Release Process

Beehive Shipping/Code management strategy

Assumptions

- A major release (X.0.0) would happen on the order of every 1 to 1.5 years.
- A minor release (X.y.0) would happen on the order of every 3 months.
- A “fix pack” release (X.y.z) would happen when warranted.
- Committers will run unit tests before checking in a change to provide for continued stability.
- We will designate a committer to function as the Beehive release manager. That person will be responsible for such tasks as calling for a release /branch, physically generating/labeling an X.y.z, informing community of release candidates and releases, calling votes on quality level assessments of RCs and releases.
- After the first GA quality version 1.x.z is cut and offically released, Beehive will make best effort to maintain backwards compatibility at all times. Beehive will have a strict policy of maintaining backwards compatibility within a major version.
    - Beehive can deprecate APIs. If an API must be deprecated, it will be documented as deprecated in one major release and removed in the following major release.
    - If Beehive must break backwards compatibility, Beehive will provide migration utilities where possible.

Goals

- Release early and often.
- Create branch to stabilize/wrap-up on a given feature set while new feature work continues in the mainline.
- Easy supportability for users on any formal major or minor release, i.e., provide a means to generate a “fix pack” on a major or minor release that doesn’t include new features.
- Achieve balance between having a more stable line that supports our users and moving the community and users forward on features.
- Flexibility and speed.

Non-goals

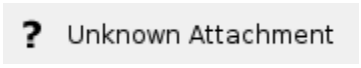
- Create a release on demand for any user or spin too many cycles creating fix packs on a major/minor release for only a few issues. (Instead we should encourage users who want immediate turnaround to use nightly builds.)
- Have active work going on in legacy branches simultaneously. (Typically work in the last legacy branch will taper off once a new branch is cut. The overlap may be about 3-6 months.)

Proposed approach

- Community decides to drive towards a release of a minimum agreed-upon feature set and on an agreed-upon approximate timeline.
- Once features are code complete, release manager calls the question of a soft code freeze/releasing/branching and community decides when to branch.
- Branch for X.y.z work.
- Fix bugs only in X.y.z branch and continue working in mainline on X<.y+1>.0 or <X+1>.y.0.

NOTE: There may be cases where it is critical to put a new feature into a branch. In this case fellow committers will be informed and if adding the feature would create an inordinate amount of risk another committer can request that the added feature be rolled back from the branch and put in the mainline only.

- Cut an X.y.z.
    - Release manager informs Beehive community that there is a release candidate.
    - Community works with it this release for a time.
    - Release manager calls on committers for a quality assessment of the RC (alpha/beta/GA), i.e., X.y.z will be cut and labeled and the community will use it for a bit before the committers give an assessment of the quality level.
    - If quality is not GA, team iterates on this process in the branch. If the quality is GA, release manager calls a vote for release. If vote passes, release manager informs larger community of Beehive release.
- Continue to fix bugs in the X.y.z branch. When appropriate, cut a “fix pack” release of X.y.z either by labeling or “branching from this release branch”. Integrate any fixes made in the branch either immediately or periodically back to the mainline.



Rampdown

Goals

- Explicitly stabilize.
- Begin to narrow the frequency of checkins to lock on a set of shippable bits.
- Drive the open bug count in a particular release to zero.
- Allow committers the freedom to pick & choose the bugs they wish to work on.
- Create an effective method of punting bugs to a later release.

- Have a means to document known issues.

Non-goals

- Control checkins
- “Force” committers to work on particular bugs

Potential approach

Assumes agreed-upon feature set and timeline and that the process is iterative.

- Committers take first pass at bugs by reviewing them, choosing bugs they would like to work on and assigning them to themselves if they want them.
- Mail to dev list regarding unassigned bugs remaining. Ask committers which bugs we must fix to ship and ask for volunteers to take them.
- All checkins must have an associated bug. All checkins must pass checkin tests.