Photo by James Pond on Unsplash

6 Programming Habits That Make You an Ineffective Programmer

Take note of these so you can change yourself for the better

Daan
Mar 23 · 5 min read ★

We all have both good and bad habits. Programming habits are no exception to this. But once you start being aware of your bad habits you could change yourself for the better. If you work on breaking one of these bad habits in this list you will not only impact yourself. Most likely it will also impact the people around you.

And because bad habits are easier to break today than tomorrow, we'll go over this list of six programming habits that make you less effective than you could be.

. . .

1. Attending Meetings

Meetings — they are probably the number one productivity killer. However, most developers still attend way too many meetings. There are two types of developers when it comes to meetings.

The first group will skip every meeting to spend time behind their keyboard. This group thinks most meetings are a waste of time and that it’s better to do some actual work.

The second group is quite the opposite. This group seizes every opportunity to attend every single meeting that is scheduled.

If you find yourself to be in the second group you’re wasting a lot of time that you could have also spent on writing code and being productive.

The thing with meetings is that most meetings are scheduled for an hour by default, even if the agenda could be handled in less than half an hour. The thing with meetings is that we can say *no* to a lot of meetings. Or maybe start saying *no* to meetings before noon, so you can be productive in the morning. And if you really have to say *yes* to a meeting, at least say *no* to long meetings.

> Meetings are indispensible when you don’t want to do anything — John Kenneth Galbraith

. . .

2. Over-Engineering

Over-engineering is one of the bad habits that many developers tend to have. When looking at a codebase you’ll find over-engineered pieces of code more often than not.

What over-engineering basically comes down to is that you make the design of a product more robust or complicated than is necessary. One way over-engineering gets introduced into a codebase is when a developer is already adding code that he thinks might be helpful in the future.

This additional piece of code gets added to the codebase but it probably never gets used. Most of the time the reason that more gets build than what is really necessary is based on speculation. Maybe the best way to explain over-engineering is that it is code that solves problems that don't exist.

Over-engineering can lead to code that is being designed to be so generic that it loses sight of the main task that it was initially designed to perform. Therefore it becomes not only hard to use but fundamentally unintelligent.

. . .

3. Writing Your Own Data Structures

Writing your own data structures falls in the category of reinventing the wheel. It's an extremely inefficient habit to have. All the data structures you need are already out there and ready for the take. Most of the time there's no need to recreate a certain data structure.

And data structures aren't the only example where developers try to reinvent the wheel. Way too often developers tend to recreate certain pieces of code.

If the same piece of code already exists and is known to be stable and well maintained just go for that route. Your version adds nothing new, or even worse it is missing features. The only new things that it potentially introduces are bugs or constraints.

Reinventing the wheel also has a positive side. It's perfectly fine to reinvent the wheel if you want to get a deeper understanding of something. But most of the time this is discouraged because it takes too much time. Sometimes that time cost can be justified, and sometimes there is no way to justify it. In other cases, the task is so critical that getting it wrong can have a terrible consequence — which makes reinventing the wheel not your best option.

If you want to break this ineffective habit, you're probably better of by not reinventing the wheel as a default.

. . .

4. Inconsistency

Consistency really is key when it comes to software development. The problem with being inconsistent comes from the unavoidable fact that time destroys software. The longer a piece of software exists, and the more people that work on it, the more chaos comes in.

> Consistently bad is better than inconsistently good

It's good to know that consistency has a high impact on the maintainability of your codebase, especially in the long term. If you've decided to use camel casing for your variables, stick with it. Want to use spaces instead of tabs? Great! It doesn't matter what you're doing in your code, at least do it consistently.

. . .

5. Not Planning

Rushing into a coding project might seem exciting at first. However, that excitement might lose you a lot of time. If you jump straight into the coding part you'll eventually lose sight of the bigger picture.

Code needs to be planned and organized before you begin. How can you tackle this certain problem? What structure will you implement? What are the overall goals that you're implementing?

These are all great questions to ask before you start coding. And these questions can make you more aware of the fact that there's a lot to think about before writing code.

When you fail to plan, you'll end up with some feature that's not exactly what the customer asked for. Or even worse: your solution is not the right one. This results in you having to come back at that piece of code later on and fix it — which is inefficient.

. . .

6. Not Asking For Help

Every developer, no matter how experienced, will be stuck every once in a while. When you're in a situation like this it's important to keep a short feedback loop.

Asking for help doesn't mean that you're incompetent. People think you're incompetent if you're staring at your screen struggling with the same problem for hours.

Before you ask for help, make sure you've checked all the things that you know of. Disturbing other developers unnecessarily is not what you want to accomplish.

More often than not some other developer can give you a push in the right direction. This way you save yourself a lot of time because you can start working on your task again instead of figuring it out all by yourself.

> Help only comes to those who ask for it — J.K. Rowling

Productivity Programming Software Development JavaScript Web Development