Christine Chen (chenc55)
Hours to complete: 3

Three difficulties that you have encountered during software development. Discuss (underlying cause, why people should care about it, why it is not easy to solve or work around, tools that could help, brainstorm possible solutions) and illustrate with an anecdote.

1. UI testing (I was having a hard time coming up with a testing difficulty—I think this is more of a reflection than a difficulty.)
One internship I did in undergrad involved UI testing. The first part of it was manual testing (performing certain workflows, test with certain inputs, etc.). This was a mind-numbing experience to say the least. The second part of it was working on automated tests. I can't remember what software we used. I do remember being surprised at how time consuming it was to set up the automated tests and get the software to understand how to do a sequence that was very simple for a human. But there was this feeling in the air that things were only going to get better and soon, we would be able to minimally rely on humans to perform testing (if at all). That was a good four or five years ago, and I don't know what the state of UI test automation is at the moment. However, reflecting on this experience now (of doing manual and automated testing), I think the human is really important in testing because the human brings a spontaneity and creativity that is rather hard to program (e.g. clicking a weird combination of buttons). This does make me wonder about the ideal delegation/delineation of testing tasks between the human and the automated system.

2. How to get another pair of eyes on code early on in development.

I think the idea of **pair programming** is good. Two pairs of eyes on code as it is being developed can be super helpful in catching bugs in real time. In addition, one person might know something that the other person does not that will make the code more efficient, secure, reusable, etc. The difficulty for me comes in the fact that I'm a pretty slow coder to start with, and I'm even slower when I'm still learning the concepts. While I have had some good experiences, for the most part pair programming has been a rather stressful experience that ends up with the other coder taking over as I do my best to keep up and understand what's going on.

In one internship I did, we had **code reviews** rather than pair programming. So we coded individually but then got someone else on the team to come and review the code (i.e. one person explaining the code to the other person). My experience with code reviews was that they were a bit short for the reviewer to actually get a good grasp of what was going on. It was also hard to feel invested sometimes as it wasn't the code that I created.

So the issue here is really how to get the expertise of another human into the loop as early as possible without defeating the purpose (because one person just takes over) or missing things (because one person is not invested/doesn't have enough context).
Thinking about pair programming specifically, is there some way perhaps to match programmer skill levels? From a quick Google scholar search, it looks like there has been quite a bit of research studying pair programming. I think this would be an interesting area to dig further into.

3. Setup reproducibility (for open source/research code).

Anecdote: A year or so ago, I went through the process of setting up a measurement tool that had been created by a previous lab member. It was a very difficult, time consuming process with lots of back and forth between me and the author (as I believe this was the first time someone else apart from the research team was trying to use the measurement tool). Finally, I got it to work. A few months later, I needed to set up the measurement tool again and, while the process was a bit smoother, it was still difficult. I definitely wished I had done a better job of documenting all the hacks that had made things work the first time. It was pretty frustrating as the whole software development process was held up by trying to reproduce an environment that I already had.

Underlying cause:

- The author forgot about some changes/installations that made things work.
- Poor documentation.
- Especially for research code—it is not the researcher's primary job to package up the code in a way that is easy to run/setup (the researcher should be focused on research!).

Why people should care: In terms of why researchers should care, my sense is that there is a push towards open sourcing research code and reproducibility of results. Being able to easily re-create an experiment is getting more and more important.

Tools that could help: Philip Guo presented at DUB last quarter and talked about a tool developed in his lab called Porta.[1] Porta allows authors of technical software tutorials to gain insights into how users are actually using the tutorials and where they are having the most difficulty. It does this by tracking navigation of the tutorial web page and recording information such as running shell commands and invoking the compiler. It uses that metadata to "annotate" the tutorial with information on where users got stuck, error messages they saw, etc. I wonder if this sort of approach could be used by a tool to keep track of all the packages that have been installed, versions of tools, etc. so that when a researcher goes to open source the code, the tool can help automatically populate important fields in the specs/requirements. And then, as others try to utilize the code, the tool can help them keep track of what they've done to try to get it working and (optionally) send back information to the author on where things are going wrong.

---

[1] "Porta: Profiling Software Tutorials Using Operating-System-Wide Activity Tracing": http://www.pgbovine.net/publications/Porta-profiling-software-tutorials_UIST-2018.pdf