Choose Boring Technology
看到新奇技术跃跃欲试？“无趣”的技术往往更可靠
March 30th, 2015
2015 年 3 月 30 日

原文链接：http://mcfunley.com/choose-boring-technology

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.
有 Kellan 做我的主管大概是我职业生涯中最棒的一件事。我花了足够长的时间去观察 Kellan 在技术方面的种种决定，从这一过程和结果中受益良多，并在我之后的工作中开花结果。感谢 Kellan，我能作为一个工程师写下《现在，用数据驱动你的产品！》，而不是为了做出正确的技术决定上面苦苦挣扎。

Being inspirational as always.
永远充满正能量

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.
在离开 Etsy 以后，我对技术恢复了热情。我终于可以整理我的思绪并把之前所学的东西诉诸文字。如果 Kellan 发现我对他的思想体系了解的如此之深，也许会把他吓一跳吧。

Embrace Boredom.
拥抱“无趣”

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.
每个公司大概有三次为公司增加新技术的机会。你可以随时使用它，但是它用完即止。在公司发展成熟，业务逐渐稳定以后，你可能还会有几次机会，但是你最好对此抱有保守的预计。

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.

每当你用 Node.JS 写网页，选择 MongoDB 作为你的数据库，或者使用存在不到一年的技术发现技术（Service Discovery Tech）时，你都在使用这一次次的机会。如果你还想自己写个数据库的话，天哪，你有麻烦了朋友。

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].
如果你的公司从事 javascript 咨询服务或者数据库业务，这些决定都可能是明智的，但这样的情况很少。大部分的情况是，你的公司要完成的是“重塑全球商业”、“重新发明网上支付”之类看上去高大上的使命。在这样的情况下，用你有限的精力去改进 ssh 只会华丽的失败，或者你运气好的话，让成功迟一些到来。

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.
但是，如何定义“无趣”却有些微妙。显然你不会去用那些既无趣又糟糕的技术，但是现在市面上仍然有很多“无趣”但靠谱的技术，比如 MySQL，比如 Postgres，比如 PHP，比如 Python，比如 Memecached，比如 Squid，比如 Cron。

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.
那些“无趣”或者功能不够强大的技术的好处在于，它们所能做的事已经被很好地理解了，并且更重要的事，它们出错时，我们也知道如何补救。任何了解我的人都知道除非当前技术错误百出，我被搞得不堪重负时，我才会对其更新换代。

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.
当你在选择技术时，有些不确定性你可能会有所预计，而有些你对此则毫无头绪。举个例子，当你的数据库 CPU 利用率达到 100%时，你知道它可能会发生一些问题，但是当你在写入数据的时候垃圾收集进程停止了？莫名其妙对吧！

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.
我是“无趣”技术的坚决拥护者，但是可靠性仅仅只是原因之一。每当你做出一项技术决定时，你的团队，你的公司架构，你的整个系统都会受到影响。

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.
为你的公司增加新技术需要代价。显然，当你在使用 Ruby 的时候加进 Python 是不明智的，因为这么做所增加的复杂度会超过新技术所带来的边际效益。但是人们往往一提到 Python, Scala, MySQL 或者 Redis 便开始失去判断力，为了能在工作中使用最好的工具而对其负面因素视而不见。

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.
作为一个技术管理人员，你的职责概括来说就是将商业问题映射到各类软件应用的解空间内。如果你没有任何负担，你的确在各个方面都选择你所了解到的最好工具。

Problems
Technical Solutions
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.
当做出选择不需要付出高昂的代价时，道理很简单，选择你最顺手的工具。但是可以肯定的是，负担总是有的。我称其为“运营负担”或者额外的学习成本。我将会需要时不时的关注一下它，需要研究怎么进行单元测试，需要了解它的特性，需要自己写一个启动脚本。。。总会有事情会冒出来。

Problems
Technical Solutions
The way you choose technology in the world where operations are a serious concern (i.e., "reality").
当运营成为一项重要因素的时候，你应该这样选择。

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.
“在工作中使用最好的工具”这一想法的最大问题是你对“最好的”和“工作”这两个词的看法。你的工作是让公司能够继续运转下去，而最好的工具能够在发生状况以后让你不至于为了解决问题而抓狂。

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.
通常情况下，能够让系统可靠地运转远远比省力地搭建它要重要。成熟并且高效的开发者对此都深有体会。

Choose New Technology, Sometimes.
当然，有时候我们也需要新技术

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.
如果根据之前的原则一路走到黑，那就意味着选择了 Java 以后，就只用 Java 实现整个网站。下面是我的一些建议。

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.
首先，你必须认识到你必须对增加新技术进行有效的沟通，并且不要期望能够一蹴而就。新的技术最后会对整个公司产生影响，所以你必须让全公司的人知道有这么一回事儿。你的公司可能要求你事无巨细一一汇报，也可能放手让你去做而不用跟别的部门打招呼，但是不管怎样，你必须让大家知道这是一个你一定会开诚布公的话题。

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.
这里我建议你先想一想：你手头的工具是否足以解决你的问题？如果答案是“是”，你很可能会发现真正的问题是某人实在是太想用新的技术了。这种情况下，你应该果断地放弃这一想法。

I just watched a webinar about this graph database, we should try it out.
“我刚刚在网上研讨会上看到这个图数据库，我们应该试着用一下！”

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.
当你发现很少的几个技术就能解决大部分的问题时，你也许会很惊讶。即使有一些看似无法解决的，也可能只是实现起来太复杂而已。所以当你觉得你无法用你现有的技术解决问题时，再想一想，也许会有创造性的解决办法。

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.
当你发现解决问题的办法太困难成本太高时，一个有效的办法是写下每一个导致这一情况的直接原因。这跟前面的建议有点相似，但又有细微的差别。

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.
新技术可能正好能填补你的技术空白（比如当你需要缓存系统，你可能会上memecached），但是新技术也可能和你正在用的技术有所重合。如果情况是后面一种，你需要对此抱有明确的预期。划定好时间线，告诉所有人“我们必须引进新技术”，这样你才能够将意外状况限制在可控制范围以内，并且避免为了优化你的解决方案短时间内引进各种新技术。

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.
不过放心，这个过程不会让你头大，也不会引起论战，这只是你的一个课后作业。你只需要回答完每个问题，然后在会议里讨论它就行了。如果你想要增加新的技术或者在你的基础架构上增加新的服务，用这些问题考验你的想法，然后你的心里应该就有答案了。

Just Ship.
想清楚就尽快开工吧

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.
多语言编程的卖点就在于它能够让开发者随心所欲的选择工具解决他们的问题。但是事实真的是这样吗？这个想法不仅太傻太天真，并且十分的有误导性，因为你会发现你日复一日地为了让系统正常运行花费大量的时间。

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

所以说，明智的技术选择能够让你的工程师们有时间和自由去考虑那些更重要的问题，并且那些新技术所声称的好处并没有被证实。

Update, July 27th 2015: I've produced a talk based on this article. You can see it here.
2015 年 7 月 27 日更新：我以这篇文章为主题发表了一篇演讲，链接请戳这里。

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.
1.Etsy 在早些年里深受其害。我们招了一大帮 Python 工程师然后决定要给他们找点事儿做。我们能想到的唯一一件事就是让他们写了一堆毫无意义的中间层，并且我们花了好多年我们才把这些玩意清理干净。与此同时，90%的搜索延迟仍然高达两分钟。Etsy 并没有因此失败，但是它在好多年以内毫无作为，并使它花了更多的时间得以成功。

We often casually refer to the boring/bad intersection of doom as "enterprise software," but that terminology may be imprecise.
2.我们常常把那些集无趣和糟糕于一身的东西叫做“企业软件”，当然，我是开玩笑的。

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.
3.（上文中未直译拉姆斯菲尔德，此句省略）

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.
4.Etsy 的活动订阅源可以很好地说明这一点。当我们尝试要增加这一特性时，我们花了很大的功夫去把 PHP,MySQL,Memcached，Gearman(一个 PHP 工作服务器)和我们的系统整合在一起。但是和用 Redis 实现上面的特性相比（当然这依然可行），前者就显得简单多了。
这个项目之后运行的一直不错。我们的注意力在接下来几年里都在其他方面，而活动订阅源在这段时间内规模增加了 20 倍并且完全没有人去照看它。因为使用了共享平

台，不管需求是什么我们都不用调整这个项目。这就是使用有限制性的技术能给你带来的长期好处。
当然，凡事都不是绝对的。活动订阅源中使用 memcached 是靠谱的，仅仅使用 PHP 实现全文搜索却不是，所以 Etsy 最后使用了 Solr。