4530 Week 12: Estimation, Planning, Teams

Agenda:

1. Administrative/logistics:
    1. Project deliverables + demo [See website: https://neu-se.github.io/CS4530-CS5500-Spring-2021/assignments/project-deliverable]
    2. Exam [See Piazza: https://piazza.com/class/kjt1yiu0x646wy?cid=627] 8-10am on 4/28 (Time set by registrar, not me)
        1. About 6 questions, some multi-part
        2. No writing code, some reading code
        3. Open book/open notes/Open linked references from the course website - no lockdown browser etc.
        4. Review suggestions: Make sure to have watched all of the lessons + be familiar with content from the review in-class, look over week-by-week learning objectives
        5. If unable to take exam due to time-zone issues, please contact me directly, ideally take during Wand/Boyland times
           Wand: 4/22 6:00-8:00pm
           Boyland: 4/23 1:20-3:20pm
2. Review Lessons 12.1, 12.2
    1. Q: What projects have you been involved in planning - in terms of estimating how long it will take?
        1. From work: sprint planning with story points

    2. From work: Agile cards (planning poker)
2. Q: How good do you think your past estimates have been? (Over/under estimate)?
    1. Experiences:
        1. At established co-op: “pretty good”, at startup: “not so great - under-estimating”
        2. “Always some bug that comes up” (Can try to build this in, but hard!)
        3. Writing tests takes longer!
3. Q: Do you ever time-box homework assignments (say “I will only put 10 hours into this”)
    1. Experiences:
        1. “That’s the dream…”
        2. “Yes, with essays” (acceptance criteria is not well-defined)
        3. “No because there’s a repercussion if I dont get a good score” (acceptance criteria is quite clear, want to meet it)
        4. If you wait to start until 8 hours before deadline…
    2. Compare to software projects - can carry-over tasks to future sprints
4. Q: What do we do in agile if we didn’t finish everything at the end of the sprint?
    1. We move to next sprint
    2. Break up the task
    3. Be open about what happened - try to get help, COMMUNICATE
5. Brooks’ Law: “Adding more developers to a late project just makes it more late”

   1. Why?
      1. Getting people up to speed
      2. Different people have different experience levels
      3. Software process scaling limitations
         1. Monolithic vs micro service application architecture
         2. “Too many cooks spoil the broth”
   2. How do we make this better?
      1. Break up team when project gets too big (“Two pizza teams”)
      2. Documentation/knowledge sharing - Effectively combine synchronous and asynchronous communication
      3. Limit interfaces between teams to limit communications scaling problems
6. Q: Should we always use this agile approach? (Sprint-based planning, updating estimation as we go)
   1. “NO”
      1. Handling bugs/incidents that arrive in an unpredictable way should be handled differently than new features that have business purposes/deadlines
      2. Short-term project (?)
      3. Big projects on a deadline where failure is not an option - might need something a bit more hybrid between waterfall/plan in advance + agile/plan as you go
   2. Caution: Buzzword bingo, getting lost in the

        details of your “scrum master” spending hours telling you how to be agile and have a standup meeting
    7. Q: What kinds of metrics are available to us in software engineering?
        1. Quantitative metrics:
            1. Sprint velocities
            2. Code coverage by tests
            3. Build success
            4. Bug-related stuff
            5. Cyclomatic complexity/other measures of code complexity
        2. Don’t use quantitative metrics in performance review
            1. Might not be including all/right metrics
            2. Impossible to quantitatively measure everything
                1. McNamara Fallacy - end up making bad decisions
                    1. Standardized test scores
                    2. Economics - unemployment rate, GDP
3. Team meetings

April 8, 2021 Course Meeting

Agenda:

1. Project deliverables update - page limits now up for documentation (note - these are maximum lengths, not suggested lengths)
2. Logistics for next week (No class on 12th), April 19th

3. Lessons 12.3, 12.4 review
    1. Q: Have you experienced a team project that went well? That is - good project output, plus good team dynamic. What went well and why?
        1. Well-planned projects where tasks are broken down into fine-grained increments and assigned help - and willingness to adapt [shared understanding of what needs to be done and by whom]
    2. Q: Alternatively - what goes wrong? Where do you find friction in a team project?
        1. “So many groups where one person…. Just doesn’t do the work?”
        2. “So many groups…. Where I’m not allowed to do the work? (Where someone just does it all)”
            1. Balancing act: Do I just do this myself, or do I help someone else to do it?
        3. Rush to get things working, no accountability for whose job it is to test/ensure quality of that code (if this isn’t planned)
    3. Three pillars of functional teams (Debugging Software Teams)
        1. Humility
        2. Respect
        3. Trust
    4. Focusing on project end goals vs team end goals
        1. Should I finish a project myself, or should I wait for my teammate/help my teammate to finish what they were supposed to do?
            1. Ideas…

        1. If it’s due soon, maybe we need to do it ourselves - especially if they seem busy or unresponsive
        2. Especially if one of your responsibilities is mentorship/learning, then this it’s probably important to have a focused plan for “transferring that knowledge”
        3. Depends on how communicative that teammate has been
            1. Teammate who is “trying”
            2. Teammate who is “not trying” - Not doing work, but ALSO not communicating
    2. Humility on the part of you, the person who knows how to do it, and trust from your teammate who is not
    3. “Man you totally got the control flow wrong on that method there. You should be using the standard Visitor pattern like everyone else”
        1. Don’t make it personal - remove the “you” here
        2. Don’t start with “Man”
        3. Don’t shame with “everyone else”
        4. “Should be” - don’t demand a specific change
        5. “Wrong” - is it black and white what is right/wrong here?
        6. Alternative: Explain why that method won’t work and a solution , “I recommend

            using the visitor pattern instead" ?
        7. "I feel..."
            1. I feel like the method is a bit confusing
            2. I felt confused when reading this method.
5. Post-mortem reviews
    1. Q: Why do a post-mortem review?
        1. Learn from mistakes
            1. What happened?
            2. What did we do? What happened after we did those things? What were we expecting?
        2. Avoid repeating them
        3. Example: Post-mortem after major bugs
            1. "We need to add monitoring for X"
    2. Blameless post-mortems
        1. Avoiding "Name, blame and shame" cycle
        2. https://aws.amazon.com/message/41926/
6. Engineering productivity - how to decide how to change processes
    1. Example: "We want to change all development from JavaScript to Haskell... code will be much better with fewer bugs, because of the design of the language" - how do we figure out if this is the case?
        1. Pilot study?
            1. Look at errors
        2. Framework to design a study - Goal/Signal/Metric
            1. Goals:

         1. **Qu**ality of code
         2. **A**ttention form engineers (distraction)
         3. I**n**tellectual complexity (is this harder than it needs to be/adding speed bumps)
         4. **T**empo and velocity (how quickly we do things)
         5. **S**atisfaction (how happy are engineers?)
      2. Signals - what we want to measure
         1. Is the code more readable?
         2. Is the code more maintainable?
         3. Are there bugs?
      3. Metrics - things we actually measure
         1. Consider both qualitative + quantitive metrics
   2. This is hard, ideally do this with sociologists
4. Team meetings