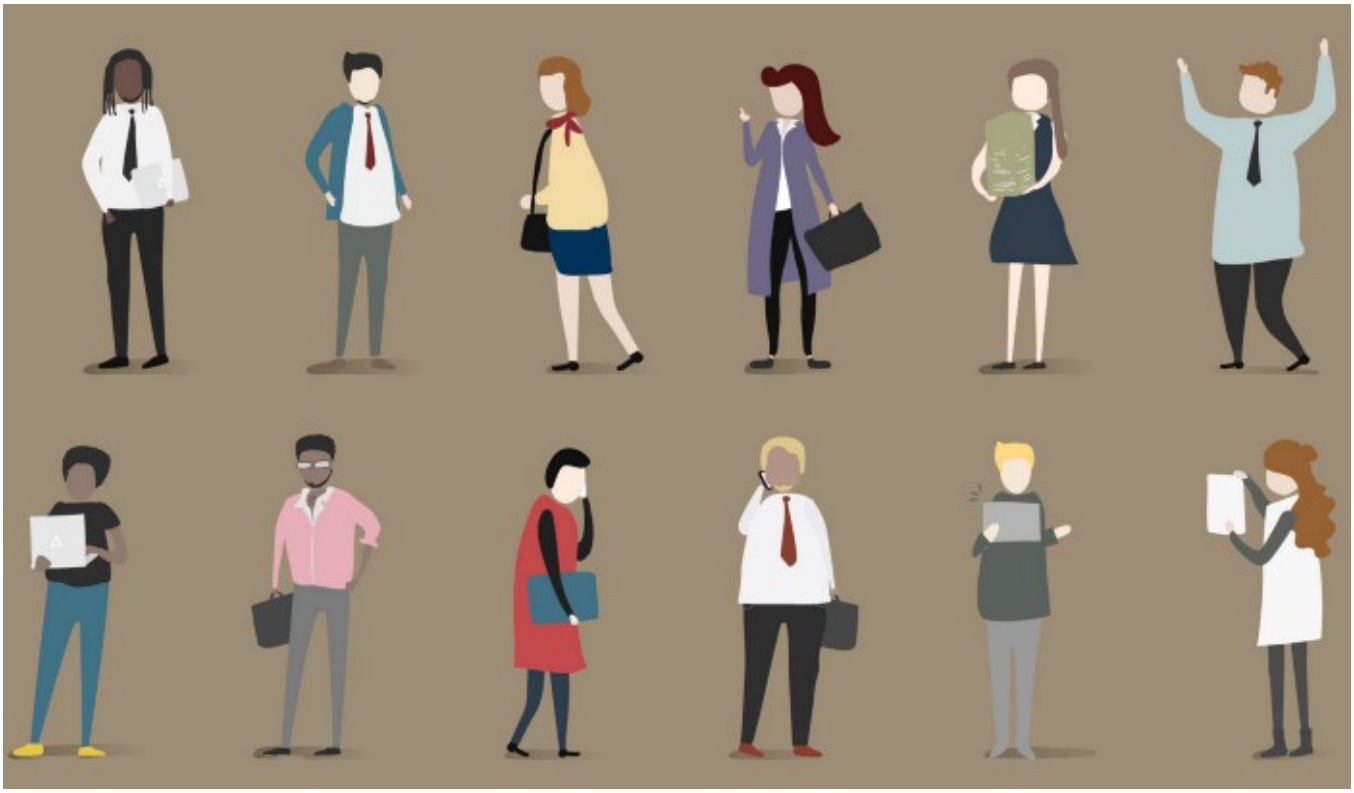
All the best engineering advice I stole from non-technical people

Marianne Bellotti



Vector created by rawpixel

As I focus on becoming a better manager of engineers, I have been reflecting more and more on the advice that produced a 10X boost in my abilities at that same stage. More often than not the best advice, the things that stuck with me, came from people who had no background at all in software.

I have been blessed (or cursed depending on your viewpoint I guess) with the privilege of high impact work on critical systems. This is work that naturally brings out a lot of self-doubt and anxiety. Over time I have found myself facing the same challenges, writing down the same thoughts and retelling the same stories in response. It's intriguing that the stuff that really seems to make a difference in the quality of software never seems to be about software.

These are five of my favorites.

1. "People like us make our money in the seams of things"

Who said it: a Senior Official at the National Security Agency (NSA)

Context: There's a long story about how I ended up in a State Department conference room with the NSA, but suffice to say it was a pretty crazy meeting in which a lot of very important people with impressive titles said a lot of stupid things about computer security. This would be the first of many times in my career in the federal government where we came in expecting to have a productive meeting with senior level leadership and were treated instead to a completely pointless dick measuring contest between three letter agencies. I remember the woman — and yes it was a woman — who made this comment was redirecting the conversation after her colleague had said something both wrong and insulting in response to a question from DHS. She was trying to defuse a

situation, I doubt she thought much about her choice of words, but her insight was sharp and to the point. Security and reliability are more likely to go wrong in the seams between components. That means literal integrations, but it also means organization seams. Places where no one is sure who owns what, or who is responsible for what are unlikely to have proper monitoring and much more likely to be two or three upgrades behind. The seams are where things get lost, sometimes for years. So if your mandate is security or availability the seams are your best bet of finding a big pay off.

How I changed my approach: Shortly there after I started applying a creative writing technique called 100:10:1 to evaluating and rescuing software systems. The idea behind 100:10:1 is simple:

- Brainstorm **100** things that could go wrong
- Pick **10** on that list that feel like the most likely and investigate them
- Find the **1** critical problem you're going to focus on.

The numbers themselves are not significant. The idea behind 100:10:1 is that you generate so many potential ideas you scrape the absolute bottom of the barrel and bring non-obvious things to the forefront. As you dig into the ones that feel the most likely, over time the list whittles itself down. Some ideas just end up flat out wrong, some merge with other ideas, some split off, some get ruled out because there's already someone looking at them... eventually you hit on something in the seams that everyone has missed.

Approaching problems this way also changed the way my team behaved for the better. Often the hardest part of working as a team is giving people permission to bring up ideas that are ultimately ruled out. People tend to self-censor to avoid looking or sounding wrong, because they assume everyone else is keeplng score. When we allow this on a team it creates blindspots. Fortunately, humans are very bad with probabilities. Very bad. I became comfortable with being wrong 95% of the time

because I understood that when I found that one critical problem all the ideas I had before that had been ruled out would be forgotten. I found a few problems much more experienced engineers had missed and it gave me super powers in the eyes of my colleagues.

When I shifted to a leader of teams the benefits became more pronounced. We talk a lot about psychological safety and giving people permission to speak their mind, but there's not a whole lot of guidance around how you do that. Being a person who is not afraid to ask the stupid questions and who lets ideas get thrown out easily without a big argument has created an environment where correcting each other and debate doesn't feel like that big a deal. It normalizes the process and people stop keeping score because it's clear that their boss isn't.

Often I find it useful to just acknowledge the situation up front: I tend to think in edge cases, but edge cases by definition are unlikely. As a result I tend to bring up a lot of irrelevant things. When you're discussing a problem or situation there are basically three buckets of information: things that are true, things that are false, and things that are true but irrelevant. Having a culture where people are not optimizing to avoid mentioning something true but irrelevant means we make better technical decisions overall because we get the benefit of every teammate's complete perspective.

2. "Know what people are asking you to be an expert in"

Who said it: Leslie Ryan, my TOEFL instructor

Context: Leslie certified me to teach english as a foreign language, a skill that allowed me to travel the world and work for small NGOs that could not otherwise pay me a living wage or sponsor a visa. He told a group of us this on the eve of our first solo experience in front of a classroom. I got

the impression that he gave this speech to every group he certified at precisely that moment in the program. Knowing that made it no less impactful.

Leslie pointed out to us that most of our future students would be professional people. They didn't need us to teach them about business, law, medicine, or economics. They were already experts in those things. What they needed from us was to learn how to do those things in English. They wanted to access our native fluency in order to supplement their existing skills. We should allow them to be experts in what they were experts in and remember that what they had hired us to be experts in.

Bonus advice from a technical person: "You didn't trick anyone, we know who we hired"

Who said it: Mikey Dickerson, Administrator of USDS

Context: One of a few meetings I had with Mikey in the basement of USDS's headquarters on Jackson Place just in front of the White House. I don't remember exactly what we were talking about, but I remember how this comment felt like a shot to the heart. Mikey had a habit of cutting to the quick like that. For many people it was unnerving, a point of contention and an endless source of misunderstandings and hurt feelings, but you got used to it after a while.

I hadn't thought I was making decisions based on imposture syndrome, but I did feel completely out of place working alongside some of the people who worked for USDS at the time. Engineers who had gotten in at the ground floor of some of the largest and most prestigious tech companies today — Google, Facebook, Twitter, Netflix, Amazon. People who had grown those companies to their current size. How could I NOT be overwhelmed?

I don't think Mikey meant to be profound. He was just stating the truth as

he saw it: USDS understood that I was not the greatest software engineer in the world. They knew where I was in my career, what my skill set was and where I was weak. They had hired me because I seemed to know how to get technical things done in a sustainable fashion in government. That was the skill set they were looking for me to bring to the table.

In the same conversation Mikey went on to tell me that "I would have paid you your whole year's salary just based on your first three months of work. Even if you had done nothing else, I would have happily just signed the checks."

After that I stopped getting so concerned about all these great engineers realizing I didn't belong there and started to focus on learning as much as I could from them.

How I changed my approach: I think about Leslie's comment a lot when I'm working with engineering teams. Too often engineers get so wrapped up in wanting to prove they're good enough they railroad colleagues with complimentary skillsets without realizing it. Software engineers suddenly act like they can be their own product managers, designers, copy editors, and sales people. It isn't always clear to professionals in other disciplines that this need to be a one person team is how engineers look for acceptance and respect.

When I find myself itching to interrupt someone with my thoughts about a topic I try to ask myself "what am I being asked to be an expert in here?" Often I realize that in my enthusiasm to show my casual knowledge I'm about to correct someone that who has devoted a considerable amount of time and effort into developing their expertise in the topic. I never regret keeping my mouth shut and letting them speak.

3. "Before you can make things better, you have to stop making them worse"

Who said it: A group therapy facilitator at NAMI

Context: The entire series of events that lead me to NAMI's friends and family group session in the summer of 2014 it too long and way too personal for this blog post, but I did end up there — $14,000 in debt, utterly heartbroken. I thought about this statement a lot back then. Later on when life returned to normal I realized that those same troublesome patterns of behavior I had had to confront lurk in much more banal situations too. The extremes only made them obvious, not unique.

How I changed my approach: A big part of what I do as an engineering manager is stopping truly brilliant people from executing on plans that begin with the words "I can just do this myself in a weekend."

If you're doing things for colleagues and denying them the ability to either do it themselves or participate in the process, you're not helping, you're making them dependent on you. Even if the end result is much faster doing it yourself.

Not long ago we had a Staff Engineer on another team reach out to my team and offer to migrate us to a new system for deploys. He had just done it for one of his team's services and was confident he could finish it up without any help from our team. He highlighted this as an advantage. We didn't need to reprioritize. He wouldn't need any effort from us at all! When I said no I got push back from my team. It sounded like a great deal. The Staff Engineer was so talented, why would I turn down the help?

I pointed out that if someone did it for us, then we wouldn't know how the new system worked and that might cause problems long term that would invalidate the time saved by accepting the offer. Still my team was not convinced, so we compromised and requested a meeting to go through the steps to move to the new system. If it was as easy as it sounded we would accept his offer.

Turns out there were several requirements specific to our service that weren't available yet in the new system. Even worse, some of these missing pieces had no clear resolution. The Staff Engineer would have gotten stuck and either had to pull my team into a migration we weren't ready to do yet, abandon it halfway through, or make critical architecture decisions for us.

Doing things for people is rarely as helpful as it seems like it should be. If they don't understand what you built, you've made things worse. If they don't know how to maintain it, you've made things worse. If you didn't know enough about their requirements to get implementation correct, you've made things worse. If they don't care about using or maintaining the thing you've built because they didn't build it and have no sense of ownership or obligation to it, you've made things worse.

It's almost always better to help by supplementing existing efforts rather than taking some work away and bringing back a solution.

4. "To go left, turn right"

Who said it: Sifu Jesse Teasley

Context: I met Sifu when I was about 16 years old. He had dreadlocks and taught me how to fight with a sword. He also introduced me to Chinese philosophy, specifically: Taoism.

How I changed my approach: One of the core concepts of Taoism is the harmony of opposites. Basically Taoism acknowledges that the human mind loves binaries: white -vs- black, right -vs- left, soft -vs- hard, good -vs- evil. This is the way we are wired to think, but it's also inaccurate. Nature does not honor such strict distinctions. To the point that their usefulness at all is debatable. They make complex thought easier by eliminating some aspects of variation, but they also make us vulnerable to big mistakes when the pattern doesn't hold.

Taoists understand that we're inclined to over-optimize based on these false dichotomies. If softness is a desirable feature in given context we try to eliminate all hardness. If white is preferable, we scrub away the black. These states of purity are not just unattainable, they are built on a dichotomy that is not real. This insight went on to influence Buddhism, particularly what would ultimately become Zen Buddhism.

Taoists believe that the path to the truth is in harmonizing the dichotomy. Recognizing that humans are unlikely to be able to abandon binaries altogether and simply putting a bit of black in the white, and a bit of white in the black. That's why the symbol of Taoism is the yin-yang.

I liked listening to Sifu because he framed this stuff in a very pragmatic way which made it less spiritual and more applicable to me. Another thing he used to say a lot was "when you punch someone you need to pull your arm back, before you launch it forward. If you don't your hit will be weak." It was a variation on the same thing: we are stronger by considering the opposite first.

We all know we should challenge assumptions, but that calls on us to realize we are making an assumption in the first place, which is not as easy. The Taoist advice of not moving in any direction before looking for and considering the opposite path bypasses this problem. Everything is an assumption when you are doomed to see in binaries.

5. "Thinking is also work"

Who said it: Unknown

Context: I always associate this statement with an older woman whose charming pre-war Bronx apartment I ended up at for a Winter Solstice party as the college student, but that person was a stranger and it seems unlikely that we talked that much. I have tried to match up this advice against other possible sources from my past lives to no avail. It's a false

memory that has stuck with me throughout my entire adult life 🤷‍♀️

How I changed my approach: On a personal level it gave me permission to take time when I needed time. Why should I feel guilty about leaving the office to go on a walk? Thinking is also work.

But it also influenced how I ran engineering teams. When I was in a traditional office environment I used to tell my people: If it's 2pm and you've finished your work for the day and you have no meetings, just go home. You're not cheating the organization, you're putting that energy in the bank. You're going to have some on call rotation where you get paged at 3am. Or a hard week where we have to work late to get something out the door. These things happen and are impossible to predict exactly when. If you're done for the day go home, relax, spend some time with family. Put that time in the bank because we will certainly spent it later.

It seems silly that people need to be told that, but many of us have been trained to believe that if we're not seen to be working, then management will assume we're slacking off.

I think of my role as a manager as part ad exec. My job is to sell the engineers that report up to me back to the organization that hired them in the first place. An organization's nature is to drift toward over optimization. This means it isn't enough for all our employees to be super productive, there has to be a clearly defined method and process that creates that productivity so that it can be reproduced to *guarantee* productivity.

Yet you could fill a whole library with research that shows this is not how effective teams are built or run. Effective teams need trust. That's not to say that frameworks for decision making or metrics tracking are not useful, they are **critical** — but replacing trust with process is called bureaucracy. The trust must exist first in order for KPIs, OKRs, SLOs (or whatever acronym is selling books now) to work.

But trust also degrades naturally over time. Italian researchers Cristiano Castelfranchi and Rino Falcone have a model of trust in which it's ***observability*** not success that is the key factor. Under their theory an entity that is silently successful can end up seen as less trustworthy than an entity that visibly fails. If we recover from failure quickly and efficiently, trust increases. Whereas when we succeed and no one notices we become more and more unknown and uncertain. This explains what is known as the *service recovery paradox*, when consumers trust a service provider more after a failure than they did before the failure.

Organizations have to constantly be reminded that they hired great people who know what they are doing. They know that in the beginning, but over time if the value of their employees is not observable then the trust degrades and bureaucracy becomes more and more attractive to leadership. The reason why people have to be told to take time off when they have unlimited vacation time, or go home early if they're finished at 2pm, or not to answer emails after midnight is because they have innate understanding that being *observed* working is more valuable than ***the results of their work***.

Obviously this creates a culture that leaves everyone worse off. The employees burn out. The organization's efficiency degrades. Poor performance decreases trust which increases bureaucracy... The turning point in my life was the day I realized to run great engineering teams I didn't need to be the best engineer in the world, I needed to get good at advertising my people and their stories up the chain of command. I needed to improve their observability so that we can keep bureaucracy at bay by maintaining a high level of trust.