JAVAWORLD

FEATURE

12 programming mistakes to avoid

The dirty dozen of application development pitfalls — and how to avoid these all-too-common programming blunders

By Peter Wayner
Contributing Editor, InfoWorld
DEC 16, 2019 3:00 AM PST

If you need any better proof that code is art, look no further than how the programmers view their mistakes. Just as the world is filled with wildly divergent opinions about painters, architects, writers, and poets, the realm of programmers can't agree upon much beyond the requirement that the code doesn't crash. Even this is a stretch. Some are fine with failing code as long as it recovers gracefully before the user notices.

The debates are usually born out of experience. When a developer says not to do X, it's probably because some evening, weekend, or even spring vacation was ruined because someone around the office did X and it failed badly. X seemed like a good idea at the time but it was an intellectual trap and now the survivors want to warn the world about it.

[Also on InfoWorld: 10 bad programming habits we secretly love]

The problem often arises when doing the opposite of X, call it Y, has its own failure modes too. Another team of developers dodged the X trap by choosing Y, but then they ran into their own lost weekends of hair pulling and teeth clenching. All of their tears are now boiled down into a bitter liquor that they push upon all of their guests. When you visit, you must chug it and instead of chanting "cheers," "skol," or "Nostrovia," you might say "functional" or "serverless" or "lambda." Choose wisely, young Padawan. The terms change.

Is there a hope for unifying the X's and the Y's? Maybe not on the same development team, but maybe in your mind. There's no reason why you can't learn from the mistakes of both teams. The best path to nirvana is often the middle one. You can borrow the lessons from both X and Y and steer your code away from the issues that caused so much grief.

Below you will find the most common programming pitfalls, each of which is accompanied by its opposing pair, lending further proof that programming may in fact be transforming into an art—one that requires a skilled hand and a creative mind to achieve a happy medium between problematic extremes.

Programming mistake No. 1: Playing it fast and loose

Failing to shore up the basics is the easiest way to undercut your code. Often this means overlooking how arbitrary user behavior will affect your program. Will the input of a zero find its way into a division operation? Will submitted text be the right length? Have date formats been vetted? Is the username verified against the database? Mistakes in the smallest places cause software to fail.

Some developers exploit the error catching features of the code to cover up these failures. They wrap their entire stack with one big catch for all possible exceptions. They dump the error to a log file, return an error code, and let someone else deal with the issue.

[Looking to upgrade your career in tech? This comprehensive online course teaches you how.]

Programming mistake No. 2: Overcommitting to details

On the flip side, overly buttoned-up software can slow to a crawl. Checking a few null pointers may not make much difference, but some software is written to be like an obsessive-compulsive who must check that the doors are locked again and again so that sleep never comes.

Relentless devotion to detail can even lock up software if the obsessive checking requires communicating with a distant website over the network. I have several packages that slow to a crawl if I fire them up on a laptop without a Wi-Fi connection because they're frantically trying to phone home to see if a new version might be available. The Wi-Fi LED flickers, and the software hangs, constantly looking for a hotspot that isn't there.

The challenge is to design the layers of code to check the data when it first appears, but this is much easier said than done. If multiple developers work on a library or even if only one does all of the coding, it's difficult to remember whether and when the pointer was checked.

Programming mistake No. 3: Not simplifying control

Too often, developers invite disaster by not simplifying control over tasks in their code.

Mike Subelsky, one of the co-founders of OtherInBox.com, is a keen advocate of there being one and only one place in the code for each job. If there are two places, odds are someone will change one but not the other. If there are more than two, the odds get even worse that someone will fail to keep them all working in the same way.

"Having worked on one code base for three-plus years, my biggest regret is not making the code more modular," Subelsky says. "I've learned the hard way why the Single Responsibility Principle is so important. I adhere to it strongly in new code, and it's the first thing I attack when refactoring the old code."

[Also on InfoWorld: 10 software development cults to join]

Subelsky, as you may surmise, is a Ruby on Rails programmer. The framework encourages lean code by assuming most of the structure of the software will fall into well-known patterns, a philosophy that Rails programmers often summarize as "convention not configuration." The software assumes that if someone creates an object of type `Name` with two fields `first` and `last`, then it should immediately create a database table called `Name` with two columns, `first` and `last`. The names are specified in only one place, avoiding any problems that might come if someone fails to keep all of the layers of configuration in sync.

Programming mistake No. 4: Delegating too much to frameworks

Sometimes the magic tools lead only to confusion. By abstracting functionality and assuming what we want, frameworks can all too often leave developers at a loss for what's gone wrong in their code.

G. Blake Meike, a programmer based near Seattle, is one of many developers who finds over-reliance on automated tools such as Ruby on Rails a hindrance when it comes to producing clean code.

"Convention, by definition, is something outside the code," Meike says. "Unless you know Ruby on Rails' rules for turning a URL into a method call, for instance, there is no way, at all, that you will ever figure out what actually happens in response to a query."

He finds that reading the code often means keeping a manual close by to decipher what the code is doing behind his back.

"The rules are, while quite reasonable, not entirely trivial. In order to work on a Ruby on Rails app, you just have to know them. As the app grows, it depends on more and more of these almost-trivial bits of external knowledge. Eventually, the sum of all the almost-trivial bits is decidedly not trivial. It's a whole ecosphere of things you have to learn to work on the app and remember while you are debugging it," he says.

To make matters worse, the frameworks can often leave you, and any who come after you, stranded with pretty code that's difficult to understand, revise, or extend.

As Mike Morton, another programmer, explains, "They carry you 90 percent of the way up the mountain in a sedan chair, but that's all. If you want to do the last 10 percent, you'll need to have thought ahead and brought oxygen and pitons."

Programming mistake No. 5: Trusting the client

Many of the worst security bugs appear when developers assume the client device will do the right thing. For example, code written to run in a browser can be rewritten by the browser to execute any arbitrary action. If the developer doesn't double-check all of the data coming back, anything can go wrong.

One of the simplest attacks relies on the fact that some programmers just pass along the client's data to the database, a process that works well until the client decides to send along SQL instead of a valid answer. If a website asks for a user's name and adds the name to a query, the attacker might type in the name `x; DROP TABLE users;`. The database dutifully assumes the name is `x` and then moves on to the next command, deleting the table filled with all of the users.

There are many other ways that clever people can abuse the trust of the server. Web polls are invitations to inject bias. Buffer overruns continue to be one of the simplest ways to corrupt software.

To make matters worse, severe security holes can arise when three or four seemingly benign holes are chained together. One programmer may let the client write a file assuming that the directory permissions will be sufficient to stop any wayward writing. Another may open up the permissions just to fix some random bug. Alone there's no trouble, but together, these coding decisions can hand over arbitrary access to the client.

Programming mistake No. 6: Not trusting the client enough

Sometimes too much security can lead paradoxically to gaping holes. Just a few days ago, I was told that the way to solve a problem with a particular piece of software was just to `chmod 777` the directory and everything inside it. Too much security ended up gumming up the works, leaving developers to loosen strictures just to keep processes running.

Web forms are another battleground where trust can save you in the long run. Not only do bank-level security, long personal data questionnaires, and confirming email addresses discourage people from participating even on gossip-related sites, but having to protect that data once it is culled and stored can be far more trouble than it's worth.

[Also on InfoWorld: 9 lies programmers tell themselves]

Because of this, many Web developers are looking to reduce security as much as possible, not only to make it easy for people to engage with their products but also to save them the trouble of defending more than the minimum amount of data necessary to set up an account.

My book, *Translucent Databases*, describes a number of ways that databases can store less information while providing the same services. In some cases, the solutions will work while storing nothing readable.

Programming mistake No. 7: Relying too heavily on magic boxes

Worried about security? Just add some cryptography. Want your databases to be backed up? Just push the automated replication button. Don't worry. The salesman said, "It just works."

Computer programmers are a lucky lot. After all, computer scientists keep creating wonderful libraries filled with endless options to fix what ails our code. The only problem is that the ease with which we can leverage someone else's work can also hide complex issues that gloss over or, worse, introduce new pitfalls into our code.

Cryptography is a major source of weakness here, says John Viega, co-author of *24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them*. Far too many programmers assume they can link in the encryption library, push a button, and have iron-clad security.

But the reality is that many of these magic algorithms have subtle weaknesses, and avoiding these weaknesses requires learning more than what's in the Quick Start section of the manual. To make matters worse, simply knowing to look beyond the Quick Start section assumes a level of knowledge that goes beyond what's covered in the Quick Start section, which is likely why many programmers are entrusting the security of their code to the Quick Start section in the first place. As the philosophy professors say, "You can't know what you don't know."

Programming mistake No. 8: Reinventing the wheel

Then again, making your own yogurt, slaughtering your own pigs, and writing your own libraries just because you think you know a better way to code can come back to haunt you.

"Grow-your-own cryptography is a welcome sight to attackers," Viega says, noting that even the experts make mistakes when trying to prevent others from finding and exploiting weaknesses in their systems.

So, whom do you trust? Yourself or so-called experts who also make mistakes?

The answer falls in the realm of risk management. Many libraries don't need to be perfect, so grabbing a magic box is more likely to be better than the code you write yourself. The library includes routines written and optimized by a group. They may make mistakes, but the larger process can eliminate many of them.

优化

消除

Programming mistake No. 9: Closing the source

One of the trickiest challenges for any company is determining how much to share with the people who use the software.

tricky狡猾的机智的

John Gilmore, co-founder of one of the earliest open source software companies, Cygnus Solutions, says the decision to not distribute code works against the integrity of that code, being one of the easiest ways to discourage innovation and, more important, uncover and fix bugs.

"A practical result of opening your code is that people you've never heard of will contribute improvements to your software," Gilmore says. "They'll find bugs and attempt to fix them; they'll add features; they'll improve the documentation. Even when their improvement has been amateurly done, a few minutes' reflection by you will often reveal a more harmonious way to accomplish a similar result."

amateur业余爱好者

[Also on InfoWorld: 15 noob mistakes even experienced developers still make]

The advantages run deeper. Often the code itself grows more modular and better structured as others recompile the program and move it to other platforms. Just opening up the code forces you to make the info more accessible, understandable, and thus better. As we make the small tweaks to share the code, they feed the results back into the code base.

n.轻微调整

Programming mistake No. 10: Assuming openness is a cure-all

Your cloud, your way: Why Cloud Verified matters