Programming peaked

by Samir Talwar
Tuesday, 25 November 2025 at 09:00 CET

I remember my first job vividly.

It helps, of course, that I still consider many of the people I worked with friends, and I know some of them even still read this blog. (Hi!)

I also think it might have been the beginning of the end. At least, as programming is concerned.

Programming in 2025

It's 2025. We write JavaScript with types now. It runs not just in a browser, but on Linux. It has a dependency manager, and in true JavaScript style, there's a central repository which anyone can push anything to. Nowadays it's mostly used to inject Bitcoin miners or ransomware onto unsuspecting servers, but you might find a useful utility to pad a string if you need it.

We don't *write* the JavaScript, of course. We ask an autocorrect machine to make it up, and complain at it until it generates something plausible enough. Often, it does, and it only installs a malicious package 20% of the time. 30%, tops.

Fortunately, when we do need to make manual changes, our editor has our back. The most popular is "VS Code", which needs only a few gigabytes of RAM to render the text. It ships a web browser, because everything ships a web browser. It can also perform fancy refactoring techniques such as renaming a variable (in a single file).

In order to test our application, we build it regularly. On a modern computer, with approximately 16 cores, each running at 3 GHz, TypeScript only takes a few seconds to compile and run.

Once our work is done, we create a "pull request". This is a way of emulating open-source development inside a single company, which as we know, is the only way to work. Typically, this means that the code is downloaded and built on another computer, and then several hours later, a colleague will come along and ask to change a few words. Once we change these words, the computer builds everything again, and then the next day, the same colleague will allow the code to be merged into the mainline.

Once it's merged, we take that JavaScript and package it into a "container", which is a fancy word for "kitchen sink". The packaging process takes 20 minutes, because for reasons no one understands, we download the entirety of the Debian package repository every time we do it.

We also download all the JavaScript dependencies again, of course.

Then, we take that container, and attempt to push it to a "registry". Sometimes it works, sometimes a security scanner complains because we have an insecure version of Perl in the container. We don't need Perl in the container, but it's there, and no one knows how to get rid of it. So we turn off the scanner.

Next, we write a lot of YAML to tell something called "k8s" how to run the container. ("K8s" used to stand for something but ever since the wars of 2019, we don't have enough letters to spell it out, and the true meaning has been lost to time.) Unlike the JavaScript, the YAML is not type-checked, and in fact *cannot* be, because we use a string templating language to stick bits of YAML inside other YAML. We all agree that this is the best kind of programming.

We run our k8s cluster in the "Cloud". It's a bunch of services that run on Linux, but we don't run Linux ourselves, we run it on VMs that we rent by the hour for approximately the same cost as buying a computer outright every month. We do this because no one knows how to plug a computer in any more.

We send the YAML to the cluster, which eats it up, digests it, and then does something random. Hopefully, it will run a program. It might also reject it, run a different program, or shut down something else. As there's no way of testing this except doing it, there's no way of telling.

We therefore have an entirely separate cluster that we use to verify that our sacrifice to the YAML gods is holy.

There was some programming in here somewhere, but I don't really remember where.

Fifteen years ago

Turns out, I'm old now. But not so old that I've forgotten how things have changed.

I started my first *real*, full-time job in 2010.

We wrote Java, which is a general-purpose, highly-verbose programming language. It's mostly type-safe (except `null`, which we did our best to avoid), and achieves this by making you type out everything so many times and verifying that you didn't make any mistakes.

Like JavaScript, it's fast to compile, taking only a few seconds as well. Of course, this was on my single-core, 2 GHz computer.

The editor was cool too. We used Eclipse, which was a bit like VS Code. It could also perform refactoring techniques, though it was a bit smarter: it could extract and inline functions, classes, interfaces, etc. It also compiled the code as you typed, so often, you could compile the application and run the tests on every keystroke.

We wrote tests, of course. Usually first, and ran them constantly. We did this using a strange method called "talking to each other", in which we'd discuss the functionality at hand, and write it down in code so it could be verified. We'd run these tests constantly; it'd usually take around a minute to run all 10,000 unit tests, but you could be a bit more selective if you were in a hurry.

Funnily enough, everything ran at about the same speed as it does now. Strange, that.

Of course, we pushed code directly to trunk. We were pairing, after all. We'd then start work on the next thing, perhaps after a coffee break. Typically, we'd let a tester know so they could check out the changes and take a look.

Now, we didn't have containers, so we made these things called "JARs", which is a big zip file of the application and all its dependencies, pulled from Maven Central. (Maven Central is a bit like NPM but for Java, and also actually vetted submitters and their domain names before allowing them to publish.) This produces a different kind of dependency hell to containers, and I would not wish it upon anybody, but it did, at least, kind of work.

We'd run that inside a Java server, such as Apache Tomcat, on a Linux VM, on a computer. Deployment was done through "Puppet"; nowadays you'd use Ansible, but whatever.

The computer was inside a datacentre.

The datacentre was down the road. If the computer broke, we could go and turn it off and on again.

And here's the funny thing: it never broke, because the person who built it did it well, because it wasn't taxed within an inch of its life, and because we were keeping an eye on it.

I miss those days

I think back to those days, and remember writing Java, pairing with my old colleagues.

We were good at it, but the tools also had our backs. Eclipse was dependable; it did so much amazing stuff that I haven't seen since, Maven is a thoughtful approach to dependency management that solved several problems (such as namespacing, and trust) in elegant ways, and no one has even tried to copy them. And I really, really miss deploying to a real computer, which I own, and can touch.

It astonishes me every time I realise that builds and deployment have become *slower* in the last 15 years, not faster. I was able to go from code to shipping in less than a minute back then, and now, it takes hours, or sometimes *days* to get it merged and deployed. Even pushing an application to k8s is an ordeal. Especially if everything has to go through a staging environment first.

I don't really understand why we've stopped hiring testers, but I wish we could get back to it. I like having people around who are good at breaking things. It makes me feel safe when they're on my side.

Now, some things are better. I'd much rather use Git than Subversion. I actually quite like containers, when they're built well (i.e. not with a Dockerfile), and deployed sensibly. And I think the Cloud is useful, up to a point, especially when starting out.

But this world… I do not like it.

Where did it all go wrong?

I have been trying to figure out what happened, and I think I can point to the catalyst.

NPM happened.

Now, this doesn't mean it's NPM's fault, it just means that it made it possible.

You see, with NPM, node.js (and therefore JavaScript) became a serious programming language, which is both a blessing and a curse.

We also saw the rise of React, which is possibly the greatest tragedy to ever befall front-end programming (and I say this as a recovering React fan).

And then, because we could create applications in the browser, we got Electron. Now everything's a browser!

I love writing JavaScript, and I'm glad I can run it on the server. But this doesn't mean I think it's a good idea to use it for everything.

I would really like to reset, and go back to using a sensible tool for the job at hand.

Just a little.
