LCG software quality

Massimo Lamanna

CERN LCG/SPI

Quality Assurance

- Improve on the quality of the different components with respect to functionality and performances.
- Help the different software components to be "accepted", i.e. to provide those features (appropriate interface, effective documentation, handy examples, easy installation kits), which make a software tool a success in HEP.
- Building software which can be effectively deployed in a variety of environments (different Regional Centres and desktop installations, different operating system, compilers, etc..) and mantained for a number of years.

QA in LCG

- Phase 1: QA will listen to the developers to ease the development of first packages, gathering valuable input for the SPI project. In parallel, the testing and the documentation infrastructure in SPI will be followed in detail; wherever possible this infrastructure will be used also for QA.

- Phase 2: QA will focus on the validation of the final products (and of the procedure to arrive to it), exploring the tools and metrics best suited to improve the quality standards

- There is no precise plan to improve on some parts of the process (Requirements gathering, formal Use Cases verification, ...)

All inputs are most welcome!

Quality Assurance (cont.d)

- In the first phase, the expected problems will come to the fact that the all infrastructure is immature and too much pressure is expected on the different teams, all focused on delivering the first betas of their packages.
- On a longer term, it is not clear yet the best approach to guarantee high quality standards without too much overhead on the developers. If the procedure is too heavy, it will simply be refused; this is expected in particular due to the features of the HEP enviroment.
- Many of the QA procedures do not translate in a set of predefined procedures and automatic programs; nevertheless we might benefit from some automatic evaluation tools (software metrics). This field is open for investigation.
- There is a general suggestion to use multiple platform and compilers at least to build the code and perform basic testing; to be sensible this require automated procedures.

QA summary

- There is basically only one requirement: the LCG software should be of a quality comparable or superior to the one normally achieved in the experiment software. The users will finally judge... Note that:
- A small fraction of these users are simply using the executables provided; in most of the cases they will modify them.
- Tipical usage patterns are:
  - The user gets a kind of example program, tipically the result of the activity of somebody else. The user modifies it ...
  - The starting point is a piece of C++ using the contributed library. The user takes the code as the only documentation and uses the library in a new class or program.
  - A user should/want to modify a code written by somebody else because it stops the program execution (is suspected to slow down execution). Error or execution dumps are used to look up in the reference documentation to figure out where the problem is.

QA to POOL/(SEAL)/((PI))
but also to SPI

- Liaison for requirements
  - Very good with POOL
  - Very encouraging with SEAL
  - This produces naturally a sort of QA for SPI
- Analyze the usage of the different SPI tools
  - Used?
  - Not used for a good reasons?
- Disclaimer:
  - The fact that practically only POOL is mentioned in the examples (good and bad...) reflects the narrow and constructive relationship with a relative mature project and does not imply any final judgment

Savannah portal

- Many projects use the Savannah portal right from the start and look happy with it
  - Good user feedback
  - Over 40 projects in by now…
- Most of the requirements (fine functionalities) best dealt with SPI
- Problems:
  - Difficult to re-inject patches a Savannah repository ☹
  - Adding too much functionality maintenance headache ☹☹☹

Hosted Projects: 42

- 4 infrastructure
- 7 LCG Application Area
- 7 CMS
- 4 LHCb
- 2 LCG Grid deployment
- 1 HepPackages
- 17 ATLAS

Savannah portal

- Example (Requirement from projects)
  - Missing advanced statistics tools (for example from Bug Tracker)
- Solutions:
  - SPI has an example script (python+MySQLdb) to interrogate the Savannah database back-end to provide statistics
  - More elegant and probably much better for maintenance

Savannah BugTracker usage

- Unfortunately the activity looks low as well as the user base. The statistics is low for all hosted projects (LCG and "LHC experiments specific" projects) and numbers are similar across most of the projects ☹

```
•savannah_bugstat example for project ORCA
•Number of closed bugs: 33 ( over 52  submitted by 6 users )
•Mean time to solve:  265  hours ( +- 449 )
•Max/Min to solve: 2038 / 0  hours
•---
•savannah_bugstat example for project POOL
•Number of closed bugs: 25 ( over 35  submitted by 11 users )
•Mean time to solve:  147  hours ( +- 305 )
•Max/Min to solve: 1404 / 0  hours
```

CVS structure and directory structure

- CVS Defined (months ago)
  - Basically followed in POOL
  - Smoothly evolving without revolutionary changes ☺
- bin/lib structure being fixed (changed)
  - Requirements understood by all parts
  - Positive attitude of the projects involved (SPI, POOL, SEAL)
  - Hope for a fast convergence (meeting on Monday, decision on Wednesday… of the same week ☺)

Testing

- Main tool provided CPPUnit
  - Same family of JUnit, PyUnit, …
- Very good documentation and expertise from SPI
- Still not much used by developers
  - IMHO, the quality of the tool (actually very good) does not play a role ☹
  - The fact that testing legs way behind must change ☹
  - Sensible tests are absolutely needed to attack more OS/compiler/architecture combinations!
  - Testing is not a cosmetic activity: it should go together with the development!!! (common sense, eXtreme Programming,…)

Automatic tests execution and miscellaneous tests

- Tool selected: OVAL
- Main modification:
  - Define responsibilities between SCRAM and OVAL
  - SCRAM builds, OVAL executes ☺
- It can be used to steer existing tests, CPPUnit tests, scripts executing commands, valgrind (memory check)
  - Working on POOL 0_3_2
  - Broken in 0_4_0

```
rc = os.system(cmd)
        if rc != 0:
            raise MyError(cmd)
print '[OVAL] OK'
```

Code reviews

- POOL itself is enforcing code reviews (based on the SPI rules). Presently the scheme is that 2 developers get somebody else code, submit remarks (~10 most outstanding ones) to the code developers which reply (accepting the suggestions or explaining why they are refused).
- Definitely a practice to be encouraged in all projects ☺

Documentation

- POOL Workbook adequate and improving ☺
- Examples needed (POOL has some) ☺
- Doxygen
  - Every day a "snapshot" version is created, put on the web, and checked for packages with too many warnings
  - Mail sent to the developers (with explanations of the origin with the errors)
  - Slowly improving (see next transparencies)
- LXR
  - OK, useful
  - Ideal to build on it (code lines ←→ html links)

Doxygen

- Doxygen is very popular because can create the alibi "but yes, we have documentation!!!"
- Even this requires discipline!

```
POOL_0_3_2
----------
Common                    :     20  warning messages
DataSvc                   :     42  warning messages
FileCatalog               :     32  warning messages
MySQLCatalog              :      5  warning messages
MySQLCollection           :      1  warning messages
POOLCore                  :     10  warning messages
PersistencySvc            :     21  warning messages
Reflection                :     45  warning messages
ReflectionBuilder         :     13  warning messages
RootStorageSvc            :      2  warning messages
StorageSvc                :     62  warning messages
Tests                     :      4  warning messages
XMLCatalog                :      3  warning messages
```

Slowly improving

Developers focused on functionalities…

```
POOL_0_4_0
----------
Common                    :     20  warning messages
DataSvc                   :     42  warning messages
FileCatalog               :     31  warning messages
MySQLCatalog              :      6  warning messages
MySQLCollection           :      1  warning messages
POOLCore                  :     10  warning messages
PersistencySvc            :     20  warning messages
Reflection                :     46  warning messages
ReflectionBuilder         :     13  warning messages
RootStorageSvc            :      2  warning messages
StorageSvc                :     63  warning messages
Tests                     :      4  warning messages
XMLCatalog                :      3  warning messages
```

Note that the doc is usable anyway!!!

User Doc → Use Cases

- Presently (start-up projects), examples are naturally very simple
  - the simple ones are often the most useful
- For all projects it is mandatory to verify use cases interacting directly with experiments code!
- Experiments role essential!
  - Promote the creation of a user basis
  - Bug finding/fixing
  - Impact on the evolution (priority, functionality,...)
- It has to be started ASAP

Summary (1)

- Interesting job!
  - In perspective, interaction with all LCG projects
  - Because it is difficult?
- The SPI QA has to do with
  - The usability of the proposed tools
  - Flexibility to adapt
  - ☺

Summary (2)

- The LCG QA has to do with:
  - Cohesive behavior across projects
  - Be ready to compromise...
    - Homogeneity will be a major added-value of LCG software
  - Implementation of good practices
  - Try to improve on the developer culture
    - Out of the permanent crisis mode
    - Long term perspective
      - Don't be locked to a specific piece of software *forever*
    - Concentrate on the exciting part and do not fight for dull details ☺

Outlook

- Automatisation
  - To be improved!
  - To be built on top of nightly build system
  - Essential for regression and interoperability
  - Also to monitor the "SPI-compliance" of all projects
- Better metrics usage
  - Now just skeleton → complexity = n. of lines ☺
  - Logiscope (v.5.1 installed under pttools on Monday)
  - Absolutely needed to go for "hot spots" if we do not want to prepare zillions of tests
    - In our software!
    - 3rd party or legacy software (e.g. SEAL)