Sep 29 2018

LEARNED AT FIGMA - WHAT MAKES A GOOD ENGINEER?

In my previous blog post, I talked about my learnings of the properties of Figma as a company. This second part will focus more specifically on things that I learned about engineering as a result of being at Figma.

Knowing what to work on

In school, we're taught tools to solve problems. During internships, we're given a chance to practice using those tools. And this is as it should be — you need to learn basic addition before learning algebra.

But what I came to learn is that the further out from school you go, the less you are provided with clear directions. The narrow path widens into an open field. So the skill that becomes increasingly relevant is not just "doing well something someone asked you to do", but "figuring out what the thing to do is to begin with". Fortunately, I've had plenty of chances to exercise this skill at Figma and watch other people do so.

On a small scale, this means asking questions like:

- Does completing this task help my teammates? (e.g. unblocking or refactoring)
- Does completing this task help us figure out uncertainties? (e.g. tell us something about whether our approach on a product or technical problem will work)
- Does completing this task help plan the project better? (e.g. task that are at risk of taking a long time are better done early than right before a launch)

And on a larger scale, it means asking similar questions but being even more strategic.

One of my team's big projects took 6 months, when very initially, we planned for a little over a month. Why were we off by a factor of almost 6? Part of it was just that we didn't quite have in mind the multiplicative complexity aspect of Figma and didn't plan for a lot of the small features we'd have to interact with. But what really took the most time was that 6 months was really three attempts that each took 2 months.

The problem was that we originally came up with some product designs, which people deliberated on and one was "approved". We started working on this, and ran into a lot of interesting technical challenges. Product challenges also came up as we found more designer use cases that we wanted to support without having too many ad-hoc options. In the process, we often deliberated and iterated on those details. All reasonable things. But when we did user testing, the high-level model turned out to be confusing to users and ultimately had to be thrown away.

There was nothing wrong with trying different approaches, of course. It's a natural part of the product process to iterate until you create something good. The mistake was that we prioritized the difficult technical challenges without realizing that the main area of uncertainty was the product. Since then, I've advocated for quickly prototyping a basic design with the aim of producing something user-testable as fast as possible. It turns out that once you're familiar with the codebase, if you ignore interaction with other features, good engineering practices and writing test, you can build a prototype of a feature in like a week. Sure, this might involve very dirty hacks like copy-pasting entire files to avoid spending time refactoring and other unacceptable things to do on the main branch. But those good practices can be done *once* you're sure you're working on the right problem.

Teamwork

One particular aspect of knowing what to work on is knowing how it impacts the team around you.

At a very basic level, this means things like:

- Work on tasks that will unblock your teammates first
- Refactor code that you think could cause confusion to your teammates

A little bit more advanced would be things like:

- Write a new testing infrastructure that makes it faster and less tedious to write tests (real example, it's called interaction tests at Figma and it's great)
- Replace a pattern of using raw pointers with weak pointers to make sure no one accidentally runs into undefined behavior (also a real example)

But those things are pretty self-explanatory in how they help. It takes some skill and experience to learn to recognize them, but it's not too hard. Harder is figuring out your impact as an individual agent in a complex system made up of many humans. A driver changing lane abruptly has no awareness of the traffic wave that propagates miles backwards.

One way to think about planning work efficiently in a company is that you have a highly non-linear, multi-dimensional function to optimize. Variables include things like what the tasks are (not always all known ahead of time), the time it takes to do each task (not always known ahead of time), the dependencies between the tasks (not always known ahead of time), the skills of available engineers, the desires of available engineers/what they want to work on, random stuff like urgent bugs or people going on vacation, etc.

As you'd expect, optimizing such a function is really hard, especially since the function contains lots of unknowns. So intuitively, you can imagine that the more you can reduce those unknowns, the more easily the function can be optimized. Even if it's not my job to optimize that function (it's my manager's and his manager), it certainly help the company if everybody pitches in a little bit.

Part of it is good communication. One of our engineering values is "communicate early and often". Communication reduces unknowns which generally helps everybody.

But the big one I learned is that it's important to be predictable in how long it takes me to do stuff. Up until now, I've always had an aversion towards making time estimates. And it is true that it's not always feasible, in which case forcing an estimate is worse than not having one. But it turns out that estimating is yet another skill among many, and the more you do it, the better you get at it. The better you get at it, the less painful it becomes.

Another aspect of being predictable is to sequence things appropriately. I often ran into situations where I saw opportunities for improvements to the codebase. But if I'd gone down and followed it right away, it would have added a couple days to do tasks and that would probably cause some traffic congestion elsewhere. So I made note of those things I wanted to improve and during down times when there was less inter-dependencies, I actually made those improvements happen.

How far technical skills get you

In these last few months, I learned skills about product engineering as a full-time engineer that I hadn't had the chance to exercise as much as an intern. And if you squint at them a little bit, a lot of these skills are what you might call soft skills. The traditional advice isn't wrong: soft skills are important.

But I find that a lot of the time, this advice is framed in a context as exclusive to raw technical abilities, such as being a fast coder. No one encourages you to practice competitive programming after school, presumably because there are more 'important' things to improve on.

What I'm finding from working (i.e. observing other people) at Figma is that raw technical ability seems to unlock more than increased production. It makes it possible to do things that in practice just don't ever get done if they're anticipated to take too long.

Consider, for example, building a prototype for a feature. If the feature is fundamentally very complicated product wise, it shouldn't matter if it takes 2 days or 2 weeks to build it as long as that *is* the optimal thing to do in terms of expected value . In practice, it's hard to feel comfortable spending weeks building prototypes due to the worry of wasting time on a failed experiment, so that step easily gets skipped over.

There's other types of work, such as refactoring some core internal API or abstractions, that interfere a lot with ongoing work by other members of the team. An extreme example, for example, would be to rewrite the application in another language. This has to be able to get done fairly quickly, otherwise it just causes too much disruption for it to be ever worth it (or be perceived to be worth it).

Found this blog post interesting? Please leave a comment! Remember to check out figma.com/careers and feel free to ask any questions!