Development and QA Pipeline

Marek Szuba

Karlsruhe Institute of Technology

2011.02.10 / Review of NA61 Software Upgrade Proposal

Outline

Introduction

- "Pipeline" — a set of tools and procedures facilitating development and testing of SHINE software, building of its packages, preparation of documentation and so on
- Mostly absent in present software; the upgrade is a good moment to change this
- In the following slides we shall go over various relevant concepts

Source-code Repository

- **Currently:** CERN Subversion service:
  - collaboration-wide trunk commit rights
  - activity mostly in trunk
- **Proposed:** CERN Subversion service:
  - restricted write access to trunk
  - QA procedures for incorporation
  - extensive use of branches
- Possible use of Git during development

Build System

- **Currently:** a set of hand-made make files:
  - fairly cross-platform
  - complex
  - difficult to configure
- many parameters both build- and run-time
- meant to run in place
- **Proposed:** use CMake:
  - highly cross-platform
  - flexible and configurable
- clear separation of build- and run-time set-up
- implement the install phase

Auto-Building

- **Currently:** none!
- **Proposed:** a *buildbot* set-up
  - build automatically when the source code changes
  - helps maintain repository integrity
  - easier roll-out of new versions
  - can handle multiple platforms
  - can also run tests, check policies *etc.*
- to be run on trunk, selected side branches

Testing

- **Currently:** no defined tests; run by hand
- **Proposed:** a fine-grained test system:
  - unit tests
  - test coverage analysis
  - performance benchmarks
  - ...
  - executed automatically: periodically, at build time

Unit Testing

- Define test procedures for each unit (*e.g.* class) of code
- Facilitates pinpointing problems
- Many choices: *cppunit*, *mockpp*, ...

Coverage Analysis

- Problem: is our code entirely tested?
- Coverage analysis: line-by-line execution statistics
- Tools available: *gcov*, *covtool*, ...

Performance Benchmarks

- Keep track of how changes affect our performance
- Identify bottlenecks
- Can identify certain bugs, *e.g.* memory leaks
- *e.g. gprof*, *Valgrind*

Documentation

- **Currently:** hand-written; scant, largely out of date
- **Proposed:** extensive documentation generation
  - not the perfect solution — but helps
  - encourages commenting the code
  - *Doxygen*, ROOT *THtml*, ...
- Appropriate policies as well

Policies and Procedures

- Complementing automated systems
- Some examples:
  - code: require test routines
  - code: comments!
  - documentation: hand-written where automated insufficient
  - release: new commits must meet all these before entering trunk
- Some manual intervention needed...
- ...but automation helps a lot

Summary

- No development/QA pipeline in current software
  - ...and we've been paying the price
- A good moment for a change
- Much automation possible, using standard tools
- Manual part minimised, worth it in the end

Prior Experience

- Pierre Auger software
  - CMake, buildbot, unit tests, doc. generation
- The *Pygr* project
  - building, unit/performance/coverage tests, doc. generation, policing
- NA61 sub-projects
  - CMake, doc. generation
- Experienced people **available**!

THANK YOU