Release Strategies: The CMS approach for Development and Quality Assurance

Elizabeth Sexton-Kennedy on behalf of the CMS Collaboration
Fermilab, P.O.Box 500, Batavia, IL 60510-5011, USA
E-mail: sexton@fnal.gov

Abstract. Now that CMS has started data taking there is a balance to be struck between software release stability for operations and the need to improve the physics and technical performance of the code. In addition new code may need to be developed to correct for unforeseen data taking conditions, and has to be integrated into the mainstream releases with a minimum risk. To keep the process under control, CMS uses regular (twice a day) Integration Builds. A complex set of validation steps is used to verify the software at various stages, from the regular Integration Builds to running a full software and physics validation suite on the grid for major releases. CMS has adopted a development model that tries to strike the correct balance between the needs of stability and a constant improvement; this paper will describe our experience with this model, and tell the story of how the commissioning of the CMS offline has proceeded through the perspective of the past year's releases.

1. Introduction
In order to effectively organize the making of releases of CMS software, CMSSW, it is important to understand the roles involved. Therefore the who of releases is discussed in the next section. The third section presents what we are releasing with some notes on the software itself. Then finally there is a discussion of how we put releases together as illustrated by our development model and testing procedures.

2. The Who of CMSSW Releases
Releases of CMSSW software are put together by a release manager, which is a rotating duty among the people who are trained to do it. The release manager is responsible for cross project integration and the overall QA of the release. If a problem shows up in the nightly monitoring, it is their responsibility to make sure the appropriate people are notified to fix it. Helping the release manager, are the sub-project coordinators, for example the reconstruction or simulation coordinators. They are responsible for integration within their project as well as the changes induced in other sub-projects by their change. When a new development is proposed by one of them, they will list the CVS tags of all of the necessary package changes and coordinate the tagging of all dependent software. The third group of people, are the administrators and developers of the different software packages. These work on three different categories of software: CMSSW event processing, user analysis, and data and workflow management software.

The developers of CMSSW are mostly part-time physicists. These people split their time between developing code, doing analysis, detector operations, data/release validation, and meetings. It is therefore sometimes difficult to get their full attention or participation. There are however a core of professional software developers, which in the beginning of the project created the core software, that now work to preserve the quality of the entire project in addition to maintaining the core. Because of this, the project is a cooperative one, in which the physicist contribute their domain knowledge, and the engineers make sure that it is implemented in a reasonably performant and maintainable way.
A second category of CMS software is the individual user analysis software used by a small group for a specific analysis. There are very little requirements placed on this software, it is only required that it be publicly reviewable by the Collaboration. Users are encouraged to use common tools as much as possible[1], however many use their own private code.

A third category of CMS software is the data and workload management, DMWM, software. This software is independent of CMSSW releases. There is very little overlap in people between this category and the above two. The Venn diagram in the insert of ‘figure 1’ below gives an idea of the overlap integrated over the lifetime of the project. There have been 184 contributors to the DMWM repository, but only 5 of those have contributed in all 3 areas.

Notice that there is a much larger overlap among the people who have contributed to CMSSW and private analysis code. The coding skills learned in the more disciplined area of CMSSW development does carry over to higher quality software in the user repository for those people that work in both. The plot itself is of the number of unique committers to the CVS repository each month, as a function of the month. Note that the manpower supply has seasonal fluctuations. The green shading highlights vacation times. The red region represents the time of preparation for CMS’ first conference presenting results from the initial 3pb$^{-1}$ of LHC data taking. The drop in CMSSW participation, and the increase in user analysis activity over this time reflects the changing focus of the physicist involved in the project.

![Figure 1](image)

**Figure 1.** Number of unique committers to CVS each month as a function of month in the three different categories of CMS software.

### 3. The What of CMSSW Releases

The rest of this paper will concentrate on the release event processing software which covers the applications of simulation, reconstruction, triggering, validation, monitoring, event display and analysis tools. For being such a diverse set of applications there is one thing they all have in common. There is no way to measure the physics performance of the software, until it can be exercised with, and for the simulation, compared to real LHC data. The exponential turn on of integrated luminosity in 2010 has made release integration particularly challenging as the collaboration developers struggled to incorporate detector understanding into the software. In ‘figure 2’ the total integrated luminosity is plotted as a function of time starting in March of 2010. Event reconstruction becomes more
challenging as the instantaneous luminosity increases, so those values are also marked on the plot. Note that instantaneous luminosity increased 5 orders of magnitude between March and November. Over this same period of time the number of lines of code changed by a small fraction of the total.

The start of the ‘figure 2’ corresponds to the last eighth of ‘figure 3’. ‘figure 3’ plots the number of source lines of code as a function of month and release number, for the three languages used in CMSSW.
source lines of code integrated into a release as a function of time. Note the long period of time in 2009, marked in red, in which new code was being developed but not integrated into a release. The problem was that the 3_1_0 release was marked as the release to be used for first data taking in the fall of 2009. This forced a long period of scope creep for this release which made it nearly impossible to converge, and required heroic efforts on the part of the release manager. To address the problem CMS adopted a new model of development, called the train model.

4. The How of CMSSW Releases

4. 1. The Train Model
With the advent of data taking there is a balance to be struck between software release stability for operations and the need to improve the physics and technical performance of the code. To address this, CMS moved to a system of regularly scheduled releases, similar to an extreme programming approach [2]. As a first step, the feature set for a new development release must be declared within the first week or two of the beginning of a cycle. The subproject coordinators vet the request for inclusion, to make sure that it has a reasonable chance of making the schedule for the release and that required updates to dependent software have a reasonable chance of adjusting to the changes. This helps to avoid mission creep, and manage expectations of the release. The consideration of the dependencies implies that interface changes to the core software must happen very early in a cycle, changes to the simulation shortly after that, local reconstruction, followed by high level reconstruction after that.

The pre-release schedule is set in advance and is usually 6 weeks long; however some accommodation is made for major CMS milestones and the seasonal variation of manpower. New developments are continuously integrated 2 times a day in the integration build, IB. Frozen pre-releases are tested by our data operations team once a week. Developers are asked to check the results and the subprojects are asked to update the validation tables. The cycle ends with a high statistics data processing which is signed off by the Physics Validation Team, or PVT. The PVT takes a more global view of the physics performance of the release and evaluates it’s suitability for all of the intended applications of the releases, for example a reprocessing of all of the data taken in 2010 with updated alignment and calibration constants. Once the evaluation is in, the collaboration makes a decision about the timing of deployment of the release. This will depend on events like accelerator technical stops, and physics conference deadlines.

As soon as the PVT starts its work, the next release cycle is opened for the developers. By continuously integrating new developments we avoid the problems of the big bang and reassure our developers that their work is valued and will eventually be put into production, depending on its urgency.

If a particular feature is not ready on time, the developer must wait for the next scheduled release, or the next train to leave the station. Once a release has gone into production, its feature set is gelled. Bug fixes and a limited number of mission critical features (for example something required to take data) are allowed to be added to a bug fix release. As the commissioning period ends there are fewer of these. In addition the proposed change must have made it through the vetting process of the development release. Only then can it be back-ported to a production bug fix release.

CMS has many train cycles per year but only 2 or 3 of them will be deployed for data taking or reprocessing.

4. 2. Quality Assurance
As mentioned above CMS builds and runs all of the CMSSW software that has been accepted for release by the subproject coordinators twice a day. A web page summarizes the results for both the production and development releases. An example snapshot of this web page is in ‘figure 4’. It is the responsibility of the release manager to check the status of the builds and report problems to the
developers or subproject leaders which caused them since the release manager knows best what has changed from build to build. The purpose of doing this twice daily is so that the increment of change is as small as possible. The words in blue are links to details in each category. Figures 5-6 are snap shots of these pages.

![CMSSW integration builds](image)

Figure 4. Snap shot of the CMS integration build summary page.
Summary for CMSSW_3_10_X_2010-10-12-1300 IB on platform slc5_i686_gcc434 -- Back to IB portal

| Error Type       | # of Packages/total # of errors |
|------------------|---------------------------------|
| diskError        | 0/0                             |
| compileError     | 0/0                             |
| linkError        | 0/0                             |
| pythonError      | 0/0                             |
| compWarning      | 0/4                             |
| duneError        | 0/0                             |
| mineError        | 0/0                             |
| dependency violations | 25/unknown            |
| scram err        | 0/unknown                       |
| scram err        | 0/unknown                       |
| libchecker       | 339/unknown                     |

Color in this table is a key for the status field in the following table, green means successful

Rule based dependency checking

Does it link to a library it doesn’t need

Headers are zero suppressed

For the new libchecker errors and the SCRAM errors and warnings please click on the linked number to see the details.

| Status | Subsystem/package | compWarning | Dependency violations | UnitTest logfile | libCheck |
|--------|-------------------|-------------|-----------------------|------------------|----------|
| 0      | GeneratorInterfaceGenExtensions V01-03-04 | 33          | -                     | -                | -        |
| 1      | GeneratorInterface/PatternRecoV01-01-00 | 2           | 3                     | -                | -        |
| 2      | GeneratorInterface/PythonInterface V01-04-42 | 11          | -                     | -                | -        |
| 3      | L1Trigger/UTrace V02-00-04 | -           | -                     | -                | -        |
| 4      | CalibCalolometry/EventCovarianceNoiseAnalysisAlgos V01-01-00 | -           | -                     | -                | -        |
| 5      | CalibCalolometry/EventPolarizationOffsets V02-00-24 | -           | -                     | -                | -        |
| 6      | CalibTracker/StripDCS V02-01-00 | -           | -                     | -                | -        |

Figure 5. Snap shot of the CMS integration build details page.

In the above page, you can see that there were no Errors in this particular build, 4 packages had warnings, 339 linked to libraries they didn’t need, and 25 packages depended on packages that they should not depend on; for example a reconstruction package that depends on a simulation library. Only columns with non-zero entries are listed in the table on the same page. Each package_cvs-tag is a link to the log file that gives details of the problem listed in the column on the right.

In ‘figure 6’ below the monitoring of quality and performance metrics is illustrated. There is a combination of static and dynamic checks that are done for each build. The green bar next to the valgrind link means that this build found no memory errors for the 4 standard applications (the 2 steps of simulation, HLT reconstruction-filtering, and full reconstruction) for 2 different types of simulated data and one detector dataset. Ignominy is a tool that creates dependency graphs and can detect circular dependencies. Code rule violations includes the results of scripts that look for known constructs that reduce the maintainability of code, for example using a “using namespace” declaration in a header file. The plots are history plots covering a period of the last two weeks of builds. The performance suit link includes the results of the igprof and perfmon [3] run on the same applications as the valgrind memcheck test listed above. The source code level details of the most expensive functions, via the igprof-navigator, are linked from this page.
5. Summary
As long as CMS is still in a commissioning mode, the intense release and accompanying validation effort has to continue. However we are physicist manpower limited, so it should be recognized that all needs cannot be simultaneously met. The balance between stability and latest functionality must be kept. Continuous quality testing of the code contributed daily, ensures that the new features required for physics will be ready for deployment with minimum risk.

References
[1] A Tour of the CMS Physics Analysis Model (PS08-5-166), Benedkt Hegner
[2] http://en.wikipedia.org/wiki/Extreme_Programming
[3] The Evolution of CMS Software Performance Studies (PS24-1-182), Matti Kortelainen