Even Really Good Programmers Make These Mistakes

Common mistakes all programmers make, both in coding and theory

Ryan Nehring Follow
Nov 7 · 5 min read ★

Photo by PxHere

Everyone makes mistakes and we programmers are no exception. We tend to think that if our code runs, we've avoided mistakes and can pat ourselves on the back and enjoy a well-earned victory drink.

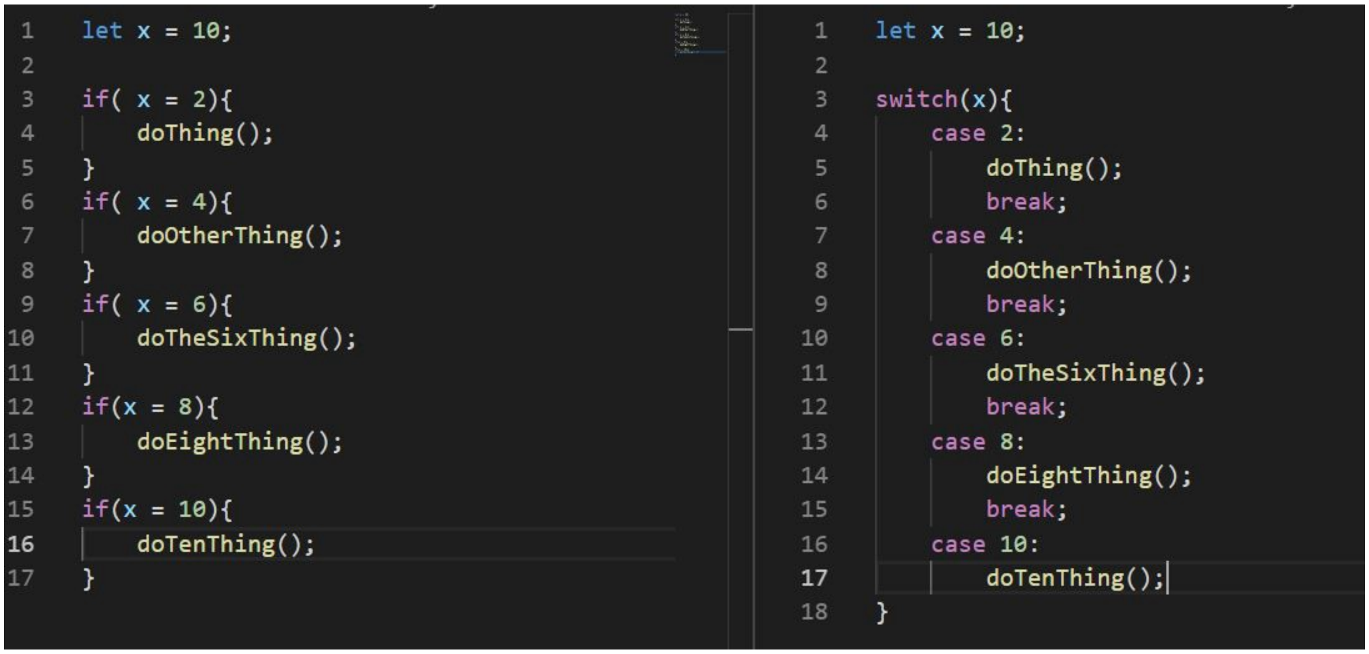
However, code that "just runs", should not be the benchmark by which we measure our success! Non-breaking mistakes are still problematic, especially to maintainability down the line. No one writes perfect code, but there are some mistakes and oversights that we as a community are particularly prone to commit. They are easily avoidable and they make tangible differences to the quality of our projects.

Let's examine some of the most common mistakes programmers, of all skill levels, make. I'm going to separate the list into two sections. First, we'll talk about coding mistakes. Actual "nuts and bolts" coding issues that are pervasive but easily remedied. Secondly, we'll examine some larger theory-based issues that are less tangible but can impact productivity and maintainability.

. . .

Coding Mistakes

Forgetting switch exists



```
let x = 10;

if( x = 2){
    doThing();
}
if( x = 4){
    doOtherThing();
}
if( x = 6){
    doTheSixThing();
}
if(x = 8){
    doEightThing();
}
if(x = 10){
    doTenThing();
}
```

```
let x = 10;

switch(x){
    case 2:
        doThing();
        break;
    case 4:
        doOtherThing();
        break;
    case 6:
        doTheSixThing();
        break;
    case 8:
        doEightThing();
        break;
    case 10:
        doTenThing();
}
```

Let's start our list with a bit of controversy. There's some debate among programmers as to whether or not `switch` is preferable to good old fashioned `if`. In the example above, you can see the `switch` statement actually takes up one more line of code than the same comparison with `if` . I've also seen benchmarks that suggest in some instances `if` can actually be faster than `switch`.

So why do I fall squarely into the pro-switch camp? Because using `switch` statements feels so much more elegant and contained to me than multiple `if` statements. It's inherently more readable — it's a single clear block of code achieving a single goal: to make a comparison and do something with the result.

Multiple chained `if` statements feel open-ended and unstructured — more spaghetti code-ish. Even when accompanied by a closing `else` statement, they seem like unconnected operations running in sequence, because they essentially are.

Not commenting code

Commenting code is like saving for retirement — we all know we should be doing it and yet all too often we don't. We tell ourselves we'll come back to it later and make up for it and then rarely do.

The importance of good code commenting should be a given by now, but every day thousands of functions are released into the wild with absolutely no context or explanation.

What may seem obvious or self-evident to you, may look like magical incantations in ancient Latin to a more novice programmer looking to implement your functionality in their project.

It's such a simple thing to take a few moments and include a quick descriptive comment about how a piece of code works or why it's necessary. Take the time and develop good habits around commenting your code as you write it. Not only does this lessen your technical debt in future, but it can save *you* when you come back to a chunk of code months or even years down the line and need to understand what you were thinking when you wrote it.

Oversized functions

Function bloat happens. It's a natural consequence of the development process as we figure out how to achieve the functionality we want for our application.

Oversized functions can become a real problem however as your application matures. Often they are much less efficient than abstracting into multiple smaller functions which can affect the performance of your application.

Speed concerns aside, they also make maintaining and changing your project far more difficult. Issues like variable scope or application state get harder to manage when you have a single function doing four different things to a piece of data.

. . .

Development Theory Mistakes

Using new frameworks just because

I've written before that developers should fear no framework, but that's not the same as saying you have to use every new coding fad that gets a few Github stars.

New frameworks can be exciting and offer genuinely better ways of doing things, in some cases. It's important to evaluate whether or not that new framework is going to actually benefit you in achieving the goal you have for your project. Often, you'll find the answer is that it doesn't.

By all means, play around and test the waters of new frameworks, but don't feel compelled to architect your newest application around them just because they're hot.

Remember, the best path forward is the one that lets you actually finish your project and ship it. Spending three months trying to shoehorn it into a new framework can be frustrating and wildly unproductive, often leading to failure and lost time.

Not planning

New projects can be exciting. Having a great idea and getting to work on it immediately can build valuable momentum and energy, but, at some point, the scope of the project will begin to get out of control.

Taking some time to whiteboard or notebook out some architecture can be extremely valuable and is a step too often skipped by developers. It's tempting to move ahead boldly trusting ourselves to keep everything in our head, but ultimately is unrealistic and counterproductive.

Code & Quill makes some amazing notebooks for exactly this sort of work (not affiliated in any way, just a fan). Learning to slow down at the right points in development and do some planning can pay massive dividends further down the road in keeping your data structures and application flows straight.

Not collaborating

Very few things will make you a better developer faster than collaborating with other programmers. Collaboration opens you up to new ways of thinking about problems or potential solutions and it can spark major moments of inspiration.

Sometimes another programmer will have a legitimately better way of handling some piece of application development. More often you'll find that combining their ideas with your own will culminate in a vastly superior solution than either of you were capable of independently.

If you use Visual Studio Code (my preferred IDE), it comes built-in with an amazing peer programming solution called Live Share. Live Share allows you and teammates to work simultaneously on a single codebase without having to clone GitHub repos or emailing zips back and forth. It's fast, intuitive, and easy.

· · ·

At the end of the day, every programmer has their own process. For all our differences, however, most of us tend to make some of the same mistakes. Many of them are so easily remedied it's worth listing them.

A few minor tweaks to your approach both at a code level, as well as in how you think about your architecture, can save you time and make you more successful.

Maybe some of these things don't seem like mistakes to you. That's fine, but for many of us making these changes can radically improve our output and the maintainability of our codebase. That gives us all more time to go out and build the next amazing thing!