Intro to Methods of Software Engineering Fall 2018

Lecture 13 — November 13, 2018

Patrick Lam *version 1*

Real-World Software Engineering Practices

One of our key goals in the spaceship activity was to give you experience in using Git to share source code. As we said, a key part of software engineering is working together in teams to ship software. Not all companies do all of the things here, but they are best practices.

This particular formulation is the "Joel Test", which Joel Spolsky proposed in 2000: `http://www.joelonsoftware.com/articles/fog0000000043.html`. We'll apply a modified Joel Test to your Capstone Design Project in fourth year.

The Joel Test: 12 Steps to Better Code.

1. Do you use source control?
   As you've seen, source control enables you to collaborate with others productively, and also to go backwards in time to an older version of your code.

2. Can you make a build in one step?
   Should be: check out source code, build, deploy. As a software engineer, automation is your best friend. Automating builds eliminates errors and allows fast iteration.

3. Do you make daily builds?
   This avoids blocking your teammates and enforces the norm that your system is generally in a working state (not "I'll fix it later.")

4. Do you have a bug database?
   You'll forget about bugs that are not in the bug database.

5. Do you fix bugs before writing new code?
   Better to minimize interval between bug creation and bug fixing when possible; older bugs are harder to fix, because you forgot the context.

6. Do you have an up-to-date schedule?
   Joel Spolsky proposes "Evidence-Based Scheduling", which amounts to recording estimates and tracking how close you are to your estimates. Estimates can be notoriously unreliable.

7. Do you have a spec?
   Make sure you are implementing the right thing. It's easier to change the specification than the code. (Prototypes/mockups are useful, though.)

8. Do programmers have quiet working conditions?
   The concept of "flow" is super important for writing code. Typically it'll take 15 minutes

to get started. If you're interrupted, then you need another 15 minutes to get started again, which is a huge drain on productivity. People often wear headphones to code, but is that really the best thing?

9. Do you use the best tools money can buy?
   In the corporate context, tools are cheap, while developers are expensive. (Note that your salary is only about one-third the cost of employing you.)

10. Do you have testers?
    Automated testing has evolved a lot since this article was written; best practice today is to never use tests driven by humans according to a script (although some of you might still have such jobs on your first co-op term).

    On the other hand, exploratory testing is still a best practice today. Testers can be better than developers at exploratory testing, since they didn't write the software.

11. Do new candidates write code during their interview?
    You can see why this is useful from the company point of view. Let me address the other side. What if you're in SE and you can't yet write code? Consider this article: Mark Guzdial. "Anyone can learn programming: Teaching > Genetics," CACM Blog. `http://cacm.acm.org/blogs/blog-cacm/179347-anyone-can-learn-programming-teaching-genetics/fulltext`.

12. Do you do hallway usability testing?
    As with testers: get the software in front of someone else as soon as possible. One always has trouble seeing problems in one's own writing/designs/etc.

Again, the Joel Test is most effective for evaluating a team's effectiveness. A score of 12 is perfect; Joel writes that a score of 11 is tolerable; and yet companies do operate at scores of 2 or 3.

A high Joel Test score is necessary but not sufficient for writing useful software, though. No matter how good your process, if you're not solving someone's problem (product/market fit), your software won't get used much.

Software Engineering core courses

In some ways, the 3-course sequence SE 465 (3A)/SE 464 (3B)/SE 463 (4A) is at the core of the "software engineering" content in your curriculum.

SE 465 (Testing). There is no magic bullet for producing test cases, but there are tools that help, as well as some concepts that you can understand.

SE 464 (Architecture). Helps you think about designs and how to put together the pieces of your applications.

SE 463 (Requirements). Got to solve the right problem.