An Introduction to Professional Software Development

Peter Bell
Aug 21, 2019 · 6 min read ★

What is it really like to write software for a living? How do you spend your days? What skills best correlate with success? Read on to find out more- the answers may surprise you!

When I speak to people who don't program for a living, they often think of a programmer as some kind of lone wolf, slinging byte-code in a windowless basement, fingers speeding over their backlit keyboard, with the green light of their terminal window reflecting in their glasses. That's a pretty decent summary of how Hollywood usually portrays programmers, but it doesn't have much to do with writing code for a living. So, what exactly is a software engineer, what's it like to be a professional software developer and what skills will have the biggest impact on your success?

```
require "rails_helper"

RSpec.describe CampaignSegment, type: :model do

  describe 'Associations' do
    it { should belong_to(:campaign) }
    it { should belong_to(:segment) }
    it { should belong_to(:drip) }
  end

  describe "Scopes" do
```

```
identity_spec.rb
location_spec.rb
message_spec.rb
person_drip_spec.rb
person_segments_spec.rb
person_spec.rb

    expect(campaign_segment.segment_dri
  end

  it "returns the drip for the campaign if the segment doesnt have a drip" do
    drip = create(:drip)
    campaign = create(:campaign, drip: drip)
    campaign_segment = create(:campaign_segment, campaign: campaign, drip: nil)
```

What does a software engineer do?

You'll usually turn up to your office and catch up on some emails and slack messages before kicking off the day with "stand up." In a standup, the team gets together and shares what they did yesterday, what they're working on today and any blockers that are stopping them from getting their projects finished. Standups usually runs less than 10 minutes and they're a great way to get a sense for what the team is doing, to share hints and tips with your team and to ensure that if you're blocked, someone will get you going again so you don't waste too much time.

After standup, it's probably time to write some code. In some teams, you might do some of that solo, but you'll probably spend at least some time pair programming. When you're pairing, you have two developers with just one computer. At first that might sound like an inefficient use of time, but it turns out that there are studies showing that pairing is at worst only slightly less efficient that individual coding, and any inefficiency is more than paid for by the substantial reduction in bugs. On some teams, you might even do a little mob programming where 3, 4, 5 or even 10 developers work together, sharing ideas with a single "driver" typing in the code. Pairing and mob programming both show that the real value of coding is in the thinking — not the typing!

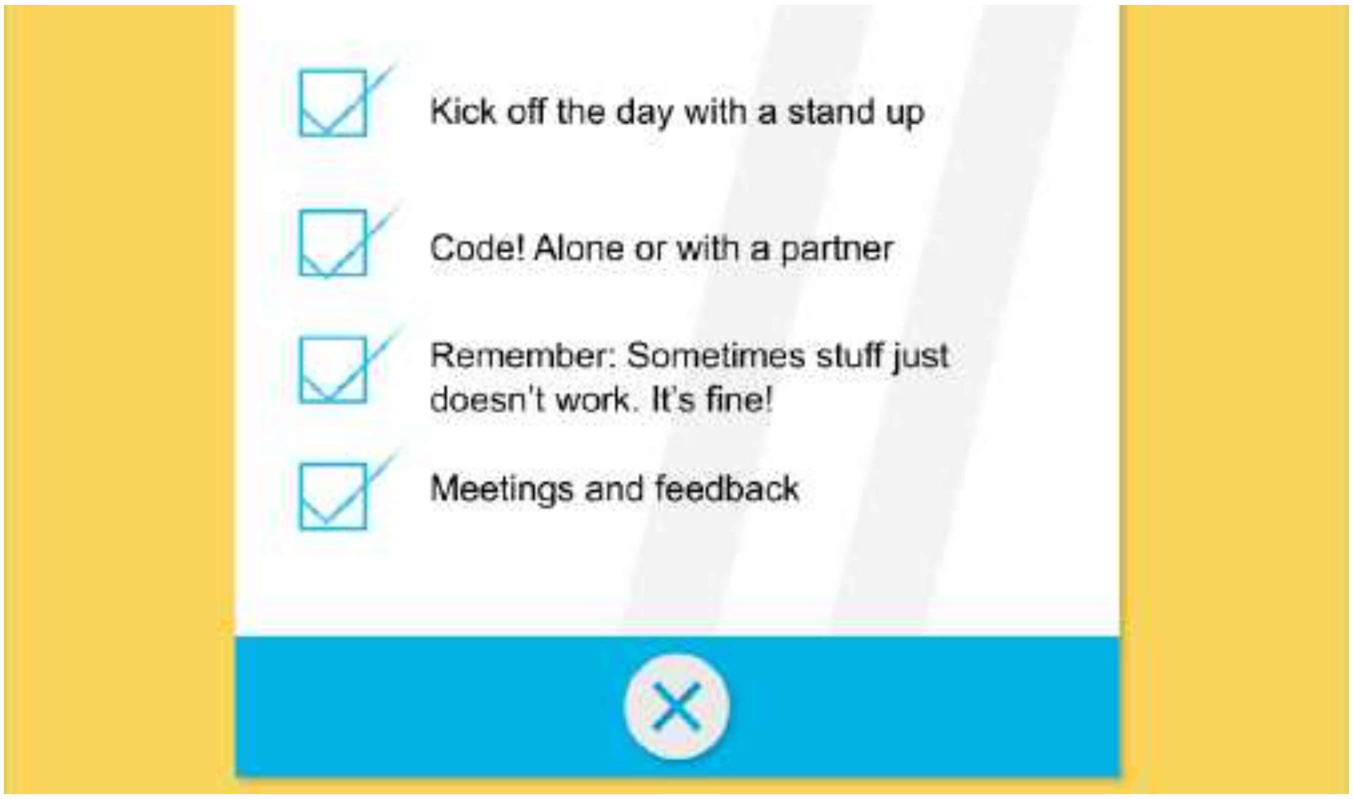
Some days will be great. You'll get some business requirements, perhaps some mockups from a designer, and you'll get into a flow of writing a test and then the simplest possible code to make it pass (Test Driven Development). New features will flow from your fingertips and you might wonder how it is that you get paid to do something that's so much fun!

There are other days too. You're throwing together a simple feature, you find a post on StackOverflow and some instructions for the library to use and estimate that it's at most a half hour of work including testing and deployment. Four hours later you're wondering if Starbucks is still hiring for Baristas, because you're clearly not cut out to be a computer programmer — you can't even follow simple instructions!

Don't worry — it's not just you, and it's not because you're not intelligent or inexperienced or don't "have the talent." It's the dark side of coding. Sometimes stuff just doesn't work. I wrote my first code 40 years ago, and just this week I managed to spend two hours completely failing to add Stripe (a payment processor) to a Rails app — and there is almost nothing easier to do as a programmer than add Stripe to your app!

The good news is that if you take your time, methodically debug what's happening, and, if necessary, are a little flexible about your implementation (maybe we could try a different library or tweak the UI to make the problem easier to solve) you'll figure it out eventually!

In the afternoon, you might have a couple of meetings scheduled to discuss features with a business analyst, product owner, or a business stakeholder depending how your company is structured. It's a chance to learn more about the underlying problem they're asking you to solve, and to provide feedback based on your experience to come up with better ways to solve the problem that might be quicker or easier to implement. That sums up what a software engineer does on a day-to-day basis.


Kick off the day with a stand up
Code! Alone or with a partner
Remember: Sometimes stuff just doesn't work. It's fine!
Meetings and feedback

What is a 10x developer?

This is a pretty good time to bring up 10x developers. They do exist — I've been lucky enough to work with a few of them. How can they be ten times as productive as the average software developer?

Well, it's not that they type ten times as fast or work ten times as long. It's simply that they add much more value by being thoughtful about the best way to solve a problem. If this is a quick spike to test user demand, could you just use Typeform instead of coding up a custom form? If you want to implement 2FA (two factor authentication) is there a prebuilt library or a SaaS product you could integrate with to reduce the time to market?

What is software development?

It’s also a pretty good time to bring up types of software development. Tuning algorithms for Google’s core search product, building websites and mobile apps for most businesses and writing embedded C or C++ for a microcontroller are very different jobs.

If you really want to be a Google software engineer (or work at any other tech giant) at the heart of their most business critical algorithms, the job will be much more like that of a research scientist. You’ll be coming up with hypotheses, running simulations and creating (or customizing) algorithms for a scale of problem that few people in the world are dealing with. You’ll probably also have an extremely strong academic background in both mathematics and computer science. And if you’re looking to write C for microcontrollers you may have to learn a little more about the hardware than you’d expect.

But for the vast majority of software engineering positions, you’re going to be writing front end, back end or mobile code — typically for displaying lists, detail pages, forms and perhaps admin panels and dashboards. These are well-researched and understood fields. There are great libraries and frameworks to help you to do the work, and you’re pretty unlikely to come across a problem which will require you to implement a binary sort or a linked list from scratch.

So what makes a great programmer?

It turns out that for the vast majority of software engineering roles, the best computer programmers have a few traits in common. They are:

- **Humble** — If you can't admit that you're wrong, it's going to take you a really long time to learn new things. And learning is a great software engineer's superpower.
- **Curious** — Bugs are a chance to learn something new. Whether it's about your inability to remember where to put semicolons or an unexpected interaction between a new version of a framework and a library. It's important to be able to "not know" whats up and to enjoy the process of figuring it out, and curiosity can really help with that.
- **Passionate** — You shouldn't be pulling all-nighters, and it's important to have a balanced life, but if you're not passionate about tech (and ideally about the mission of the company you're working for), you're going to find it hard to keep coding when you're having a tough day.
- **Empathetic** — Perhaps more than anything else, the best programmers are able to empathize with both business stakeholders and end users to come up with efficient ways to make their software more capable and more enjoyable to use.

Now don't get me wrong. If you're terrified of computers, can't break bigger problems down into smaller, more easily solvable ones or hate Googling for answers or fixing bugs, you're not going to have any fun as a programmer. But if you choose to learn to code, once you become competent in your primary language, your life experience to date and your communication skills and determination will be the most important elements in helping you to become a really great programmer.

. . .

Originally published at https://flatironschool.com.

Programming