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

Page 2 of 2

Millions of open source projects have been launched, and only a tiny fraction have ever attracted more than a few people to help maintain, revise, or extend the code. In other words, W.P. Kinsella's "if you build it, they will come" doesn't always produce practical results.

While openness can make it possible for others to pitch in and thus improve your code, the mere fact that it's open won't do much unless there's another incentive for outside contributors to put in the work. Passions among open source proponents can blind some developers to the reality that openness alone doesn't prevent security holes, eliminate crashing, or make a pile of unfinished code inherently useful. People have other things to do, and an open pile of code must compete with hiking, family, bars, and paying jobs.

Opening up a project can also add new overhead for communications and documentation. A closed source project requires solid documentation for the users, but a good open source project also needs extensive documentation of the API and road maps for future development. This extra work pays off for large projects, but it can weigh down smaller ones.

Too often, code that works some of the time is thrown up on GitHub in hopes that the magic elves 侏儒 will stop making shoes and rush to start up the compiler—a decision that can derail 使出轨，偏离轨道 a project's momentum before it truly gets started.

Opening up the project can also strip away financial support and encourage a kind of mob 暴民，暴徒 rule. Many open source companies try to keep some proprietary adj.专卖的，专营的 n.所有权，所有人 feature in their control; this gives them some leverage to get people to pay to support the core development team. The projects that rely more on volunteers than paid programmers often find that volunteers are unpredictable. While wide-open competitiveness 竞争力 and creativity can yield great results, some flee 逃走，消失 back to where structure, hierarchy 层级，等级制度, and authority support methodical 有系统的，有方法的 development.

Table of Contents

SHOW MORE

Programming mistake No. 11: Following too many trendy ideas

Trends often begin with a good plan to fix some gnarly 粗糙的，艰难的 problem. If we can just rewrite our code with a new language, framework, or collection of libraries, we can eliminate the difficulty. The plan usually ends up bearing some fruit 结出果实，奏效. The trouble isn't the cost of rewriting the code—although that is often substantial adj.大量的，实质的 n.本质，重要材料—it's the shift to the new paradigm 范例, which often leaves a mirror image of the failure mode. A yang instead of a yin.

The problems get worse if you need to accomplish anything substantial. Many of the newest ideas and frameworks offer only the important features. Everything else hasn't been written yet. Sure, you'll save some work on the features that are ready, but you'll often write more of the glue 胶水，粘合剂, background, or filler code. Being on the cutting edge 最前沿 is hard work.

Programming mistake No. 12: Ignoring too many trendy ideas

It's easy to dismiss the clever, cute, or ingenious 有独创性的，机灵的 new pile of code as folly 愚蠢，荒唐事, or as a capricious 反复无常的，任性的 vanity n.虚荣心，空虚，浮华 project. Many of the times you'll be right. But occasionally you'll be wrong and then you'll wake up to look at a stack of code that's hopelessly out-of-date.

In the beginning, you can console yourself by focusing on the fact that the code still runs. Even long out of date code can function successfully for awhile. Eventually, though, bits and pieces will start to fail and you won't have easy options. Perhaps some API will be shut down without leaving a forwarding address. Maybe a partner will stop taking XML files and you'll be forced to launch a crash coding project to generate YAML or GraphQL results.

[Keep up with hot topics in software development with InfoWorld's App Dev Report newsletter]

Technical debt isn't a well-defined concept like monetary debt, but the philosophical debates won't matter when it's midnight and the code won't compile because you're stuck using version 2.1 of the compiler while the rest of the world has moved on to versions 3.0, 4.0, or maybe one of those non-numbered quantum leaps of nomenclature like NG. Pay now or pay even more later.

This story, "12 programming mistakes to avoid" was originally published by InfoWorld.

Peter Wayner is contributing editor at InfoWorld and the author of more than 16 books on diverse topics, including open source software, autonomous cars, privacy-enhanced computation, digital transactions, and steganography.

Follow

Page 2 of 2