Use as few tools as possible

W. Adam Koszek

Koszek ORG

wojciech@koszek.com

2022-10-05T17:47:00Z

Choosing a technology stack is an essential part of building and engineering your software product. This is the matter you'll be stuck in forever. Unless you build a 2-line program it's important to make good decisions at this stage.

I see a lot of "technical fashion" in the computer world. People come up with new and different solutions to the same problems, depending on things like programming language or a method itself. And then they start encompassing their new creations into new products. Thus we see new wave of computer languages, tools and transpilers which essentially do the same, but in a different way.

I think these developments are good, since they show creativity and curiosity. It can be tempting to start using them in your software immediately. One of the guidelines might be: if the tool is there and is easy to install, just do it. But here's my advice.

Don't.

Use as little tools as possible; as long as you can. Counterintuitive? Not quite.

First of all, from my experience, adding anything to your toolset costs you something. Most often time. It's because things are easy to install when you work on your powerful laptop, when everything is setup and works. When you try to replicate the flow on your CI system later, it starts to lead to problems. Setup is harder and if you use something like Travis CI, it takes long time to debug and run.

Second is that the more tools you add, the less likely you'll find other people comfortable with the same tools. Basically remember that you may end up doing most of the work, if the technology you picked is unknown in your team, or if people aren't willing to touch it.

The more you work with something, the more familiar you become. My example: if I started using CMake, Rake, Ant and other build tools together for each of the projects I've done, I'd never really learn make well. So I use "make" whenever I can, since it works on all operating systems. FreeBSD/OSX and Windows all have it, which is nice, since I don't need to fiddle with their package managers to get basic functionality (even if on Windows is nmake I don't care – the syntax of the file is the same.) This is this type of decision you need to make and just stick to it for some time. And yes, you'll have people tell you make doesn't work for many scenarios, but you may never hit these problems, since your projects will be 100 files at most, and make handles that well.

(And yes, I know that if you build a modern project it's possible you'll end up with all these tools at once, but for your smaller projects, just stick to one tool)

Similar decisions I made:

I'm not using Rake for Ruby projects, since most often I need to execute some shell commands, and Rake is not shell. So most often I ended up debugging both Ruby and shell that way. Nowadays I just use `make`. Once I figure working shell command, I literally paste it to `make` it work. I also get free parallel execution with `make -j`, while in Rake I'd have to write it myself.

Another example are transpilers. I try to write plain HTML if I have to do a website, since most of snippets for e.g.: Twitter Bootstrap use it. If I used HAML it would be 10% better, but I'd spend 30% more time retyping these snippets by hand, which defeats the purpose. If you shuffle website code everyday, HAML may make sense, but for most of us is unnecessary.

Software is a moldable, and there are no hard guidelines, but here's my take on it:

- only wire new tool to your toolbox when the existing tool doesn't work. For example, if you start a team project and most of the people use Python but will have to write some front-end software, picking Coffeescript makes sense
- if you're in the team, unless the tool is 50%-100% better, use whatever other people would be willing to use
- if you want to use new fancy tool anyway, do it for a new project, starting from scratch.

Would be interested to hear your opinion on this.