14 Habits of Highly Productive Developers

Including tips by top developers from
Google, Microsoft, Adobe, Shopify and more

Zeno Rocha

Part One: Principles

Hello World

“There is no elevator to success, you have to take the stairs” — Zig Ziglar

I started developing software more than ten years ago. I built many sites, created dozens of open source projects, and pushed thousands of commits. Besides that, I spoke in more than a hundred conferences, and had the opportunity to chat with a ton of developers along the way.

I was fortunate enough to be in contact with some of the best software engineers in the industry, but I also met a lot of programmers who are still doing the same thing for many years.

So what separates one group from the other? What's unique about people who work on tech giants such as Google, Microsoft, Spotify, and others? What's special about people who create the most used applications in the world? How can some developers be so prolific at work and also outside their jobs?

These questions stayed on my head for a long time. I realized that I could buy the best mechanical keyboards, go to the most famous tech conferences in the world, and learn all the newest frameworks. Still, if I cultivated bad habits, it would be impossible to become a top developer. Because of that, I decided to reach out to the best developers I know and ask them tips on how to be more productive.

This book doesn't offer a straight path or pre-defined formula of success. This book is a result of a quest. A quest to uncover what habits can be cultivated to become a better software engineer.

Why Habits?

“First forget inspiration. Habit is more dependable.
Habit will sustain you whether you’re inspired or not. ” — Octavia Butler

If you ever tried to lose weight, you know how frustrating that entire process is. You can exercise as hard as you can for three hours, but if you do that only once in a week, it will have zero effect on you. What truly generates results is when you go multiple times per week. Then suddenly, a few months later, you'll start noticing changes in your body.

Consistency matters, and that same concept applies to your professional career as well. Things take time, and intensity is not always the answer. The habits you decide to cultivate (or don't cultivate) will determine your future life opportunities. As described in the book Atomic Habits:

> *"Habits are the compound interest of self-improvement. The same way that money multiplies through compound interest, the effects of your habits multiply as you repeat them. They seem to make little difference on any given day and yet the impact they deliver over the months and years can be enormous. It is only when looking back two, five, or perhaps ten years later that the value of good habits and the cost of bad ones becomes strikingly apparent." — James Clear*

I often get emails from other software engineers asking what programming language they should learn. That's a very important question to ask yourself indeed. However, whatever language you choose today, it's invariably going to change in the future. The most valuable question would be: "What habits do I need to cultivate in order to be effective in *any* programming language?"

That's why I decided to focus this book on habits instead of tactics.

Now let's dive in! Are you ready?

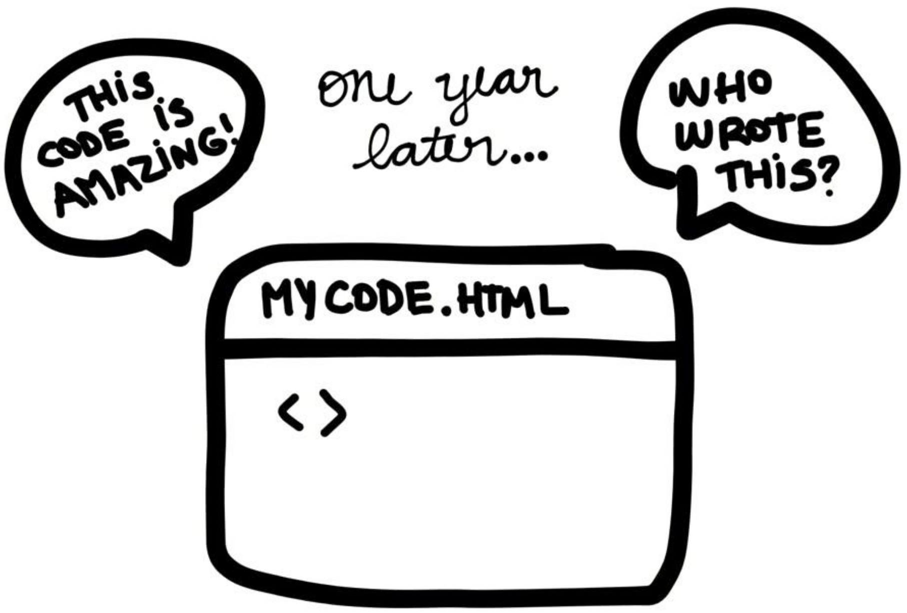
HABIT #1

Do it for your
future self

Habit 1: Do It For Your Future Self

“When you feel the need to write a comment, first try to refactor the code so that any comment becomes superfluous.” — Martin Fowler

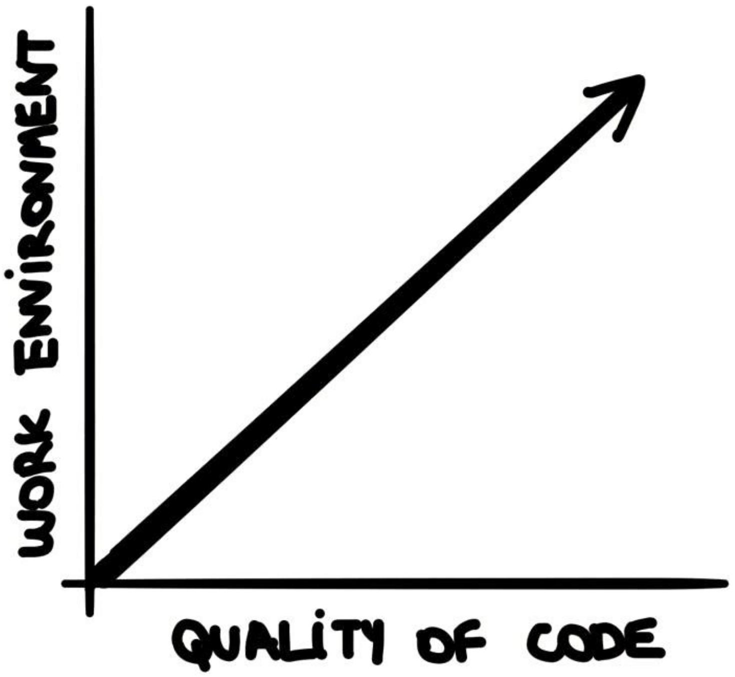
I don't like to judge other people's code. I don't say this to prevent people from judging my code either. I say this because even the best programmers in the world have written crappy code in the past. And that's not because they were still learning and didn't know all the best practices. The reason why we all write crappy code is related to the circumstances around us on that particular week, day, or time.



Imagine trying to write code that is simple to understand, well documented, fast, and modular, while you have a lot of anxiety going on in your life. It's very hard to be efficient as a programmer when you are facing a lot of pressure from your boss and have crazy deadlines to meet. Still, many of us go through those experiences at least once in our lives. For those of you who are still being treated like this every day, let me tell you something - there are better places to work out there.

Now let's assume that you are not in a toxic work environment. Let's pretend that you're working on a side project, and there's absolutely no external pressure for you to finish this software. You open your code editor and start programming. Ideas begin to flow, and everything seems to *click*. After hours of coding, you look at all the files you wrote and feel proud of it. Everything makes sense; it's easy to read, easy to maintain, easy to evolve.

Suddenly, there's a change of priority, and you abandon that idea. You get involved in other projects, you start using different languages, you try new frameworks. A year later, you decide to continue what you started, so you open that project again, and it's unrecognizable!



We have all been there, really. And the root cause is that you're writing code for your current self. Instead, you need to write for your future self. Your current self has all the context needed to understand that block of code, your future self is involved with other things and doesn't even remember what that function X even means.

Don't try to be clever, don't try to code something to feel smarter, you don't need to show all the new tricks you just learned. Just write readable code, think about maintainability, and use meaningful names for your classes, functions, and variables. Next time you start developing, ask yourself this question: *"Will the future me understand the intention of this code?"*.

// TODO: Open a current project that you're working on. Is there any refactoring you could do in order to make the life of your future self easier?

// Question: What are the things you do today to help yourself in the future?

Daniel Buchner (Microsoft):

> *"Obviously, everyone tries to build something that will make their current deliverable a success, but to help ensure the future of your work is more predictable, with better long-term outcomes, a critical skill to develop is the ability to look around corners. There are likely many paths you can take that will lead to success for your present deliverable, but choosing the one that sets up the next version for success, and the five after that, is more challenging.*
>
> *When thinking about this, I tend to look at larger industry trends, the progression of where I believe technology will be in 3-5 years, and what emerging standards appear solid enough to take a bet on. You're not always going to find something actionable to incorporate for every project you undertake, but when you do, a small modification to your course early in a project can significantly change where you end up over time."*

Lais Andrade (Google):

> *"One of the best things to keep the codebase future-proof is to have good standards for best practices, which are followed by the team. This goes beyond styling, all the way to which pre-existing tools to use for certain known problems. Reinventing the wheel is something most of us are easily compelled to do, but a quick search can reduce the amount of new code added, code which would require future maintenance. It also helps in avoiding familiar mistakes that were already solved by good libraries.*

The second key point is very famous, but surprisingly difficult to put into practice: early optimization is the root of all evil. It's easy to write a one-liner without comment that does exactly what we need, but it's very hard to debug such a line a few weeks down the road. Peer reviews are extremely useful to avoid such pitfalls: whenever you need too much context to understand exactly what an incoming line/method/component is doing, you should step back and ask them to add more intermediate variables with meaningful names or to add a few comments explaining why it's done that way, or even to break it down into smaller and simpler components.

The third and last aspect is to always test everything I write. Unit testing, integration testing, performance/benchmark testing, all of it. Enforcing a simple policy of "no new code without unit tests", for example, already works wonders. You can definitely notice the difference between code bases that apply it and the ones that don't. If tests are taken as a fundamental aspect of the project and are given the right priority, then it's easy for the team to spare some cycles working on better testing tools (quality of test code is as important as the quality of production one) and infrastructure (like continuous integration, etc). No code added without unit tests, no bug fixed without a regression test. Such small steps already have a huge impact on the overall quality of the project."

Thanks for taking the time to read the first chapter.
I can't wait to share the rest of the book with you :)
Wanna help? Share it with your friends!

- Zeno