R2014a Lessons Learned

Table Contents

How Lessons Learned are Managed

GMAT lessons learned include things that we did well and should keep doing, and large scale things we should be doing to improve the software or our project.   Lessons learned are each discussed by the team and if we decide there is a real issue, we require a plan for improvement.   To make sure we are efficiently handling lessons learned, here are some high level guidelines for creating them.

What is a Lesson Learned

Lessons learned are issues that cause significant problems or could have caused significant problems, or are issues where we did something that significantly improved the software or our project.   Lessons learned require group discussion and probably a change in team habits, process or strategy.

Lessons learned satisfy one the following criteria:

- Issue that is putting the project at greater risk than necessary
- Issue that is causing significant inefficiency
- Issue that is significantly lowering quality

What is Not a Lesson Learned

A lesson learned is not a minor annoyance, a tweak to an existing process, or something that can be resolved between team members in the everyday process of getting work done. Team members should bring these types of issues up at meetings, or work them among the team members involved.

A minor issue, (i.e. not a lessons learned), satisfies one of these criteria:

- Tweak to an existing process
- Minor annoyance or gripe
- Can be resolved by just picking up the phone, or discussing via email, or weekly meeting
- Does not require significant change in habits or processes

Things We Should Keep Doing

- (SPH) New sub-task approach seems to work very well.
- (JJKP) I enjoyed sitting in B23 one day/week. It led to some good working discussions.
- (TGG) automated regression testing continues to pay off, especially automation of system GUI tests.
- (JJKP) Working more closely with missions has been very successful.

Things We Should Change

Do Better

- (SPH) We are letting too many open tickets get scheduled simultaneously. Lot's of task switching as a result.   Our process is designed to have a few weeks of work on anyone's plate at any given time and to discuss/track progress on those items weekly. By letting upwards of 30 tickets on any one persons plate at a time, it breaks how we work.
- (SPH) Use Git as it was designed.   Some of us missed the push step, most don't create separate repos for different work areas.   The benefit of Git is not realized without the philosophy/approach change.
    - (DJC) As an addition to this, we could have simplified the code freeze piece by merging into production before building RC1, and then doing all testing and RC fixes in the production branch, rather than making the RC's off of master with a merge after the release was complete. Ongoing work could then proceed in master with no code freeze needed, and production would contain only contain the RC's and the final release. This would require some adaptation in the test and build systems.
- (JJKP) We need a task-master during the release process.
    - (SPH) Make sure nothing slips throught the cracks
    - (SPH) Send out status reports of what issues are being encountered and who is working them.
    - (SPH) Assuming we don't go to a more agile release process, and we keep a large annual release schedule, we probably need to define a few higher level roles with clear responsibilities.   Release manager, build packager, script test lead, GUI test lead, triage lead, and those key people need to be available and in the office during release staging.   Backups should have gone through process successfully.
    - (SPH) Consider having developers build installers. Engineers have lots of testing/analysis to do during RC phase so taking this off their plate would help.
    - (JJKP) Some proposed role definitions:
        - **Release Manager (RM)**: "owns" release, initiates process, creates tracking page, sends daily status updates, tracks issues to completion, makes sure everyone gets their stuff done, brings decisions to CCB/team, maintains tracking page, collects lessons learned, documents process improvements for next release
        - **Build Manager (BM)**: controls build system, creates RCs, sends RC availability announcements
        - **Test Managers (TM)**: control GUI/script testing for each RC
- (DJC) The switch over in the test system again happened much later than we needed. We should transition to the "test what we'll release" mode at least a month before code freeze – it was in the last week this time. Or we could just stay in a "release" test mode.

Start Doing

- (SPH) Need CCSDS TDM and SPK comparators for regression testing Ephemeris files. This is essential.
- (SPH) Consider refactoring Ephemeris component.
- (SPH) Nav development branching/testing needs to follow same process as the rest of our development.
- (JJKP) We find issues late and miss others that would have been found earlier by actual mission users. But mission users can't use a version that's changing rapidly. Two options:
    - beta period before release
    - more rapid releases
- (SPH) Consider keeping test system configured for release.   Some last minute issues were found because test system did not run in release config until after code freeze.   Even if we don't keep the test system in release config, we should move "**Switch to release configuration in script test system"** to be finished in QA complete step to make sure that we are not changing the test process during late stages, something developers have taken issue with for several releases now.

Stop Doing

- ?