The CASA Release Process

Nick Elias
V0.0 - 2010 May 6

0. Introduction

The two releases I have overseen since joining the CASA project have been "eventful" to say the least. Here is a brief summary of our existing release process:

- Nick Elias announces a "feature freeze" a few weeks before the release
- The developers diligently fix the regression tests while *ad hoc* testing is performed
- Errors, including new ones independent of the regression tests, are found and fixed
- The release occurs several (or many) weeks late
- The developers, now sufficiently stressed, are ready to face the next development cycle
- Repeat

Clearly, we can do better.

Our "release process" tax is very similar to the old "build system" tax. We spend a lot of time before each release trying to get it out the door. This time could be better spent (and possibly reduced) in a distributed manner, i.e., attempting to maintain viable prerelease branches throughout the development cycle.

To create this document, I started with my own ideas, listened to the developers, modified my ideas as I saw fit, and organized them. The test system naturally meshes with the release process. I expect additional discussion on the release process and related topics during the 2010 Developers' Meeting.

1. The CASA svn System

The svn system has been expanded and regularized, consistent with industry (Table 1). Further, the test system has been more tightly integrated with the svn system. The combined systems now provide more flexibility for both test and release. For example, it is easier to create release candidates over regular intervals when there is an extra branch that allows for earlier testing.

The developer builds are not part of the formal svn system, but they are mentioned for the sake of completeness. They act as private "sandboxes" for the developers. The active branch is the communal "sandbox" and formal entry point to the CASA svn system. The test and stable branches receive incrementally more testing. The stable branch corresponds to the current release candidate. And, of course, the release branch is made available to the general astronomical community. I do not mention the patches, which generally occur in the middle of development cycles.

Branch name	Executable name	Description	Update timescale	Users
developer	casapy	Developer "sandboxes"	~ daily	Developers
active	casapy (must run a script)	Common "sandbox" Lightly tested branch No guarantees, no JIRA tickets	~ daily	Developers; Internal illuminati who wish to review code changes very quickly
test	casapy-test	Moderately tested branch	< weekly	Testing proxies: NRAO, ESO, and NAOJ staff
stable	casapy-stable or casapy (via download)	Heavily tested branch Release candidate	~ monthly	Trusted visiting or external users
release	casapy	Release	~ biannually (release cycle)	General astronomical community

Table 1. CASA svn system branches and descriptions.

Consider the desirable trends that arise naturally from this progression of svn branches. The amount of testing and the update timescale *increases* from the early to late branches. Conversely, the user expertise *decreases* from the early to late branches.

2. Concerns and Solutions

In this section, I enumerate the issues related to the release process. I employ the same concern/solution paradigm as in the roadmap document.

Concern 1: We spend too much time, effort, and sanity trying to get each release out.

That's the diplomatic way of saying that our existing release process blows big time.

Solution 1: Modify the release process, taking into account testing, users, timescales, etc.

An extra branch (stable) has been added to the svn system to further separate more buggy code from less buggy code. Also, we incorporated unit testing (more accurately, component testing) early in the development process, which should minimize the number of errors that reach the slower regression tests.

It is always a good idea to produce releases and patches on time. To my knowledge, the CASA group has never successfully achieved this goal. The other extreme, producing a release only when certain features are implemented and tested, is quite radical and could lead to a highly irregular schedule. Therefore, I will advocate quasi-regular release dates unless they become untenable.

At present, we attempt to put as many changes as possible into the code before a feature freeze. If important features are late, the release is delayed. This procedure

has a major drawback, namely that other changes in the active and test branches languish for long periods of time.

Delaying releases for important features serves no purpose, especially if we manage to successfully create stable releases on a regular basis. New features will not be implemented any faster if a release is delayed. The only reason to delay a release is to fix a serious bug (e.g., the serious spw issues that delayed the 3.0.1 patch release).

Before each release, the CASA supervisor and CASA project scientist will see how many tickets have been resolved. If the number is sufficient and if no serious bugs have been reported by developers, users, or testers, the release will go forward.

Concern 2: We need more automated testing.

"What are your views on testing?" – *Bryan Butler*
"It's like voting. You should do it early and often." – *Nick Elias*

The above statements, which are only slightly paraphrased, occurred during one of my interviews for the CASA supervisor position. They neatly summarize my feelings on this matter.

Solution 2: Increase the amount of testing, especially early in the development process.

As mentioned in **Solution 1**, Sandra Castro (with help from Jonas Larsen) implemented unit testing. Unit-like tests from the regression testing system have been converted to unit tests. New unit tests using small datasets, which run very fast, have been created as well. More unit and e2e regression tests are on the way.

In Table 2, I show the testing that must be performed for code to pass from one branch to the next. There is a lot of leeway, so tests can be moved around, changed, or added as needed. The unit tests have already caught simple errors before they reached the regression system. I hope that this trend will continue and increase to the point where moving code from stable to release becomes almost effortless.

Branch names	**Criteria to pass to the next level**	**Who runs the tests?**
developer → active	• Unit tests corresponding to checked-in code	Developer
active → test	• Minimal low-level regression tests (from Wes) • All unit tests	Automatically
test → stable	• "Almost-all" e2e regression tests	Automatically
stable → release	• "Almost all" or All e2e regression tests • GUI tests	Automatically

Table 2. The required tests to move along the svn branches.

We already possess a significant number of unit tests. In addition to running unit tests before checking code into the active branch, developers are responsible for insuring that the unit tests cover the changes. If the unit tests are insufficient, the developers are also responsible for either for modifying an existing unit test or creating a new one. I do not expect a significant slow down in development. The number of new/modified unit tests will be small, and any time spent on new/modified unit tests will be recovered when bugs are found early.

In the table, I save the GUI tests for the last stage. It would be preferable to require GUI tests to move from test to stable, but since the timescale for that move is on the order of a month, GUI tests would take up too much time.

Concern 3: During extended freezes, development continues. Code is checked into the active branch and sits without significant testing from internal users and testers (because the active branch is often broken).

This scenario leads to inconsistent untested code in the active branch.

Solution 3: Modify the release procedures to eliminate this problem.

Here is my proposal:

- When a developer commits changes to the active branch, he/she will specify whether extensive testing is required. For example, initializing an uninitialized pointer to prevent a segmentation fault does not need a lot of testing. Modifying the CLI of a task, on the other hand, does.
- When the developer specifies extensive testing, the changes remain in the test branch. Those changes don't move to the stable branch along with the others until testing is complete.
- When the developer specifies extensive testing, it must be noted in a document either on the CASA Index wiki page or global JIRA ticket (global = a single ticket contains the entire list). The CASA supervisor and CASA project scientist will use this document to schedule and request extensive testing from internal users and testers.

Concern 4: Recently, the same regression tests appear to fail over and over.

According to the developers, these failures are not serious. The images are acceptable. Most of the time, regression values vary slightly as changes are committed. If a failure occurs on an unsupported operating system (e.g., Ubuntu), fixing it has a low priority. I believe that it is important, however, that we track and eventually eliminate repeating regression failures, even if they are not serious.

Solution 4: Track and eliminate all regression failures.

When the CASA supervisor receives regression-failure e-mails, he often asks the assignees why the failure has occurred. This information is not formally tracked and it is sometimes difficult to find it.

I have asked the test lead, Sandra Castro, to create a spreadsheet that will help upper management and me to determine the status of failed regressions. Sandra and I have already discussed the format of this document and how it should be maintained (this information will eventually appear in an appendix of this document). Unless she becomes swamped with other work, she will perform the maintenance.

Concern 5: We need even more "controlled" human testing (helpdesk tickets result from "uncontrolled" human testing).

Automated testing is a good thing, but we should never underestimate the exquisite ability of humans to break code. We have *ad hoc* testers, but they are often busy and aren't available all of the time like a dedicated test team. Also, don't forget that the GUIs must be tested interactively as well.

When an important feature is added, it would be nice if we could test it quickly in a controlled manner.

Solution 5: Find more testers.

A dedicated test team, as in industry, would be ideal. Unfortunately, creating such a group is not possible due to external requirements, staffing, etc.

As a stopgap measure, I have approached several RSRO people about testing specific features that have been added or modified. Success has been limited, perhaps because the number of people I approached thus far is small and like all astronomers they are busy. I'm wondering if trying to expand the number of potential testers would help, but I have reservations about this scheme. Suggestions are welcome.

---

3. The Closure Flowchart and the Release Process

In the roadmap document, I divided up the entire development process into three flowcharts: assignment, development, and closure. I will not discuss the assignment and development flowcharts because they are not germane to this document and we already follow them pretty well anyway. Taking the release process into account, I

show a modified and slightly simplified version of the closure flowchart (Figure 1). The color key is located in Figure 2.

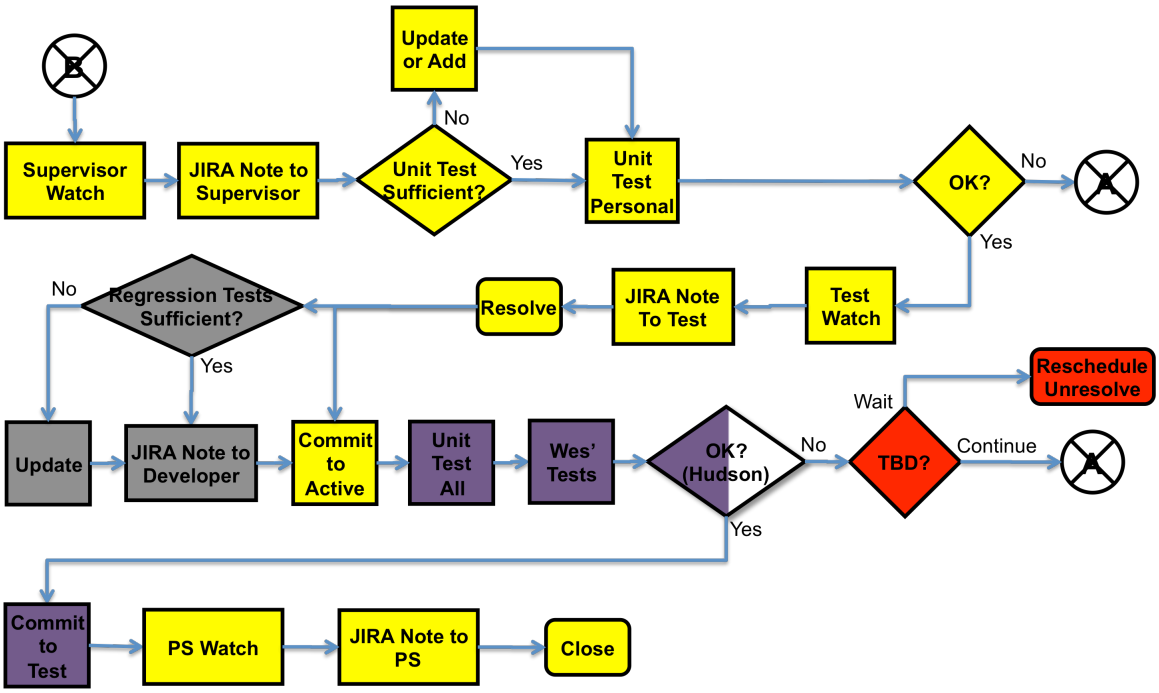

Figure 1. Modified closure flowchart of development process (originally from Roadmap document).

Figure 2. Color code for flowcharts.

The entry point is in the upper-left hand side, after the developer has finished development. He/she alerts the supervisor that development is finished by putting him on the watch list. The developer then determines if the existing unit tests are sufficient. If not, they must be augmented or added.

Assuming that the developer's unit testing is successful, he/she alerts the test lead (Sandra Castro) by putting her on the watch list. If the unit testing is unsuccessful, development resumes, otherwise the ticket is resolved (I am again looking into having a "ready for test" state). The developer commits the changes to the active branch, while the test lead determines if the regression tests need updating.

The system lead (Wes Young) runs all of the unit tests as well as his own regressions. If everything is OK, the changes are committed to the test branch and they move along the rest of the release process. Finally, the developer alerts the project scientist (Jürgen Ott) and closes the ticket. The project scientist is included because he needs to track tickets for the CASA Science Steering Committee and add significant changes to the documentation.

I decided to let the developers close the tickets after the commit to the test branch for three reasons. First, we now have a unit testing system that will cover more and more of the code over time. Second, waiting several days for the regressions to run before closing a ticket is unacceptable. Third, if any regressions fail, the ticket can be reopened, if necessary (that hasn't been happening as much recently).