Activities associated with a software build, continuous integration, and release program

Jim Kowalkowski

1 Introduction

The purpose of this document is to describe the software deployment and quality assurance tasks that are carried out by SCD staff for SCD developed products or on behalf of the user community. This includes software products that are used by our customers, used within our products, and produced, supported or maintained by SCD affiliate organizations. Each activity will contain its description and a summary of how it performed now. This document will communicate the current, ongoing, and future needs for these tasks and provide a common understanding of what the tasks are. The intended use for this document is to help determine what tasks, if any, should be centralized and managed across projects to form a coherent, on-boarded program. It will also be used to aid in assigning priority to each of the items.

All of the tasks described in this document have been loosely called development services and support systems. Many of these activities seem to best fit into the area of *Software Deployment*, although some could be described as assist services and tools within the *Software Development Life Cycle*. The *Deployment Activities*[i] that will be explored here include release, install and activate, update, and version tracking. The roles that are frequently associated with these activities include build-release engineers, release managers, and deployment coordinators. The *Software Development Life Cycle*[ii] phases that will be explored here include Integration and Testing, and Acceptance, Installation, Deployment.

This document is organized into four major sections. The Overview section is a high-level description and view of the major functions that ought to be performed by a new program broken down into distinct areas. The new program will be referred to throughout this document as the *Build, Continuous Integration and Release* program, or BCIR. The Current situation section describes how and where many of the major functions are carried out now. The Proposed program section identifies objectives for the new BCIR program and highlights some of the needs and changes to the set of customers for this program. Finally, the Operational scenarios section begins to describe in more detail the major functions performed in each of program areas, with the focus being development of use cases that show the relationship and use within each area. There is a large amount of information that could not be incorporated into this document. The End notes section has pointers to relevant documents and web pages that describe the current situation. The endnote document xiii has a fairly complete terminology section that is relevant here and is not repeated.

2 Overview

While reviewing the integration, build, and release functions that SCD is currently performing within various projects and experiments, an obvious breakdown into areas was apparent. An area represents a distinct line of work with specific goals and objectives. Each area is further broken down into the aspects of facilities, services, and projects. Below is a description and summary of each of the identified areas that can form a new BCIR program.

2.1 Areas

Release management: This area is primarily concerned with providing the release management and product packaging and building services and tools for groups developing and using software infrastructure. The services include packaging for external products.

Continuous integration: This area focuses on access and use to the Jenkins build services and associated database, testing interfaces, and web summary facilities that exist. This includes assistance with integrating software product builds and unit / integration test suites with Jenkins, and supporting the facilities and software infrastructure to permit the running of a product build within this context. This includes building development and integration candidate releases for products on-demand, triggered, or periodically. It is recognized that this system relies on the build system that is part of the software package.

Product distribution: This area focuses on product distribution through central web and file system services. It organized and allows users to access versions of full binary and source distributions for experiments across platforms and build configurations.

Validation control (future): This area focuses on testing and quality assurance. Of particular interest is the control and coordination of performance and validation runs. This is a newly proposed area. Performance evaluation for both checking accuracy and speed both fit into the testing area and can be triggered by the release building (including development and integration builds). Testing of LArSoft would likely be a first candidate here. Future candidates for this area are the Geant4 performance runs and the GENIE validation runs. Both of these systems have high demands that will almost certainly exceed the capacity and scope of the current tools and resources, requires interactions with GRID or other batch-oriented computational resources.

2.2 Aspects

Almost all the areas have some aspect of facilities, services, and projects. The facilities are the resources (computation and software packages) necessary to make an area work. The services configure the facilities to make them useful for the customers, and in this case, actually utilize the resources according to user needs. The services also aid in the support of the tools that are used within each of the areas. The projects contribute to the development for the software tools owned and maintained within BCIR. The projects come and go as features and software releases are needed.

2.3 Future considerations

It is reason to consider adding documentation generation services to the areas in the future. The purpose of this is to add, just like testing stages in a build, automatic documentation generation for software products that are using the continuous integration system. Two examples of products that could be included here for support and configuration are doxygen and LXR covering documentation embedded in source files and source code navigation.

2.4 Management

A key objective of this program is to consolidate resources for activities that are now spread across many projects within and outside of SCD. Organizing the aspects into the areas defined above allows for a clean and clear separation of responsibilities. There is also a natural flow from left to right through the areas as a product becomes ready for use by a customer. The coordinator assigned to run this program will act as a central point for gathering and meeting requirements, measuring and reporting progress towards goals, and strive for commonality in these areas amongst all customers.

2.5 Summary

The aspects and areas are summarized in a Table 1.

Release Management (Common build, packaging & release management)	**Continuous Integration** (Common facilities for build and testing)	**Product Distribution** (Common distribution site)	**Validation Control** (Common tools for launching and summarizing validation runs)	
	• Workers • Servers • Jenkins • Web tools • Database	• CVMFS • Web Site • Storage	• Future servers • Storage • Workflow tools	**Facilities** (computation resources and software packages)
• Experiment production release control • SCD production release control • External product packaging • Product dependency management • Tools configuration	• Experiment Integration • Tools configuration • Interface support	• Filesystem maintenance • Tools support • Web index management • Distribution manifests	• Future monitoring and configuration • Workflow and GRID integrate	**Services** (configure and utilize facilities, support software)
• cmake external • MRB • UPS packaging • Production build tools • Distribution build tools	• CI test interface • Database • web summary	• Production distribution tools • Web management tools	• Future interfaces • Coupling tools	**Projects** (development of the software tools owned and maintained within BCIR)

Table 1

The table shows a proposed breakdown of four identified major areas under a new BCIR program (first row), and the three major aspects of facilities, services, and projects (the last column). The cells of the table describe common example activities that are performed and the systems and software that are affected for a given area and aspect. There exists a natural relationship between each of the area. Many of the use-cases flow from left to right, with release management driving continuous integration, causing distributions to be posted, triggering validation.

3 Current situation

SCD is currently providing support for one or more of the areas defined above to many projects and experiments. There are little to no formal agrees on support levels for any of these areas. Some of the work is done as “best effort”. Much of work in the tools and release areas is done without much oversight and organized management. Services in these areas are provided at levels to keep customers happy. There is not much organization to promote common solutions for each of the areas. Individual projects develop minimal infrastructure to meet there needs a given moment.

3.1 IF release and packaging services

The SSI team currently maintains releases of several SCD-developed products. Notable ones are the art suite, the LArSoft suite of packages, artdaq_core (a subset of artdaq), and nutools. In addition, experiment software releases for mu2e and g-2 are maintained. The maintenance includes all the external packages that are used by these software systems. Each release of these products is qualified[iii], meaning that there is more than one supported build configuration depending on the experiment, the platform, and environment under which it will be used. The qualifiers define the build configuration and affect the externals that will be used to compile and link the applications and libraries. The art suite build includes several lower-level SCD-developed products: message facility, fhiclcpp, cetlib, and cpp0x.

Useful information about some of the key SCD product builds can be obtained with the UPS depend commands on supported systems. Running these commands shows how a release is constructed and viewed: it is a collection of versioned products tagged, built, and tested together to form a consistent whole. Lower-level dependent packages are built and installed first, and later packages use the installed versions of these packages. Each package knows the dependent package versions that are compatible with it.

```
-bash-4.1$ ups depend nutools v1_07_00 e6:prof
-bash-4.1$ ups depend nutools v1_07_00 debug:e6
-bash-4.1$ ups depend larsoft v03_04_03 -q e6:prof
-bash-4.1$ ups depend larsoft v03_04_03 -q debug:e6
-bash-4.1$ ups depend art v1_12_04 -q e6:nu:prof
-bash-4.1$ ups depend art v1_12_04 -q e6:prof
-bash-4.1$ ups depend art v1_12_04 -q debug:e6:nu
-bash-4.1$ ups depend art v1_12_04 -q debug:e6
-bash-4.1$ ups depend artdaq_core v1_04_06 -q e6:prof:s5
-bash-4.1$ ups depend artdaq_core v1_04_06 -q debug:e6:s5
```

Each of these is used as an external within an experiment. A complete build of these products requires consistency across all dependence packages, including the externals. A key feature is that underlying packages only need to be rebuilt if they are changed (new version is available), otherwise the existing binary distributions are reused. The cet-is wiki contains a dependency diagram snapshot[iv] of most of the products that are currently packages and build by SCD.

The build system used for LArSoft, art (along with underlying packages), and artdaq_core is cmake coupled with cetbuildtools and cetpkgsupport. Mu2e and g-2 share a binary distribution (the mu2e one). Mu2e uses scons as its build system.

Relocatable UPS[v] is used to manage built software products and their relationships. This is a simplified use of the full UPS/UPD that permits binary distributions of tarballs to be downloaded to destination computers and installed without root access. SSI occasionally submits patches and bug reports to the UPS support team to improve this operating mode.

MRB is a tool built alongside LArSoft to simplify the building of multiple products pulled from separate repositories. The original design and requirements came from discussion with g-2 and the art team. G-2 has since adapted MRB for their use. The requirements are covered in the SSI-produced requirements document (xiii).

Maintenance and ownership of packaged externals is handled in an ad hoc way. All externals are wrapped for distribution in a UPS package. The product development teams will typically introduce a new external to be added to the dependency chain of their component. The build personal will take over packaging and versioning responsibilities of the new product. The external wrapping process requires tools that aid in UPS packaging of an external. A new version of an external can be introduced by the build support personal because it contains useful fixes and updates or because it uses newer versions of lower-level products that the a distribution needs (for example a gcc is needed by mu2e, and therefore it is useful to upgrade boost). A new external can be introduced by the build personnel if it deemed good to have as a common tools and requested by several groups (Qt is a recent example of this kind of external). Platform additions and upgrades usually require several external upgrades. Developers or user that want to make use of the new platforms typically donates these.

3.2 Build services and the Continuous Integration System

The build resources used are the art development cluster head node cluck.fnal.gov (SLF6) for most products and the Jenkins continuous integration system. The packages are built and distributed for SLF5, SLF6 and OSX. They are known to work on similar Linux flavors (CentOS and Ubuntu). The complete list of experiments and projects that utilize the CI system can be seen on their main web site[vi].

LArSoft is heavily using the Continuous Integration facilities. Results and status of build stages can be seen on the main web site. Two important projects are addressing missing pieces within the system, mainly long-term results storage using a database, and build/test test summary results displays through a web site. The early development for these features being carried out in the context of LArSoft, and they have collected some requirements[vii]. The prototype for the results display can be seen on the web[viii]. LArSoft performs nightly and continuous development builds. It is not clear how nightly builds are used by experiments. Changes in LArSoft heavily affect both uBooNE and LBNE because of their dependence and reliance on it. Currently there is one main branch that is used for releases and it uses a tagged release of art.

Mu2e is finalizing plans for integrating their software build into the Jenkins system. They will likely have nightly builds and will want to run the their test suite. They will continue using the scons build system internally.

LBNE is utilizing the CI system. They provide a nightly build, but it is not widely used. They have their own release manager. An alternative build infrastructure is being finalized and is ready for demonstration.

Minerva is using BuildBot to orchestrate the building of their Gaudi based software package. They can move to the central Jenkins system and want their validation steps run automatically.

GENIE provides a nightly build using Jenkins. The artdaq team is building most of packages they own. The Darkside-50 team is also building releases.

BNL has been offering to help in the area of build coordination tools, including product packaging and distribution tools. The help might extend into cmake as a standard build system. The demonstration work they have completed will need to be reviewed and service level agreements will need to be discussed and negotiated before deciding how to move down this path.

3.3 Distribution services

Each built product is installed. The standard installation area is a relocatable UPS product tree. Each installed product is formed into a distribution tarball[ix] and placed onto the distribution server scisoft.fnal.gov so that it is available for download. Complete distributions such as mu2e, LArSoft, artdaq_core, and art development contain rules for downloading a complete set of package tarballs. A tarball manifest is put together for each major distribution. The manifest lists all the packages that are needed to form a complete distribution. A “pull products” script is also prepared. Its purpose is to take a manifest and download the missing

packages for a distribution into a user's installation area and unwind them. Each major distribution is available within a directory on the Scisoft server and acts as an index into the complete set of available packages. The current distribution bundles are available at bundles[x]. See the manifest files for each of the bundles to the list of package versions that it contains. The full list of available packages[xi] is available at the scisoft site. Some of the requirements and organization of the scisoft site are available on the redmine wiki[xii].

Build and release requests are done on-demand. Releases are triggered by completion of a feature set or by an upgrade to one or more dependent products.

3.4 Tools maintenance and support

The software products that are used for all aspects of release management and distribution are very important for the projects within SCD and the experiments that use them. As mentioned above, the ownership of these products is spread across several groups and projects. Requirements for many of the tools in this area that are currently in use have been accumulated into a document titled *Requirements for Software Product Building and Management*.[xiii] This document has a fairly comprehensive terminology section that also defines many of the terms used here. The document concentrates on the requirements for tools that are used for build, release, packaging, and distribution of software products.

The main SCD-developed products that satisfy many of these requirement include: cetbuildtools, ssibuildshims, pullproducts, cetpkgsupport, and MRB. Under the covers these packages utilize git, cmake, make, and UPS.

There is another set of open source and licensed products that aid in development, testing, and problem analysis. All of these packages require packaging and upgrade maintenance. They are used in both the production and development environments. The main tools are: totalview, allinea (profiler and debugger), valgrind, and OpenSpeedshop. The main distribution mechanism for these products is UPS.

3.5 Validation

Geant4: performance runs are carried out regularly to measure changes in execution speed from point release to point release. The results are posted on an SCD-hosted web site[xiv]. They are collecting and storing system and application performance measurements for representative Geant4 configurations. Fairly high statistics are collected increase accuracy since sampling is used to gather the performance data. There is a web interface for inspecting the collected data and comparing it with previous runs. Changes in performance can be inspected down to the function level of applications. Dedicated resources are needed for this operation to minimize unwanted fluctuations in the performance measurements. They are currently using the SSI development cluster for these runs and this has been increasingly problematic because it is not a production resource. A request is in the system for moving the running of the performance jobs to the Wilson cluster. The Geant4 group has written requirements for the data collection and processing.

GENIE validation and testing stages are being designed and analyzed now. The validation stages are going to need resources that are not provided through the CI project. Large samples will be generated and analyzed by the validation steps.

4 Justification for changes

Good support for the BCIR areas covered in this document is essential for making software products developed by SCD and affiliated experiments usable by our community. This is especially true where produced software is shared amongst experiments within and outside of Fermilab across more than one platform and computing site.

Tools to support BCIR functions are developed, prototyped, or adopted as low-level support infrastructure under a project. The CI database and web site is an example under the LArSoft project and the cmake-based cetbuildtools is an example under the art project. Support and maintenance is therefore mixed in with development of the project itself. Requirements, operations, support, and maintenance are not formally in project scope. Operations (using the tools) is carried out by members of the same development team. Unfortunately this situation makes it difficult to prioritize and manage development of the mainstream product and internal infrastructure required by stakeholders. It also puts as addition burden on the project team, which now has multiple levels of support and development to address and juggle. A second disadvantage of this organization is that many of the infrastructure tools are useful outside of the project that is generating them, and visibility to those needs is limited. The result is that reduced sharing of the tools.

As sharing of facilities, tools, and services for the areas defined here become more widespread, nesting the maintenance, development, and support within specific projects becomes unmanageable. Individual projects start diverging from the shared products as soon as problems or missing functionality are found. When underlying support infrastructure sharing is involved, the line between in-scope and out-of-scope work is fuzzy, making prioritization between well-defined project work and the extra tools work difficult.

A problem occurred recently between MRB and G-2 that illustrates some of the problems outlined here. MRB behavior was changed. The change caused failures in the G-2 build that took a few days to clear up. The build system tools set policies for code structure that must be maintained. Changes to these policies can cause many issues in the customer code, and therefore need closer management and control over changes and releases.

Distributed development, especially outside Fermilab, has increased dramatically over the past year. The number of platform is going up and we are moving well beyond SLF 5/6. In addition, we are anticipating that the diversity of architectures and system configurations will be increasing as many-core and multi-core energy-saving cores become mainstream. As a result, many things will need more attention, such as excellent test facilities and services, automation of builds across platforms and architectures, and collecting of performance monitoring data.

5 Proposed program

5.1 Objectives

The goal is to establish a program that takes ownership of the areas described in the overview section of this document. The program will be the home of expertise in build system infrastructure and release management and under it will be maintain recommended products and procedures. Initially the first three areas of release management, continuous integration, and product distribution will need to be moved into this program. The fourth area of validation control can be added as additional requirements are gathers and a project plan is created. The four areas, as shown in the table of the overview section, can be viewed as forming a connection from left to right. Release management typically triggered builds and testing on the continuous integration system, which in turn causes release candidates to be packaged and posted on the distribution server. The posted release candidates can then be used to trigger extensive validation suites.

Requests should be handled through the services aspect using any of the standard issue tracking systems. Support staff for the provided services across all areas will be responsible for interacting with facilities. They will also control the project aspect to initiate enhancements to features, provide for larger sets of changes to tools, and manage tool releases. A stakeholder community will help to prioritize changes to tools, policies, and supported packages and platforms.

It is desirable to farm out some of the tools support and development, with a primary goal being to offload work to collaborators outside FNAL. Procedures need to be included for contributing to the BCIR products and services.

It is also highly desirable to move the development and support of CI-related functions for tracking status through databases and web site development for viewing build and test results into this new program. This movement will help maximize the sharing of the developed tools amongst all interested projects and experiments. The CI test running interface should also be included here because its utility extends beyond liquid argon-based experiments.

5.2 Notes about identified customers

Artdaq: The project team would like to be able to schedule releases as needed. They are very interested in automating the running of an integration test that operates using more than one node, verifying that the distributed processing is working as expected. Part of the artdaq toolkit (artdaq-core) is already built and releases within SSI. This is the part that is shared with offline code. There will be a large number of qualified builds for a given release due the variety of supported networking hardware and support packages.

Artdaq-demo: This is a package built on top of artdaq and supported by that project team. They want to be able to schedule a release and distribution when needed. This might be the environment under which artdaq itself is tested. It is desirable to have the testing include both an Ethernet network (simpler) and Infiniband network (more specialized).

Art: The project team wants to be able to trigger a tagged release and unit / integration tests run using the CI system (across all valid qualifier combinations). They also want to build and test for other Linux systems, including CentOS, Ubuntu, and Debian. The SSI department is already helping with support of art releases and distributions.

Synergia: The project team is currently using a project-built packaging system called Contractor. They would like to move to a common supported toolkit for releases and external product management. They currently to not have a production release. They have an extensive test suite that would fit nicely into the CI system. The package dependencies for Synergia are shown in the web page referenced in the endnotes. The difficult part of support this project is the diverge platform support, which includes: blue/Q, SLF5, SLF6, Fedora, Ubuntu, New Cray at NERSC called Cori, Summit at Oak Ridge (Nvidia / IBM). frequency - no binary or standard builds. Currently developers and users do independent builds and the MPI libraries need special care because they are platform specific. On Mac OS X they use homebrew for almost all of the external products. On SLF many things come from standard system RPMs. For NERSC systems, they desire to utilize the standard modules product for binary product packaging, which will let users set up a version and use it on a system. A consistent whole is needed for all dependent products and this can be a difficult task that will need to be worked out. Ideally they want RPMs to install Synergia on Linux systems that support it. They would also like a binary install of releases on their primary FNAL cluster tev. They want nightly builds in Jenkins, and would be willing to utilize the scisoft distribution sight. There will be a stable and development release supported, and will want to declare that a tag and release is necessary which will form the development release.

CosmoSIS: The project is included in the dependency diagrams available in the endnotes. This project has many of the requirements and restrictions of Synergia regarding the platform support (no Bluegene support) including the need for MPI. The two projects also share many external packages. Currently product management is handled through relocatable UPS. The project team wants to release management support through this program. They also want test suite integration that runs on the CI system, along with specific test sites, such as Midway at University of Chicago. They want to use the continuous build features of Jenkins, triggered by commits to the central repository.

LSST DESC: Currently the needs for this project are mainly following CosmoSIS. They will diverge when specific packages need to be turned into managed products to be used on FNAL resources, such as the LSST Data Management Software Stack.

LArSoft: This project is already a large user of the areas supported by this program and this will continue.

Mu2e and G-2: Both of these experiments rely on current SSI services for releases and build tools support and this will continue. They will be moving further

toward complete CI integration, which includes running their integration tests, and hopefully parts of validation into the future.

GENIE: This project is already using the CI system for builds. Discussions have started about how the CI system might be used to aid in the running of GENIE validation. The resource load for this operation is large and will need to extend out to the GRID. It may be possible to utilize parts of the CI system database-tracking infrastructure (currently being developed in the context of LArSoft) to record progress and state of GENIE validations.

Geant4: Within FNAL, this project will look similar to GENIE. The FNAL Geant4 team already had performance and validation systems and tasks defined. It may be possible to reuse parts of the Geant4 performance tracking infrastructure within the CI system.

Main experiments not yet included in this section: DarkSide, uBooNE, NOvA, and DES.

6 Operational scenarios

Table 2 lists several of the functions that can be performed under the services and projects aspects of the four defined areas.

Release Management	Continuous Integration	Product Distribution	Validation Control	
• Tag release • Configure ACLs • Initiate build • Update Jenkins configuration • Complete release • Initiate packaging • Package external • Accept external • Configure build	• Integrate test & build summary • Integrate build • Initiate build • Check state • Check results • Check test summary • Adjust authority	• Configure ACLs • Retrieve bundle • Post packages • Update index • Build manifest • Define bundle	• Initiate validation • Check status • Configure resources • Adjust authentication • View summary • Transfer summaries	**Services**
• Maintain development make system • Maintain make conductor system • Main build coordination system	• Maintain CI test interactions • Maintain configuration of Jenkins • Maintain DB • Main web summary	• Maintain product pull • Maintain build coordinator • Maintain documentation	• Maintain interfaces	**Projects**

Table 2

Full use cases that touch functions from each of the service areas will be available on the public redmine wiki[xv].

7 Not covered in this document

Contained here is a list of topics that need to be discussed, but are not covered in this document:

- Packaging and distribution of server products, such as the postgres server
- Management of Open Source licensing, outside repository management, and redmine accessibility issues

8 End notes

---

[i] http://en.wikipedia.org/wiki/Software_deployment
[ii] http://en.wikipedia.org/wiki/Systems_development_life_cycle
[iii] https://cdcvs.fnal.gov/redmine/projects/cet-is-public/wiki/AboutQualifiers
[iv] Snapshot of most package dependencies, including Synergia: https://cdcvs.fnal.gov/redmine/projects/cet-is/wiki/DependencyViewSnapshot17122014
[v] https://cdcvs.fnal.gov/redmine/projects/cet-is-public/wiki/AboutUPS
[vi] https://buildmaster.fnal.gov/
[vii] https://cdcvs.fnal.gov/redmine/projects/lar-ci/documents
[viii] http://dbweb4.fnal.gov:8080/LarCI/app/view_builds/index
[ix] https://cdcvs.fnal.gov/redmine/projects/cet-is-public/wiki/Get_binary_distributions
[x] http://scisoft.fnal.gov/scisoft/bundles/
[xi] http://scisoft.fnal.gov/scisoft/packages/
[xii] https://cdcvs.fnal.gov/redmine/projects/cet-is-private/wiki/SciSoftRequirements
[xiii] https://cd-docdb.fnal.gov:440/cgi-bin/ShowDocument?docid=5380
[xiv] https://oink.fnal.gov/perfanalysis/g4p/
[xv] https://cdcvs.fnal.gov/redmine/projects/cet-is/wiki/UseCasesBcir27122014