatically configure QMTest producing all the needed test suites and test cases. Dependences among tests or test hierarchy must be added by humans using the QMTest GUI interface.

Test Policies

As a part of the needed requisites to have a real software testing activity the elaboration of the Sw-Testing policies was done in agreement with the LCG Applications Area projects. They are explain in detail on the Sw-Testing SPI component [13], and they cover from the design-phase of a software project and the test-plan specification to the code-phase describing how the tests must be done, place in the CVS repository and how they can be run. To be able to run all the tests in automatic way and with a minimal effort in the configuration of the test-framework tools some test-name and directory structure is needed. In this way is also easy to check in a first instance that each software component has the required unit-test (later the test coverage tool will tell more about it) and it is also easy recognize which package or packages are involved in a test that is failing. The bug-tracking activity and the implicit requirement to write a test that can check the detected bug is also a police.

The Sw-Testing polices are somehow summarized in a "Sw-testing QA check list" that it is included in the QA reports.

Test documentation

The software testing activity must be well documented and the SPI Sw-Testing component delivers simple test-plan and test-case templates. Their are simple and short to not overload the developers. Is still needed a tool that can help in the elaboration of this documentation.

SUMMARY

Software Quality Assurance in LCG Applications Area is focused around Software Testing and Bug Tracking. The SPI project provides integrated development services which enable smooth integration of QA procedures into the Software Development Process of various LCG Application Area projects. Rich set of testing tools and extensive documentation of testing procedures provides a substantial level of support for automatic software testing. The methodology of Quality Assurance and Software Testing adopted by SPI is largely based on best practices available in the Open Source Community.

ACKNOWLEDGMENTS

The authors would like to thank the developers within the projects POOL, SEAL and PI for their collaboration and feedback during the initial phase of the SPI project. M. Gallas and J.T. Moscicki are supported by the Particle Astronomy Research council, Swindon, UK.

REFERENCES

[1] The LCG SPI project is described at: `http://spi.cern.ch`

[2] E. Poinsignon et alt, "Managing third-party software for the LCG Application Area project", poster at CHEP conference, September 2004, Interlaken, Switzerland. `http://spi.cern.ch/extsoft`

[3] LCG Savannah: `http://savannah.cern.ch`

[4] `http://www.doxygen.org`

[5] Telelogic TAU/Logiscope, `http://www.telelogic.com/products/tau/logiscope`

[6] `http://www.public.eu-egee.org`

[7] `http://www.dwheeler.com/sloc`

[8] `http://spi.cern.ch/extsoft/rulechecker.html`

[9] J. Lakos, "Large-Scale C++ Software Design", Addison-Wesley Professional; 1st edition (June 1, 1996)

[10] L.Tuura et al., "Ignominy: a tool for software dependency and metric analysis with examples from large HEP packages", (CHEP) conference, 2001, Beijing, China. See also: `http://cern.ch/ignominy/ignominy.html`

[11] `http://ltp.sourceforge.net`

[12] The LCG Applications Area policies are defined at: `http://spi.cern.ch/lcgpolicies`

[13] `http://spi.cern.ch/testing`

[14] L. GUY and alt., "Distributed Testing Infrastructure and Process for the EGEE Grid Middleware", poster at CHEP conference, September 2004, Interlaken, Switzerland. `http://egee-jra1-testing.web.cern.ch`

[15] A. Undrus, "Automated Tests in Nicos Nightly Control System", poster at CHEP conference, September 2004, Interlaken, Switzerland. `www.usatlas.bnl.gov/computing/software/nicos`

[16] `http://www.codesourcery.com/qmtest`

[17] D. Chammont and C. Charlot, "OVAL: the CMS Testing Robot", CHEP, March 2003, la Jolla, California, USA. `http://polyww.in2p3.fr/cms/software/oval`

[18] The X-Unit family is described at:
JUnit: `http://www.junit.org`
CppUnit: `http://sourceforge.net/cppunit`
PyUnit: `http://pyunit.sourceforge.net`

[19] `http://staf.sourceforge.net`

[20] `http://public.kitware.com/DART`