# Software disenchantment

*Translations: French Italian Portuguese Russian Spanish*

I've been programming for 15 years now. Recently our industry's lack of care for efficiency, simplicity, and excellence started really getting to me, to the point of me getting depressed by my own career and the IT in general.

Modern cars work, let's say for the sake of argument, at 98% of what's physically possible with the current engine design. Modern buildings use just enough material to fulfill their function and stay safe under the given conditions. All planes converged to the optimal size/form/load and basically look the same.

Only in software, it's fine if a program runs at 1% or even 0.01% of the possible performance. Everybody just seems to be ok with it. People are often even proud about how much inefficient it is, as in "why should we worry, computers are fast enough":

> *@tveastman: I have a Python program I run every day, it takes 1.5 seconds. I spent six hours re-writing it in rust, now it takes 0.06 seconds. That efficiency improvement means I'll make my time back in 41 years, 24 days :-)*

You've probably heard this mantra: "programmer time is more expensive than computer time". What it means basically is that we're wasting computers at an unprecedented scale. Would you buy a car if it eats 100 liters per 100 kilometers? How about 1000 liters? With computers, we do that all the time.

## Everything is unbearably slow

Look around: our portable computers are thousands of times more powerful than the ones that brought man to the moon. Yet every other webpage struggles to maintain a smooth 60fps scroll on the latest top-of-the-line MacBook Pro. I can comfortably play games, watch 4K videos but not scroll web pages? How is it ok?

Google Inbox, a web app written by Google, running in Chrome browser also by Google, takes 13 seconds to open moderately-sized emails:



It also animates empty white boxes instead of showing their content because it's the only way anything can be animated on a webpage with decent performance. No, decent doesn't mean 60fps, it's rather "as fast as this web page could possibly go". I'm dying to see web community answer when 120Hz displays become mainstream. Shit barely hits 60Hz already.

Windows 10 takes 30 minutes to update. What could it possibly be doing for that long? That much time is enough to fully format my SSD drive, download a fresh build and install it like 5 times in a row.

> *Pavel Fatin: Typing in editor is a relatively simple process, so even 286 PCs were able to provide a rather fluid typing experience.*

Modern text editors have higher latency than 42-year-old Emacs. Text editors! What can be simpler? On each keystroke, all you have to do is update tiny rectangular region and modern text editors can't do that in 16ms. It's a lot of time. A LOT. A 3D game can fill the whole screen with hundreds of thousands (!!!) of polygons in the same 16ms and also process input, recalculate the world and dynamically load/unload resources. How come?

As a general trend, we're not getting faster software with more features. We're getting faster hardware that runs slower software with the same features. Everything works way below the possible speed. Ever wonder why your phone needs 30 to 60 seconds to boot? Why can't it boot, say, in one second? There are no physical limitations to that. I would love to see that. I would love to see limits reached and explored, utilizing every last bit of performance we can get for something meaningful in a meaningful way.

## Everything is HUUUUGE

And then there's bloat. Web apps could open up to 10× faster if you just simply block all ads. Google begs everyone to stop shooting themselves in their feet with AMP initiative—a technology solution to a problem that doesn't need any technology, just a little bit of common sense. If you remove bloat, the web becomes crazy fast. How smart do you have to be to understand that?

Android system with no apps takes almost 6 Gb. Just think for a second how obscenely HUGE that number is. What's in there, HD movies? I guess it's basically code: kernel, drivers. Some string and resources too, sure, but those can't be big. So, how many drivers do

you need for a phone?

Windows 95 was 30Mb. Today we have web pages heavier than that! Windows 10 is 4Gb, which is 133 times as big. But is it 133 times as superior? I mean, functionally they are basically the same. Yes, we have Cortana, but I doubt it takes 3970 Mb. But whatever Windows 10 is, is Android really 150% of that?

Google keyboard app routinely eats 150 Mb. Is an app that draws 30 keys on a screen really five times more complex than the whole Windows 95? Google app, which is basically just a package for Google Web Search, is 350 Mb! Google Play Services, which I do not use (I don't buy books, music or videos there)—300 Mb that just sit there and which I'm unable to delete.

All that leaves me around 1 Gb for my photos after I install all the essential (social, chats, maps, taxi, banks etc) apps. And that's with no games and no music at all! Remember times when an OS, apps and all your data fit on a floppy?

Your desktop todo app is probably written in Electron and thus has userland driver for Xbox 360 controller in it, can render 3d graphics and play audio and take photos with your web camera.

A simple text chat is notorious for its load speed and memory consumption. Yes, you really have to count Slack in as a resource-heavy application. I mean, chatroom and barebones text editor, those are supposed to be two of the less demanding apps in the whole world. Welcome to 2018.

At least it works, you might say. Well, bigger doesn't imply better. Bigger means someone has lost control. Bigger means we don't know what's going on. Bigger means complexity tax, performance tax, reliability tax. This is not the norm and should not become the norm. Overweight apps should mean a red flag. They should mean run away scared.

## Everything rots

16Gb Android phone was perfectly fine 3 years ago. Today with Android 8.1 it's barely usable because each app has become at least twice as big *for no apparent reason*. There are no additional functions. They are not faster or more optimized. They don't look different. They just…grow?

iPhone 4s was released with iOS 5, but can barely run iOS 9. And it's not because iOS 9 is that much superior—it's basically the same. But their new hardware is faster, so they made software slower. Don't worry—you got exciting new capabilities like…running the same apps

with the same speed! I dunno.

iOS 11 dropped support for 32-bit apps. That means if the developer isn't around at the time of iOS 11 release or isn't willing to go back and update a once-perfectly-fine app, chances are you won't be seeing their app ever again.

> *@jckarter: A DOS program can be made to run unmodified on pretty much any computer made since the 80s. A JavaScript app might break with tomorrow's Chrome update*

Web pages working today would not be compatible with any browser in 10 years time (probably sooner).

"It takes all the running you can do, to keep in the same place". But what's the point? I might enjoy occasionally buying a new phone and new MacBook as much as the next guy, but to do so just to be able to run all the same apps which just became slower?

I think we can and should do better than that. Everyone is busy building stuff for right now, today, rarely for tomorrow. But it would be nice to also have stuff that lasts a little longer than that.

## Worse is better

Nobody understands anything at this point. Neither they want to. We just throw barely baked shit out there, hope for the best and call it "startup wisdom".

Web pages ask you to refresh if anything goes wrong. Who has time to figure out what happened?

Any web app produces a constant stream of "random" JS errors in the wild, even on compatible browsers.

The whole webpage/SQL database architecture is built on a premise (hope, even) that nobody will touch your data while you look at the rendered webpage.

Most collaborative implementations are "best effort" and have many common-life scenarios in which they lose data. Ever seen this dialogue "which version to keep?" I mean, bar today is so low that your users would be happy to at least have a window like that.

And no, in my world app that says “I’m gonna destroy some of your work, but you get to choose which one” is not okay.

Linux kills random processes *by design*. And yet it’s the most popular server-side OS.

Every device I own fails regularly one way or another. My Dell monitor needs a hard reboot from time to time because there’s software in it. Airdrop? You’re lucky if it’ll detect your device, otherwise, what do I do? Bluetooth? Spec is so complex that devices won’t talk to each other and periodic resets are the best way to go.

And I’m not even touching Internet of Things. It’s so far beyond the laughing point I’m not even sure what to add.

I want to take pride in my work. I want to deliver working, stable things. To do that, we need to understand what we are building, in

and out, and that's impossible to do in bloated, over-engineered systems.

## Programming is the same mess

It just seems that nobody is interested in building quality, fast, efficient, lasting, foundational stuff anymore. Even when efficient solutions have been known for ages, we still struggle with the same problems: package management, build systems, compilers, language design, IDEs.

Build systems are inherently unreliable and periodically require full clean, even though all info for invalidation is there. Nothing stops us from making build process reliable, predictable and 100% reproducible. Just nobody thinks its important. NPM has stayed in "sometimes works" state for years.

> *@przemyslawdabek: It seems to me that* `rm -rf node_modules` *is indispensable part of workflow when developing Node.js/JavaScript projects.*

And build times? Nobody thinks compiler that works minutes or even hours is a problem. What happened to "programmer's time is more important"? Almost all compilers, pre- and post-processors add significant, sometimes disastrous time tax to your build without providing proportionally substantial benefits.

You would expect programmers to make mostly rational decisions, yet sometimes they do the exact opposite of that. E.g. choosing Hadoop even when it's slower than running the same task on a single desktop.

Machine learning and "AI" moved software to guessing in the times when most computers are not even reliable enough in the first place.

> *@rakhim: When an app or a service is described as "AI-powered" or*

> *“ML-based”, I read it as “unreliable, unpredictable, and impossible to reason about behavior”. I try to avoid “AI” because I want computers to be the opposite: reliable, predictable, reasonable.*

We put virtual machines inside Linux, and then we put Docker inside virtual machines, simply because nobody was able to clean up the mess that most programs, languages and their environment produce. We cover shit with blankets just not to deal with it. “Single binary” is still a HUGE selling point for Go, for example. No mess == success.

And dependencies? People easily add overengineered “full package solutions” to solve the simplest problems without considering their costs. And those dependencies bring other dependencies. You end up with a tree that is something in between of horror story (OMG so big and full of conflicts) and comedy (there’s no reason we include these, yet here they are):

Programs can't work for years without reboots anymore. Sometimes even days are too much to ask. Random stuff happens and nobody knows why.

What's worse, nobody has time to stop and figure out what happened. Why bother if you can always buy your way out of it. Spin another AWS instance. Restart process. Drop and restore the whole database. Write a watchdog that will restart your broken app every 20 minutes. Include same resources multiple times, zip and ship. Move fast, don't fix.

That is not engineering. That's just lazy programming. Engineering is understanding performance, structure, limits of what you build, deeply. Combining poorly written stuff with more poorly written stuff goes strictly against that. To progress, we need to understand what and why are we doing.

## We're stuck with it

So everything is just a pile of barely working code added on top of previously written barely working code. It keeps growing in size and complexity, diminishing any chance for a change.

To have a healthy ecosystem you *need* to go back and revisit. You *need* to occasionally throw stuff away and replace it with better stuff.

But who has time for that? We haven't seen new OS kernels in what, 25 years? It's just too complex to simply rewrite by now. Browsers are so full of edge cases and historical precedents by now that nobody dares to write layout engine from scratch.

Today's definition of progress is either throw more fuel into the fire:

> *@sahrizv: 2014 - We must adopt #microservices to solve all problems with monoliths.*
> *2016 - We must adopt #docker to solve all problems with microservices.*
> *2018 - We must adopt #kubernetes to solve all problems with docker*

or reinventing the wheel:

> *@dr_c0d3: 2000: Write 100s of lines of XML to "declaratively" configure your servlets and EJBs.*
> *2018: Write 100s of lines of YAML to "declaratively" configure your microservices.*
> *At least XML had schemas...*

We're stuck with what we have, and nobody will ever save us.

## Business won't care

Neither will users. They are only learned to expect what we can

provide. We (engineers) say every Android app takes 350 Mb? Ok, they'll live with that. We say we can't give them smooth scrolling? Ok, they'll live with a phone that stutter. We say "if it doesn't work, reboot"? They'll reboot. After all, they have no choice.

There's no competition either. Everybody is building the same slow, bloated, unreliable products. Occasional jump forward in quality does bring competitive advantage (iPhone/iOS vs other smartphones, Chrome vs other browsers) and forces everybody to regroup, but not for long.

So it's our mission as engineers to show the world what's possible with today's computers in terms of performance, reliability, quality, usability. If we care, people will learn. And there's nobody but us to show them that it's very much possible. If only we care.

## It's not all bad

There are some bright spots indicating that improving over state-of-the-art is not impossible.

Work Martin Thompson has being doing (LMAX Disruptor, SBE, Aeron) is impressive, refreshingly simple and efficient.

Xi editor by Raph Levien seems to be built with the right principles in mind.

Jonathan Blow has a language he alone develops for his game that can compile 500k lines per second on his laptop. That's cold compile, no intermediate caching, no incremental builds.

You don't have to be a genius to write fast programs. There's no magic trick. The only thing required is not building on top of a huge pile of crap that modern toolchain is.

## Better world manifesto

I want to see progress. I want change. I want state-of-the-art in software engineering to improve, not just stand still. I don't want to reinvent the same stuff over and over, less performant and more bloated each time. I want something to believe in, a worthy end goal, a future better than what we have today, and I want a community of engineers who share that vision.

What we have today is not progress. We barely meet business goals with poor tools applied over the top. We're stuck in local optima and nobody wants to move out. It's not even a good place, it's bloated and inefficient. We just somehow got used to it.

So I want to call it out: where we are today is bullshit. As engineers,

we can, and should, and will do better. We can have better tools, we can build better apps, faster, more predictable, more reliable, using fewer resources (orders of magnitude fewer!). We need to understand deeply what are we doing and why. We need to deliver: reliably, predictably, with topmost quality. We can—and should–take pride in our work. Not just “given what we had…”—no buts!

I hope I’m not alone at this. I hope there are people out there who want to do the same. I’d appreciate if we at least start talking about how absurdly bad our current situation in the software industry is. And then we maybe figure out how to get out.

September 17, 2018 · Discuss on HackerNews Reddit

Hi!

I’m Nikita. Here I write about programming and UI design Subscribe

I also create open-source stuff: Fira Code, AnyBar, DataScript and Rum. If you like what I do and want to get early access to my articles (along with other benefits), you should support me on Patreon.