Dan McKinley

Math, Programming, and Minority Reports

Tweet Follow @mcfunley

Choose Boring Technology

March 30th, 2015

Probably the single best thing to happen to me in my career was having had Kellan placed in charge of me. I stuck around long enough to see Kellan's technical decisionmaking start to bear fruit. I learned a great deal *from* this, but I also learned a great deal as a *result* of this. I would not have been free to become the engineer that wrote Data Driven Products Now! if Kellan had not been there to so thoroughly stick the landing on technology choices.

Being inspirational as always.

In the year since leaving Etsy, I've resurrected my ability to care about technology. And my thoughts have crystallized to the point where I can write them down coherently. What follows is a distillation of the Kellan gestalt, which will hopefully serve to horrify him only slightly.

Embrace Boredom.

Let's say every company gets about three innovation tokens. You can spend these however you want, but the supply is fixed for a long while. You might get a few more *after* you achieve a certain level of stability and maturity, but the general tendency is to overestimate the contents of your wallet. Clearly this model is approximate, but I think it helps.

If you choose to write your website in NodeJS, you just spent one of your innovation tokens. If you choose to use MongoDB, you just spent one of your innovation tokens. If you choose to use service discovery tech that's existed for a year or less, you just spent one of your innovation tokens. If you choose to write your own database, oh god, you're in trouble.

Any of those choices might be sensible if you're a javascript consultancy, or a database company. But you're probably not. You're probably working for a company that is at least ostensibly rethinking global commerce or reinventing payments on the web or pursuing some other suitably epic mission. In that context, devoting any of your limited attention to innovating ssh is an excellent way to fail. Or at best, delay success [1].

What counts as boring? That's a little tricky. "Boring" should not be conflated with "bad." There is technology out there that is both boring and bad [2]. You should not use any of that. But there are many choices of technology that are boring and good, or at least good enough. MySQL is boring. Postgres is boring. PHP is boring. Python is boring. Memcached is boring. Squid is boring. Cron is boring.

The nice thing about boringness (so constrained) is that the capabilities of these things are well understood. But more importantly, their failure modes are well understood. Anyone who knows me well will understand that it's only with a overwhelming sense of malaise that I now invoke the spectre of Don Rumsfeld, but I must.

To be clear, fuck this guy.

When choosing technology, you have both known unknowns and unknown unknowns [3].

- A known unknown is something like: *we don't know what happens when this database hits 100% CPU.*
- An unknown unknown is something like: *geez it didn't even occur to us that writing stats would cause GC pauses.*

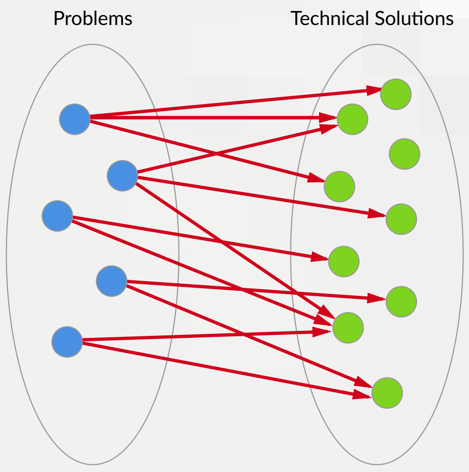
Both sets are typically non-empty, even for tech that's existed for decades. But for shiny new technology the magnitude of unknown unknowns is significantly larger, and this is important.

Optimize Globally.

I unapologetically think a bias in favor of boring technology is a good thing, but it's not the only factor that needs to be considered. Technology choices don't happen in isolation. They have a scope that touches your entire team, organization, and the system that emerges from the sum total of your choices.

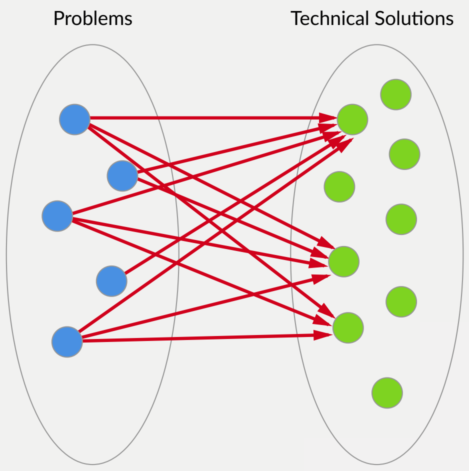
Adding technology to your company comes with a cost. As an abstract statement this is obvious: if we're already using Ruby, adding Python to the mix doesn't feel sensible because the resulting complexity would outweigh Python's marginal utility. But somehow when we're talking about Python and Scala or MySQL and Redis people lose their minds, discard all constraints, and start raving about using the best tool for the job.

Your function in a nutshell is to map business problems onto a solution space that involves choices of software. If the choices of software were truly without baggage, you could indeed pick a whole mess of locally-the-best tools for your assortment of problems.



The way you might choose technology in a world where choices are cheap: "pick the right tool for the job."

But of course, the baggage exists. We call the baggage "operations" and to a lesser extent "cognitive overhead." You have to monitor the thing. You have to figure out unit tests. You need to know the first thing about it to hack on it. You need an init script. I could go on for days here, and all of this adds up fast.



The way you choose technology in the world where operations are a serious concern (i.e., "reality").

The problem with "best tool for the job" thinking is that it takes a myopic view of the words "best" and "job." Your job is keeping the company in business, god damn it. And the "best" tool is the one that occupies the "least worst" position for as many of your problems as possible.

It is basically always the case that the long-term costs of keeping a system working reliably vastly exceed any inconveniences you encounter while building it. Mature and productive developers understand this.

Choose New Technology, Sometimes.

Taking this reasoning to its *reductio ad absurdum* would mean picking Java, and then trying to implement a website without using anything else at all. And that would be crazy. You need some means to add things to your toolbox.

An important first step is to acknowledge that this is a process, and a conversation. New tech eventually has company-wide effects, so adding tech is a decision that requires company-wide visibility. Your organizational specifics may force the conversation, or they may facilitate developers adding new databases and queues without talking to anyone. One way or another you have to set cultural expectations that **this is something we all talk about**.

One of the most worthwhile exercises I recommend here is to **consider how you would solve your immediate problem without adding anything new**. First, posing this question should detect the situation where the "problem" is that someone really wants to use the technology. If that is the case, you should immediately abort.

I just watched a webinar about this graph database, we should try it out.

It can be amazing how far a small set of technology choices can go. The answer to this question in practice is almost never "we can't do it," it's usually just somewhere on the spectrum of "well, we could do it, but it would be too hard" [4]. If you think you can't accomplish your goals with what you've got now, you are probably just not thinking creatively enough.

It's helpful to **write down exactly what it is about the current stack that makes solving the problem prohibitively expensive and difficult.** This is related to the previous exercise, but it's subtly different.

New technology choices might be purely additive (for example: "we don't have caching yet, so let's add memcached"). But they might also overlap or replace things you are already using. If that's the case, you should **set clear expectations about migrating old functionality to the new system.** The policy should typically be "we're committed to migrating," with a proposed timeline. The intention of this step is to keep wreckage at manageable levels, and to avoid proliferating locally-optimal solutions.

This process is not daunting, and it's not much of a hassle. It's a handful of questions to fill out as homework, followed by a meeting to talk about it. I think that if a new technology (or a new service to be created on your infrastructure) can pass through this gauntlet unscathed, adding it is fine.

Just Ship.

Polyglot programming is sold with the promise that letting developers choose their own tools with complete freedom will make them more effective at solving problems. This is a naive definition of the problems at best, and motivated reasoning at worst. The weight of day-to-day operational toil this creates crushes you to death.

Mindful choice of technology gives engineering minds real freedom: the freedom to contemplate bigger questions. Technology for its own sake is snake oil.

Update, July 27th 2015: I wrote a talk based on this article. You can see it here.

---

1. Etsy in its early years suffered from this pretty badly. We hired a bunch of Python programmers and decided that we needed to find something for them to do in Python, and the only thing that came to mind was creating a pointless middle layer that required years of effort to amputate. Meanwhile, the 90th percentile search latency was about two minutes. Etsy didn't fail, but it went several years without shipping anything at all. So it took longer to succeed than it needed to.
2. We often casually refer to the boring/bad intersection of doom as "enterprise software," but that terminology may be imprecise.
3. In saying this Rumsfeld was either intentionally or unintentionally alluding to the Socratic Paradox. Socrates was by all accounts a thoughtful individual in a number of ways that Rumsfeld is not.
4. A good example of this from my experience is Etsy's activity feeds. When we built this feature, we were working pretty hard to consolidate most of Etsy onto PHP, MySQL, Memcached, and Gearman (a PHP job server). It was much more complicated to implement the feature on that stack than it might have been with something like Redis (or maybe not). But it is absolutely possible to build activity feeds on that stack.

   An amazing thing happened with that project: our attention turned elsewhere for several years. During that time, activity feeds scaled up 20x while *nobody was watching it at all.* We made no changes whatsoever specifically targeted at activity feeds, but everything worked out fine as usage exploded because we were using a shared platform. This is the long-term benefit of restraint in technology choices in a nutshell.

   This isn't an absolutist position--while activity feeds stored in memcached was judged to be practical, implementing full text search with faceting in raw PHP wasn't. So Etsy used Solr.

Back home

Tweet Follow @mcfunley

GitHub LinkedIn