R2013b 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

- ???

Things We Should Change

Do Better

- [JJKP] What happened with Code 500?
- [JJKP] Overall, the issue triage/CCB process got more nebulous with the switch to GreenHopper.

    - This is probably to be expected with a switch to a new tool.
    - Maybe all this needs is documentation:
        - How should feature leads be pre-triaging things?
        - What is CCB looking for?
        - How to make sure we're giving all the right info?
- [JJKP] Our new process based around JIRA tickets is a good thing!
    - But, I've noticed a tendency to fall back into a "mini-waterfall" mode:
        1. Engineer finds an issue, describes what he sees at the time, and throws it over to the developer.
        2. CCB gets to it and thinks it seems pretty important, suggests a fix based on the info that's available.
        3. Developer sees the issue, fixes it as described, and throws it back to the engineer.
        4. But wait! Fix had implications for another issue, and maybe wasn't as big of a deal as the description made it seem. Besides, engineer was pondering a different type of fix, but hadn't fully described it in the ticket yet.
        5. But now it's set to Resolved, and Engineer doesn't verify it for another week (or month). By then, it's too late to change it.
    - Agile emphasizes more communication, more often.
        - Would Code 500 have gone any smoother if I had sat in B23 and had more face-time with Linda?
        - Would TVHF have gone any better if Darrel and I were both on the same chat channel (for example) while we were on that feature?
- [JJKP] The "kindergarten problem" is a huge impediment. What can we do here?

- We made these mistake from R2013a all over again: Just repeating here
    - SPH: Critical path issues in ticket system put us at risk and we got lucky
    - SPH: Lots of tickets waiting for action until end release cycle.
    - SPH: Lot's of work for SPH and JJKP tracked outside of ticket system. The result is that P2 items in those lists get attention before P1 release items. For example, we sent a lot of time on SBIR support, but didn't thinking about testing until weeks to months after features were ready for it.
    - SPH: Spreading ourselves too thin. Don't do more projects and improvements than we can do well.
- New for R2013b
    - SPH: Single biggest issue: I (SPH) didn't react to misprioritizations and technical challenges immediately when I saw them. Which was in May.
    - SPH Need dedicated tech support so aero engineers can focus on their roles in the development process
    - SPH: Don't release during the summer... too much schedule uncertainty
    - SPH: Too much process change for a short release cycle.
    - SPH: Need to refactor as we go more often.   Putting in a new ephem file should be easier, we should have discussed redesign options.
    - LOJ: Agreed that need to consider refactoring as new types are added. (I kept that in mind when new types were added so that refactoring will be easier later.)
    - SPH: knew after R2013a that ephem test process was brittle and inadequate but didn't fix it. Made regression analsysis painstaking and manual late in the game for R2013b.
    - LOJ: Need more through unit testing
    - DJC: Ticket timing granularity needs to be handled better.
      Here's what I mean: When multiple people are responsible for a given item and the item is due on a single fixed date, dependencies between each person's responsibilities are easily missed. That leads to, for instance, a change in requirements too close to the delivery date to be feasible.

Start Doing

- ???

Stop Doing

- ?