|  |
| --- |
| NCI CBIIT |
| Knowledge Center Code Management Process |
|  |
|  |
| **Anand Basu** |
| **7/31/2008** |

|  |
| --- |
|  |

## https://wiki.nci.nih.gov/download/attachments/9175500/edu_languages.pngIntroduction

The purpose of this document is to provide a starting point for a process for managing feature and bug fix contributions from the caBIG Knowledge Centers (KC) within the context of an actively developed product.

For any process to be successful there must be a clear understanding of the roles and responsibilities of everyone involved and the clear boundaries within which they must operate. A successful process must be repeatable, transparent and flexible to evolve to meet the needs of caBIG, CBIIT and NCI.

This document is divided into two major parts – the first part deals with the overall process of delivering changes from the Knowledge Centers from a Product Management perspective and the second part delves into the details of managing the development process using various SCM tools and technologies from a software development perspective.

Figure 1 below summarizes the overall flow of information toward building a unified KC process view.

Figure caBIG KC Components

# Terms and Definitions

The following terms are used throughout this document. The readers should familiarize themselves with these terms to extract maximum benefit from this document

**SCM: Software Configuration Management** – management of change in code repositories.

**SVN: Subversion – A version control system** – used to maintain current and historical versions of files.

**CVS: Concurrent Versioning System** – Another version control system

# PART I – Product Management Perspective

## KC Stakeholders and their Responsibilities

**User Community:** The User Community (Cancer Centers, caBIG Workspace, Product Users) provides the key input to the KC regarding new feature requests, urgent bug fixes, etc.

**Knowledge Center Management Team:** This team provides the steering and prioritization function to the KC development teams. This team consists of the KC Program Manager and the Product Managers for each of products under the respective KCs. Interrelated product managers (e.g. LSD, caIntegrator, etc.) may also be included in this team.

**KC Development Team:** The KC development team works in close coordination with the KC Management team to schedule releases, provide support, test features. The lead for each KC communicates project progress to KC management on a regular basis and provides updates on schedules and deliverables.

## The Product Management Process

For the purpose of this document, the following key roles and responsibilities are defined as:

**KC Program Manager:** The KC Program Manager is responsible for the overall management of the Knowledge Centers. They may assign designees to perform day to day management and scheduling functions.

**Product Manager:** Each product (e.g. caArray, NCIA, caTissue, etc.) under the KCs must have an NCI CBIIT assigned product manager responsible for oversight of KC source contribution alignment with the active product development. The product manager works closely with the KC Program Manager to prioritize, schedule and deliver/accept changes from the KCs into the mainstream product development cycle.

The following are the key steps involved in the product management process within each KC:

1. A Change request(s) or bug fix(es) is/are identified for a certain product
2. KC Program Manager (KC PM) reviews change request and accepts it for submission to KCs
3. KC PM assigns change request to NCICBIIT Product Manager for review.
4. CBIIT Product Mgr reviews change request and approves.
5. CBIIT PM assigns Release Manager (see Part II below for details on Release Management Process)
6. Product teams Release Manager creates Branch in SVN repository (To Do: Branch Naming Convention)
7. KC Works on Branch to deliver changes to support a.
8. KC PM lets Release Mgr know when work has been completed.
9. Release Manager inspects changes and accepts (merges) to trunk.
10. Product Manager informs KC PM of accepted/rejected status of changes.

Figure 2 below summarizes this process.

Figure Knowledge Center Product Management Process

# PART II – The Software Development Perspective

## Quick Info

* [CBIIT Enterprise Ivy repository](https://gforge.nci.nih.gov/svnroot/commonlibrary/trunk/ivy-repo/)
* [GForge](https://gforge.nci.nih.gov/)
* [Requesting a CI VM](https://wiki.nci.nih.gov/x/VAGU)
* [Build Promotion Checklist](https://wiki.nci.nih.gov/x/yqB8)

## Goals

* Consistent, easy-to-use process for making changes in parallel
* Single SVN project repository for existing projects hosted at CBIIT
* Merge back to trunk often
* Discover any merge conflicts at least once a day
* Run Continuous Integration on trunk and branches
* Use automation to make tagging, branching and entire process less error-prone and more efficient

## Development Process Perspective

There are three flavors of development covered in this process: Mainline, Branches and New Products/Utilities. Mainline (trunk) development is typically used by development teams for Major or Minor releases of the software product. Every release from such development generates an SVN **tag**.

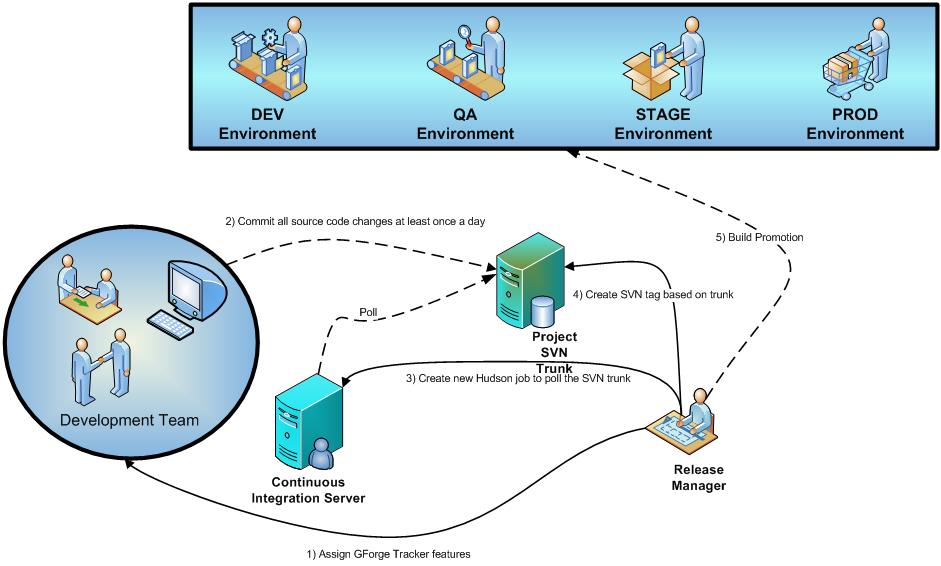
Branch development is always based upon an SVN **tag**. Often, branches are created when making a modification to code and the developers do not wish to disrupt mainline (trunk) development. These are usually bug fixes and may sometimes include emergency patches and minor feature enhancements. Knowledge Centers will always do branch development.

New Product development is when new functionality is provided for a specific need. A typical scenario for new development may be when Knowledge Centers create solutions for specific cancer centers that are not a part of the overall software product. In this case, developers may interface with the product's API (i.e. client JAR file) only. Other examples may include support for new data types - e.g. support for a new imaging modality in NCIA or support of a new microarray platform in caArray, etc.

**Mainline (Trunk)**

**Assumption**: Teams that have scheduled major/minor releases will apply changes to the mainline (SVN trunk). Knowledge Centers will not perform Mainline (Trunk) development is another designated team is also working on the product on behalf of CBIIT. If a product is not under active development, KC’s may perform development on the trunk.

1. The project's **Release Manager** determines the features that will be a part of the next major/minor release and specifies these features in [GForge](https://gforge.nci.nih.gov/)/Bugzilla .
2. All developers commit **all** code to the mainline (SVN trunk) *at least* once a day
3. The **Release Manager** configures a Hudson job to poll SVN trunk and run an integration build
4. When the development team is ready to release, the **Release Manager** creates an SVN **tag** based on the **trunk**
5. When the development team is ready to release (e.g. after several weeks), the **Release Manager** follows the [Build Promotion Checklist](https://wiki.nci.nih.gov/x/yqB8)  to get the software into production



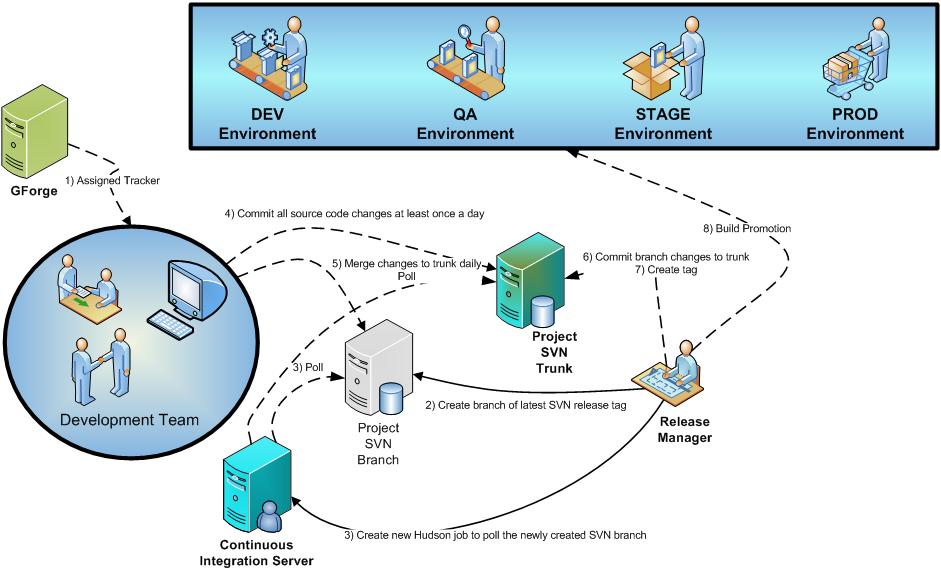
**Branches**

A new branch will be created for every patch or set of features/bugs that a KC might have been assigned to. Here's a scenario of applying a change (such as fixing some defects). The release manager will often be creating branches based on the latest release tag designated for a specific KC.

***Example****: A KC development team spends five days fixing defects for a set of items assigned to them.*

1. A developer is assigned trackers in [GForge](https://gforge.nci.nih.gov/)/Bugzilla
2. The CBIIT development team's **Release Manager** creates a **branch** based on the latest production release SVN **tag**
3. The CBIIT development team's **Release Manager** configures a new Hudson (CI server) job to poll the newly created SVN **branch**
4. The developer(s) commit **all** source code changes *at least* once a day
5. Developer(s) merge changes to the trunk (**without** committing) *at least* once a day (or configures a CI job to execute against the product’s release **tag**)
6. When the Knowledge Center development team is ready to release, the **Release Manager** **merges** and **commits** changes from the branch to the trunk.
7. When the Knowledge Center development team is ready to release, the **Release Manager** creates an SVN **tag** based on the **branch**
8. When the development team has completed changes (e.g. after the five days), the **Release Manager** follows the [Build Promotion Checklist](https://wiki.nci.nih.gov/x/yqB8)  to get the software into production
9. Once the release is in production, the CBIIT development team's **Release Manager** removes the **branch**

**Branches should always be transient - they should be removed eventually. Branches should not be a permanent fixture in an SVN repository. For instance, creating branches shouldn't be used as a technique for managing product variations (e.g. cancer-center specific code). All code should have a path back to the product's mainline**



**Tools and Commands**

**Note: This section is under construction. More information will follow.**

* How and when to create an SVN branch
* How and when to create an SVN tag
* How and when to merge a branch back to the SVN trunk
* TODO: Cover the version naming convention
* Communication between Knowledge Centers and Development teams
* CBIIT will provide tools for reverting back to a branch revision (in case code that was branched breaks the code in the trunk)

**Policies**

* All code that is checked into Subversion by a KC must have corresponding **automated unit/component tests** using JUnit, for instance. If the code coverage for new code is not at least **80%**, it will be *rejected*.
* All new features should have corresponding **automated functional tests** that are run as part of the automated build. For example, using tools like Selenium for web-based applications (http://selenium.openqa.org/) and White (<http://www.codeplex.com/white>) for rich-client applications
* KC must run a successful build (with successful automated tests) in the **Continuous Integration environment on the branch** prior to committing code back to the trunk.

**https://wiki.nci.nih.gov/download/attachments/9175500/flashkard.pngRecommendations**

|  |  |
| --- | --- |
| Recommendation | Description |
| Assign Release Manager role | Create a new **Release Manager** role per project - caArray, caIntegrator, NCIA, etc. |
| CI VM per project | Each project team (e.g. caArray, caIntegrator, etc.) should have a Continuous Integration Virtual Machine (CI VM) provisioned that runs an integration build with every change applied to trunk and branches. One Hudson CI server per project running multiple jobs. A [CI VM can be requested https://wiki.nci.nih.gov/x/VAGU] from the CBIIT Systems team. A Hudson job should be configured for:   * SVN trunk for project * Each SVN branch per project |
| Branches per patch | An SVN branch should be created anytime a team is working on a patch |
| Branches should always use an SVN tag | Branches should never be based off the mainline (trunk) – only a release tag |
| Commit ALL source files to trunk/branch every day | All developers should be committing ALL source files to trunk/branch *at least* once a day |
| Configure CI job to merge branch against trunk daily | One of the CI jobs should merge a particular branch with the trunk once a day, without committing code against the trunk (to resolve merging issues early in the process) |
| Centralized SVN repository | All developers should be working from the same CBIIT Subversion repository for a particular development project |
| SVN tag per release | At a minimum, create a new SVN tag for every new release (major, minor or patch) |