JOEL ON SOFTWARE

I’m Joel Spolsky, a software developer in New York City. More about me.

APRIL 6, 2000 *by* JOEL SPOLSKY

Things You Should Never Do, Part I

TOP 10, CEO, NEWS

Netscape 6.0 is finally going into its first public beta. There never was a version 5.0. The last major release, version 4.0, was released almost three years ago. Three years is an *awfully* long time in the Internet world. During this time, Netscape sat by, helplessly, as their market share plummeted.

It's a bit smarmy of me to criticize them for waiting so long between releases. They didn't do it *on purpose*, now, did they?

Well, yes. They did. They did it by making the **single worst strategic mistake** that any software company can make:

They decided to rewrite the code from scratch.

Netscape wasn't the first company to make this mistake. Borland made the same mistake when they bought Arago and tried to make it into dBase for Windows, a doomed project that took so long that Microsoft Access ate their lunch, then they made it again in rewriting Quattro Pro from scratch and astonishing people with how few features it had. Microsoft almost made the same mistake, trying to rewrite Word for Windows from scratch in a doomed project called Pyramid which was shut down, thrown away, and swept under the rug. Lucky for Microsoft, they had never stopped working on the old code base, so they had something to ship, making it merely a financial disaster, not a strategic one.

We're programmers. Programmers are, in their hearts, architects, and the first thing they want to do when they get to a site is to bulldoze the place flat and build something grand. We're not excited by incremental renovation: tinkering, improving, planting flower beds

improving, planning new bugs.

There's a subtle reason that programmers always want to throw away the code and start over. The reason is that they think the old code is a mess. And here is the interesting observation: *they are probably wrong.* The reason that they think the old code is a mess is because of a cardinal, fundamental law of programming:

It's harder to read code than to write it.

This is why code reuse is so hard. This is why everybody on your team has a different function they like to use for splitting strings into arrays of strings. They write their own function because it's easier and more fun than figuring out how the old function works.

As a corollary of this axiom, you can ask almost any programmer today about the code they are working on. "It's a big hairy mess," they will tell you. "I'd like nothing better than to throw it out and start over."

Why is it a mess?

"Well," they say, "look at this function. It is two pages long! None of this stuff belongs in there! I don't know what half of these API calls are for."

Before Borland's new spreadsheet for Windows shipped, Philippe Kahn, the colorful founder of Borland, was quoted a lot in the press bragging about how Quattro Pro would be much better than Microsoft Excel, because it was written from scratch. All new source code! As if source code *rusted*.

The idea that new code is better than old is patently absurd. Old code has been *used*. It has been *tested*. *Lots* of bugs have been found, and they've been *fixed*. There's nothing wrong with it. It doesn't acquire bugs just by sitting around on your hard drive. Au contraire, baby! Is software supposed to be like an old Dodge Dart, that rusts just sitting in the garage? Is software like a teddy bear that's kind of gross if it's not made out of *all new material*?

Back to that two page function. Yes, I know, it's just a simple function to display a window, but it has grown little hairs and stuff on it and nobody knows why.

Well, I'll tell you why: those are bug fixes. One of them fixes that bug that Nancy had when she tried to install the thing on a computer that didn't have Internet Explorer. Another one fixes that bug that occurs in low memory conditions. Another one fixes that bug that occurred when the file is on a floppy disk and the user yanks out the disk in the middle. That LoadLibrary call is ugly but it makes the code work on old versions of Windows 95.

Each of these bugs took weeks of real-world usage before they were found. The programmer might have spent a couple of days reproducing the bug in the lab and fixing it. If it's like a lot of bugs, the fix might be one line of code, or it might even be a couple of characters, but a lot of work and time went into those two characters.

When you throw away code and start from scratch, you are throwing away all that knowledge. All those collected bug fixes. Years of programming work.

You are throwing away your market leadership. You are giving a gift of two or three years to your competitors, and believe me, that is a *long* time in software years.

You are putting yourself in an extremely dangerous position where you will be shipping an old version of the code for several years, completely unable to make any strategic changes or react to new features that the market demands, because you don't have shippable code. You might as well just close for business for the duration.

You are wasting an outlandish amount of money writing code that already exists.

Is there an alternative? The consensus seems to be that the old Netscape code base was *really* bad. Well, it might have been bad, but, you know what? It worked pretty darn well on an awful lot of real world computer systems.

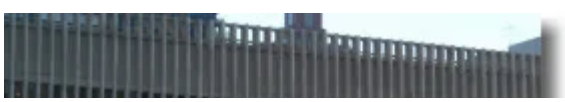
When programmers say that their code is a holy mess (as they always do), there are three kinds of things that are wrong with it.

First, there are architectural problems. The code is not factored correctly. The networking code is popping up its own dialog boxes from the middle of nowhere; this should have been handled in the UI code. These problems can be solved, one at a time, by carefully moving code, refactoring, changing interfaces. They can be done by one programmer working carefully and checking in his changes all at once, so that nobody else is disrupted. Even fairly major architectural changes can be done without *throwing away the code*. On the Juno project we spent several months rearchitecting at one point: just moving things around, cleaning them up, creating base classes that made sense, and creating sharp interfaces between the modules. But we did it carefully, with our existing code base, and we didn't introduce new bugs or throw away working code.

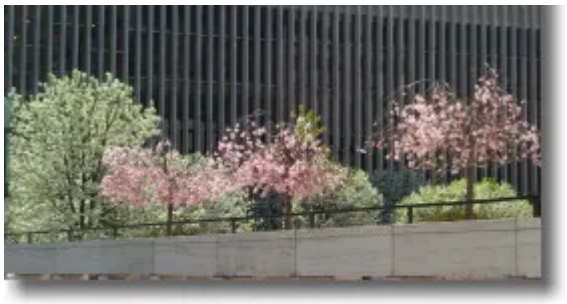
A second reason programmers think that their code is a mess is that it is inefficient. The rendering code in Netscape was rumored to be slow. But this only affects a small part of the project, which you can optimize or even rewrite. You don't have to rewrite the whole thing. When optimizing for speed, 1% of the work gets you 99% of the bang.

Third, the code may be doggone ugly. One project I worked on actually had a data type called a FuckedString. Another project had started out using the convention of starting member variables with an underscore, but later switched to the more standard "m_". So half the functions started with "_" and half with "m_", which looked ugly. Frankly, this is the kind of thing you solve in five minutes with a macro in Emacs, not by starting from scratch.

It's important to remember that when you start from scratch there is **absolutely no reason** to believe that you are going to do a better job than you did the first time. First of all, you probably don't even have the same programming team that worked on version one, so you don't actually have "more experience". You're just going to make most of the old mistakes again, and introduce some new problems that weren't in the original version.



The old mantra *build one to throw away* is dangerous when applied to large scale



dangerous when applied to large scale commercial applications. If you are writing code experimentally, you may want to rip up the function you wrote last week when you think of a better algorithm. That's fine. You may want to refactor a class to make it easier to use. That's fine, too. But throwing away the whole program is a dangerous folly, and if Netscape actually had some adult supervision with software industry experience, they might not have shot themselves in the foot so badly.

SUBSCRIBE!

You're reading Joel on Software, stuffed with years and years of completely raving mad articles about software development, managing software teams, designing user interfaces, running successful software companies, and rubber duckies.

If you want to know when I publish something new, I recommend getting an RSS reader like NewsBlur and subscribing to my RSS feed.

ABOUT THE AUTHOR.

In 2000 I co-founded Fog Creek Software, where we created lots of cool things like the FogBugz bug tracker, Trello, and Glitch. I also worked with Jeff Atwood to create Stack Overflow and served as CEO of Stack Overflow from 2010-2019. Today I serve as the chairman of the board for Stack Overflow, Glitch, and HASH.

← PREVIOUS POST
2000/04/06

NEXT POST →

2000/04/10