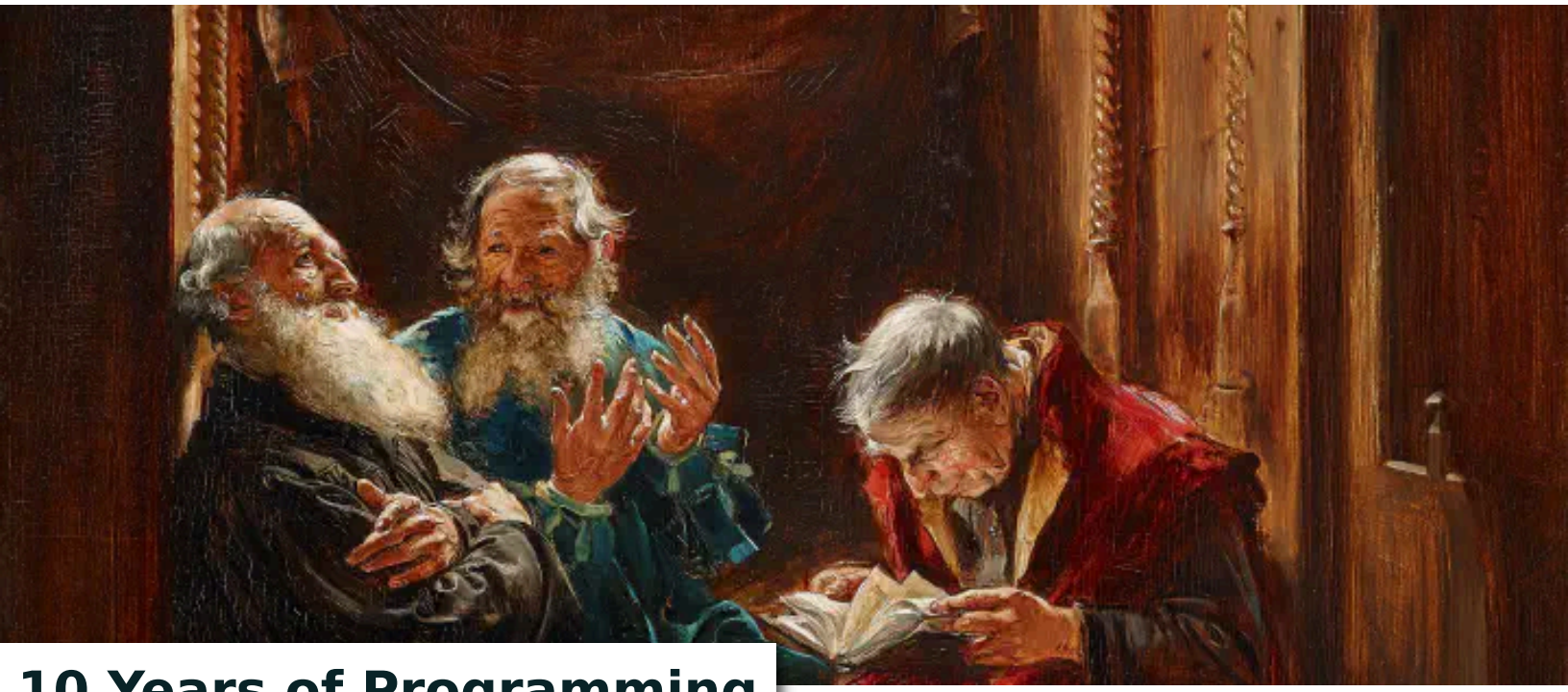

10 Years of Programming

Posted 26th September 2024 in Advice, Programming
Cover image based on *Meinungsdifferenzen*, Leo Reiffenstein (1885) Belvedere, Vienna

As of the beginning of September, I've been programming "with purpose" for a decade. Before then it was just a hobby — something I could do in my room while burying my head in the sand about the Physics degree I was failing at the time. But that summer, I realised maybe if I was choosing to spend all my free time coding, maybe it was worth trying to pursue that as a career instead.

Okay, so maybe ten years isn't all that long, but since I started studying software development properly, I've learned a lot, and I want to reflect on that a bit. I suspect not everything I learned is correct: come back in ten years' time to read my critical retrospective where I point out what an absolute idiot I was/am. But in the meantime, here are some assorted thoughts on how I practice programming.

- **I write software for people.** No, literally, I write code thinking about the people who will read it. When I write a blog post or an email, I've got a target audience in my head. Why not do the same thing with code? Is this clever type-level shenanigan really the best idea? Will my colleagues be able to quickly extend this test file with new cases? Would a comment be useful here?

- **The more I automate, the less I have to think.** I don't think I can write code without an autoformatter any more. The junk that comes out of my keyboard is bad — all on one line, indentation all over the place, terrible spacing. But an autoformatter does the tedious formatting work for me. I don't even care what that formatting is — I will accept basically anything, as long as I don't have to think about it. See also: automatically importing functions I'm using, as well as automatically removing imports I'm no longer using.

- **The more I automate, the less I have to think: Round 2.** This goes doubly if the stuff I'm automating is stuff I only rarely do. I have a few projects that I occasionally do stuff with — a LaTeX CV that I only update when I'm looking for a job, a website that only needs occasional changes, etc. Automation (a Makefile, a CI build/deploy pipeline, etc) means I don't have to reverse engineer the project every time I come to work on it again.

- **Some things aren't automatable.** Whenever I commit my code, I diff my code and try and review it to see if I've made silly mistakes, left debugging code in, forgotten to refactor something, etc. Then, when I create a pull request, I do the same thing again. Linting isn't a good substitute for this — it can catch some of the basic stuff, sure, but it won't catch anything. I haven't (yet) found a way to automate away simple diligence.

- **The right programming language is important, but probably less so than I'd like to believe.** When I first learned to program, I used Python, and then I ended up in the frontend world and started writing a lot of Javascript, then Typescript, and it just hasn't really stopped. And you know what? Javascript is not a great language. But it's good enough. I write a lot of my own stuff in Rust and miss features of Javascript, then spend my day job missing things in Rust (or Python, or whatever else).

- **Types are non-negotiable for large projects.** Okay, yeah, all programming languages have something good about them, even the bad ones, but the more people you work with, the more tools you need to communicate how code works. And types are possibly the single most effective tool to communicate how code works. Tests are cool, documentation is great, but both can be ignored or overridden. Types (okay, well-written types) stick around and enforce their rules on the codebase.

- **I like tests when they're good.** I've written before about how I didn't really get testing until I understood how to write a good test. I'm currently rewriting a core module in a system at work, and having 100 or so tests that each test one aspect of this module, that I can slowly reenable as the new functionality comes online — it makes my life easier, and gives me more confidence that my new version behaves the same as the old.

- **I *hate* tests when they're bad.** I still struggle to delete tests — it feels a lot like cheating — but sometimes they just aren't helping. The accuracy and precision of tests in large codebases would be a great topic to study, because I suspect a lot of tests are never once useful in their lifetimes in a codebase.

- **The simplest code is usually the hardest to write.** There's that famous quote "I'd have written a shorter letter, but I didn't have the time". I have similar feelings about code simplicity. Simple code isn't about writing less code (that's usually code that's missing edge cases). It's also not necessarily the easiest to read (see Torvald's "good taste" example, which is simple, but uses more indirection than a more naive alternative). Simple code is good, but it usually takes a lot of work to create it[1].

- **Perfection is a good goal, but a bad obsession.** I really like doing things the "right" way. I'm one of those people who'll come up with an idea, create an empty repo for it, and then spend the next three hours getting the tests, linting, and CI working. I say that partly in jest, but also partly in pride. Perfection is also about little things like choosing the right semantic HTML element rather than adding an `onClick` handler to a `div` ; or spending a bit of extra time to get the commit history looking cleaner; or finding the O(n) algorithm instead of the O(n^2) one.

  But I've also got to limit myself on these things. Sometimes it's not worth the extra effort. Sometimes, it's not even possible to get things perfect. Often, there are competing needs, and I just choose the needs that feel most important at the time. I recently saw a comment where someone listed their ideal websites, and it included things like not using CDNs, and ensuring that the site works on terminal browsers like Lynx. The perfectionist inside me understands these goals, but the pragmatist has won out, at least for this website.

---

1. As an aside, this is one of the issues I have with the "grug-brained developer" concept — writing simple code is not a matter of just not doing the complicated things, it's about having enough deep knowledge and experience that you can see the simple option. This is a difficult, learned skill. To be clear, a lot of the advice given in the original document is good advice, but I don't find the "just write simple code because you aren't smart" framing accurate or helpful. ↩

Share this article on Reddit, X/Twitter, Hacker News, or Lobsters.

Previous Discussions

This work is licensed under CC BY-SA 4.0