THE DEVELOPMENT AND RELEASE PROCESS FOR THE CMS SOFTWARE PROJECT

S. Argirò, CERN, Geneva, Switzerland and INFN/CNAF, Bologna, Italy *

Abstract

Releasing software for projects with large code bases is a challenging task. When developers are geographically dispersed, often in different time zones, coordination can be difficult. A successful release strategy is therefore paramount and clear guidelines for all the stages of software development are required. The CMS experiment recently started a major refactorization of its simulation, reconstruction and analysis software. At the same time, we revised our software development cycle to improve on release management, build management, distribution management and proper quality assurance via unit, regression and validation tests. In this paper we will report on the lessons learned from our previous experience and on how we are improving in the new project.

INTRODUCTION

The CMS collaboration recently decided to undergo a major revision of its offline and High Level Trigger software. This refactorization moved along two main streams:

- Adoption of a completely new Framework [1] for event processing
- Merge of the separate projects for framework, simulation, reconstruction and trigger into a single project with a common source tree.

To complete this process in the shortest possible time, but at the same time guaranteeing consistence and quality of the product, special requirements were imposed on the development and release cycle. In particular, the cycle had to be able to cope with the following features:

- large quantities of code could be ported in a few days
- very frequent releases are required, the time to deliver has to be as short as possible

RELEASE AND DEVELOPMENT CYCLE

Given the premises we outlined in the above, we set to deploy release and development procedures to deliver in the shortest possible time. In particular, we abandoned the scheme previously adopted in the CMS experiment, in which each project was released in several steps. In the past scheme, the project was divided in Subsystems, each with a clear dependency scheme on each other. The release was built by releasing the Subsystems in order of dependency.

* for the CMS Collaboration

In the new scheme, developers submit code using the *Cms Tag Collector* described below. They can all submit at any time, irrespective of the position of their code in the dependency graph. When a change in a package breaks another package, the nightly build system catches the problem and emails the package administrator.
Every week, the Release Manager integrates a "prerelease", using the last nightly build as a starting point. Every month, a "release" is issued.

Project Structure

The CMSSW project includes all the functionalities related to offline data processing. In particular it includes:

- Geant 4 Simulation
- Fast Simulation
- Reconstruction
- High Level Trigger
- Analysis
- Visualization

All the code is kept in a single CVS repository and is part of the same source tree. Also, it is released as a single monolithic unit.
The source tree is organized in a hierarchy of two levels: the first level is called *Subsystem* and the second level is called *Package*. Every package is maintained by *developers*, who have the rights to commit code, and *administrators*, who have the right to commit, tag and submit tags for integration. The control of these right is done in CVS via the `cvspm` add on.

Build environment: Scram

CMSSW uses SCRAM [2] as a build and configuration management system. SCRAM influences the way development is carried out, in that it provides the developer an easy and quick way to set up his environment. SCRAM holds a database of available projects and versions, typically releases, pre-releases and nightly builds. With two simple commands, the developer creates an area for the project and sets its runtime environment. He can then recompile only the parts of the code of interest, while using other libraries and include files from the the central project installation.

Nightly Builds

Nightly builds are an important tool to provide developers a way to check consistency of their code. In CMSSW,

nightly builds are installed as SCRAM projects, providing the following advantages:

- Developers do not have to spend time in compilation, they only have to compile the development version of their packages.
- Developers always have the possibility to stay up with the latest developments.

CMSSW uses a modified version of the NICOS [3] nightly build system. Every night the system tries to compile all packages that were submitted for nightly build by package administrators using the *Cms Tag Collector* . The system parses the output of the compiler, and identifies problems such as compilation errors and link errors. Administrators and developers of packages are informed of such failures automatically via email. The final report is displayed on a web page, which shows the packages with problems in red at the top of the list. From this page, the full logs of the build is accessible as well as the partial log of a particular package.
We plan to set up a test structure to run tests of each package nightly and use the same error reporting mechanism.

Tag Collection

When the package administrator thinks the package is ready to be submitted to the integration system, he applies a CVS tag to that package. Then uses the `CmsTC` to publish the tag. The `CmsTC` (CMS Tag Collector) consists of a Web interface to release administration, supported by a MySQL database back-end. The release manager can open a release based on an existing set of tags (usually the nightly or an existing release). The package administrator publishes a tag, and selects the release for which the tag needs to be integrated. The collection of tags can be open or closed: in the case of a closed state, tags are put in a queue to await the approval of the release manager. In "open" case, all submitted tags are taken for integration.
Publication of a new tag is notified to developers via email. The Tag Collector has a number of features that help in making the release, such as tag queries, differences in tags between releases, etc. The `CmsTC` is also used to request new packages, that have to be approved by the release manager.

Release procedure

Releases can be of two different types:

1. pre-releases
2. releases

Typically, a development cycle consists of 4 pre-releases before the final release. Pre-releases are intended as checkpoints of the development line, and are made mostly for the developer's convenience. CMSSW releases are typically made every month. Monthly planning is done in order to understand the needs and set the goals for that release. Typically, a deadline for submission of tags is decided. After the deadline, the Release Manager puts the release in a "closed" state and tries to build the whole system. Simple integration tests are also ran by the Release Manager, to make sure the product will not be "dead on arrival". A more complete *Validation* suite is ran after the release.
If compilation or runtime errors are found, the release manager solicits submission of new tags, that he can then approve using the Tag Collector.

DEVELOPMENT TOOLS AND DOCUMENTATION

A set of tools is provided in the developer's runtime environment:

1. `rulechecker`, to test if the code is compliant with CMS coding conventions
2. `includechecker`, to verify if unnecessary headers are included
3. `dependencychecker`, to generate package dependency graphs and detect circular dependencies
4. `valgrind`, to check memory usage

This tools are collected in the *Ignominy* [8] toolkit. Documentation is a very important issue in a large project. We provide documentation in two forms. A *Reference Manual* is generated using the `doxygen` [5] documentation generation system. We use `doxygen` markup for both documenting the code itself and for providing a general description of each package. We provide templates for package description pages and automatic substitution of certain keywords, like author, project version, date. The other form of documentation is a *Workbook* or *User Guide*, which is written using the `Wiki` documentation system.

TESTING INFRASTRUCTURE

Each package can register a set of tests that are ran as a build target. Tests are of two types:

1. unit tests, for which the `CppUnit` [6] testing framework is used
2. integration and validation tests, for which the `Oval` [7] system is used.

In addition, a set of application integration tests is ran manually by the release manager to provide a quick validation of the release.

DIFFERENCES WITH PREVIOUS CMS EXPERIENCE

The main differences with the previous development and release cycle are the following:

- The CMS offline was split in several projects: ORCA for reconstruction, COBRA for framework, OSCAR for simulation, FAMOS for fast simulation. The new project collects everything in the same code base, with the advantage of an easier configuration management: there is no need to keep track of which version of the reconstruction goes with which version of the framework, for example. The disadvantage is that it is hard to distribute the software other than in a monolithic piece.
- Each CMS project was released in a staged manner. The project had a series of incremental prereleases, where, in order of dependency, code would appear. The disadvantage of this approach is that development lines tend to diverge, as developers typically do not wait for their turn in the release sequence, but continue their work. The changes in the packages they depend upon are detected late. In the new scheme, developers work constantly on the same code base, and the nightly build system notifies immediately of possible problems with dependencies.
- Tag collection was done via email. In the new scheme, a database is used for tag collection, with the advantage that, in any moment, developers and release managers can check the status of a particular release or ongoing tag collection.

PLANNED IMPROVEMENTS AND CONCLUSIONS

The new release procedure adopted for CMSSW has succeeded in delivering releases very rapidly, as many as three different releases in a week. This was the most important goal, given the special nature of the project, that involves primarily porting and refactoring of existing code.
We do see several ways to improve the robustness of the development and release cycle:

- Reduce compilation time by parallelizing the build
- Modularize the distribution kit
- Optimize the testing infrastructure

In the coming months, we will be addressing these issues to be in the best possible conditions at the start of data taking.

REFERENCES

[1] C. Jones et al, The New CMS Event Data Model and Framework, this Conference

[2] http://cmsdoc.cern.ch/Releases/SCRAM/doc/scramhomepage.html
http://cms-nightly.web.cern.ch/cms-nightly/

[3] http://www.usatlas.bnl.gov/computing/software/nicos/
http://cms-nightly.web.cern.ch/cms-nightly/

[4] https://cmsdoc.cern.ch/swdev/CmsTC/

[5] http://www.doxygen.org

[6] http://cppunit.sourceforge.net

[7] http://oval.in2p3.fr/

[8] http://ignominy.web.cern.ch/ignominy/