RTAG on LCG Software Process Management

Fons Rademakers (ALICE/chair)
Marco Cattaneo (LHCb)
Gabriele Cosmo (IT)
Simon George (ATLAS)
Stephan Wynhoff (CMS)

Mandate

- Define a process for managing LCG software. Specific tasks to include
  - Establish a structure for organising software, for managing versions and coherent subsets for distribution
  - Identify external software packages to be supported
  - Identify recommended tools for use within the project – to include configuration and release management
  - Estimate resources (personpower) needed to run an LCG support activity
- Guidance – it is assumed that
  - Procedures and tools will be specified
  - Will be used within the project
  - Can be packaged and supported for general use
  - Will evolve with time

In Addition

- This RTAG does not make any recommendations on how experiment internal software should be developed and managed
- However, if an experiment specific program becomes an LCG product it should adhere to the development practices proposed by this RTAG

Organization

- One representative per experiment, plus IT
  - Marco Cattaneo (LHCb)
  - Gabriele Cosmo (IT/G4)
  - Simon George (ATLAS)
  - Fons Rademakers (ALICE/Chair)
  - Stefan Wynhoff (CMS)
- First meeting 4th February
  - Intensive meetings (> 10 hours) + E-mail discussions
- Timescale
  - Initial report in 1 month
  - Final report within 2 months (LGC workshop)

Consultation of External Experts

- Discussions with Christian Arnault on CMT during productive CMT workshop

- Presentation by H.P. Wellisch on SCRAM

Assumptions (not in mandate)

- All LCG software projects will be components of one overall architecture
- An architect will be appointed to define this architecture. The architect will:
  - Be technically on top of all issues
  - Have a wide experience in the field and a good track record
  - Be able to motivate the people, to explain the issues (like why and why not)
  - Have the final say
- LCG software projects will define a common software development process that all LCG projects will adhere to:
  - Not highly formalised, but assume that it will be based on one or more of the current best practice methodologies, e.g.: XP, RUP, USDP
    - Architecture-centric
    - Iterative and incremental approach to software development ("Release early, release often")
    - Use-case driven ("Let user feedback drive the development").
- We propose tools and procedures to support such a process
  - Which remain valid even if not all assumptions turn out to be true

General Recommendations

- All LCG projects must adopt the same set of tools, standards and procedures
  - Which must be centrally installed, maintained and supported
- Adopt commonly used open-source or commercial software where available
  - In preference to “Do It Yourself”
- All recommendations in this report are unanimously supported by the RTAG members

“Architecture-centric”

- Projects should be decomposed into small teams: 2 to 5 people
- Software should be decomposed into packages
  - A package is an atomic unit
  - With a well defined interface
    - Establish clear interfaces as soon as possible
    - Adopt coherent naming conventions
    - Language bindings: C/C++ mandatory, other languages (e.g. Java, Python, Perl, C#) optional
  - Typically one developer is “owner” of a package with check-in permission
- Supporting tools:
  - Design: no specific tool, but UML notation
  - Code versioning: CVS (including access policy)
    - Checkin only by recognized developers, checkout by anybody
  - Build tool: GNUmake
  - Configuration management: CMT, SCRAM
  - Tools for managing platform-dependencies, packaging, exporting: autoconf, CMT, SCRAM/DAR, rpm, GRID install and export tool

“Release early, release often”

- Major release 2 to 3 times/year
  - Release deliverables:
    - Source and binaries in common place distribution formats, e.g. tar and rpm, GRID package/installation format (for all supported platforms at the same time)
    - Test and validation suites with reference output
    - Up to date documentation: install, user and reference manual, design documents for core components, code examples
    - Hyperized manuals on the web (doxygen, lxr)
    - Release notes - What has changed since last release in each package
- Development release as often as possible (once per 2-3 weeks)
  - As often as needed to stay in synch with rest of LCG project
- Releases identified by number (x.y.z)
  - Use CVS tags to identify releases and package versions
- Automated nightly builds + regression tests and benchmarks
  - Using latest tags made by package developers, on top of most recent development releases
  - Build optimized and debug versions
  - Tests integration
  - Tests portability on supported platforms and compilers

“Let user feedback drive the development”

- Meetings with users
  - Planning meetings to define features for next major release
  - Annual workshop
- Facilitate discussion with users
  - Bug reporting
    - React to reports with priority
  - Animate mailing list/discussion forum (also used to discuss new features)
  - Single point of entry for all projects, with uniform look and feel
- Training
  - Hand on tutorials
- Supporting tools:
  - Considering SourceForge as single point of entry for all projects
    - Automatic creation of archived mailing lists, discussion fora (HyperNews)
    - Browse access to CVS (c.f. CVSWeb)
    - Bug reporting and tracking tool (c.f. Remedy, Bugzilla)
    - Release archive
    - Task management
    - Projects statistics

Testing and Quality Assurance

- Test on different architectures, different compilers, 32/64 bit, little/big endian, Unix/Win32
  - To maintain portability of software to future platforms
  - To increase the chance of finding bugs
  - For example (minimum subset):
    - i386, Sparc/64, IA-64
    - Linux, Solaris, Windows
    - gcc, CC, VC++, icc, ecc
- Adopt coding conventions and rules
  - Do not reinvent the wheel: use existing rules
  - Adopt a rule checking tool and an existing rule set (RuleChecker, CodeWizard)
- QA tests
  - Memory checking (Insure++), unit tests, regression tests, validation tests, performance tests (McCabe)
  - Dependency analysis (Deputy)
  - Create automatic process for development releases

External Software

- Central installation of third party software
  - In one easy to find place
    - Not buried deep inside some release structure
    - Single distribution point with rpm/tar files
  - Multiple versions made available
    - Let users (or integrators) choose the version they want via appropriate configuration tools
    - New versions installed on request
  - No need for additional "private" installations by individual projects
    - Clearly define coherent set of versions used for each release
  - Streamlined installation process
    - Uniform installation procedures for all external packages in same format as LCG software
- Build up local expertise on widely used external packages
  - Provide first level support to users
  - Interact with authors to report bugs etc.
  - In particular case of HEP specific libraries:
    - Local expertise includes active participation in evolution of the software, representing needs of CERN users

Specific External Packages

- Examples given as indication of scope, actual packages depend on recommendations of other RTAGs
- General purpose
  - E.g. Boost, Python, XML Xerces-C
- HEP specific
  - CLHEP
- Frameworks and toolkits
  - ROOT, GEANT4
- Mathematical library

Roles

- Release manager
  - Overall release manager for all LCG core projects
    - Communicates with project managers
  - Is in control and responsible for major releases
    - Defines release schedule
    - Coordinates integration of external and internal packages
    - Ensures completeness and overall consistency of deliverables
  - Quite a lot of work, so post could rotate between LCG project managers
- Librarian
  - Builds and tests on all reference platforms
  - Installs and distributes the releases
  - Keeps external software up to date
  - Configures, installs, maintains the tools supporting the process

Roles

- Tool smith
  - Installs and maintains all products and tools needed to support the development process
- QA support person
  - promotes and facilitates QA process within projects
- Technical writer
  - Keeps manuals, tutorials, etc. complete and up to date

Manpower Estimation

- 1 release manager
  - Rotates between project managers with each major release
- 1-2 librarians, 1 once project stabilizes
  - This is FTE, should always be more than 1 person
- 1-2 tool smiths, 1 once project stabilizes
  - Task could be shared with librarian
- 1 QA support person
- 1-2 technical writers

- Some of these tasks scale with size and scope of the LCG project

Specific Free Tools

- CVS
- automake, autoconf, gmake
- CMT, SCRAM (choose one, both appear to meet requirements)
- cvsweb
- LXR (Linux X-reference)
- Hypernews
- Majordomo
- RPM (RedHat Package Manager)
- Doxygen
- Deputy

Specific Commercial Tools

- SourceForge
  - Collaborative Software Development (CSD) platform from VA software
  - Contains: cvs interface, bug reporting, mailing list, monitoring and reporting, searching, Oracle9 interface, etc.
- IRST rule checker
- CodeWizard
- Insure++
- McCabe