Jacob Kaplan-Moss

Designing Engineering Organizations

How should you structure a larger engineering organization, one with dozens (or hundreds) of engineers? There are many tradeoffs to consider, and no single right answer. But, there are some structures that work better than others.

Summary: the most effective teams are **stable, multi-disciplinary, aligned to product delivery**.

The guiding principle here is that Conway's Law is inescapable – you ship your org chart. And, wherever possible, you should optimize for delivery. (This seems obvious. Is it obvious?) Thus, you want to try to build an organization that's as aligned to what you're actually shipping as possible. In other words, you're aiming for a sort of Reverse Conway's Law: design your organization to match what you want to ship.

Otherwise, your organization will struggle anywhere the product crosses organization boundaries. For example, if you have separate mobile and web groups, you'll struggle to ship a product that works consistently across platforms. This doesn't mean that there aren't times when it's appropriate to break this guiding principle – there are often compelling reasons to design teams differently. But be aware of the tradeoff in delivery speed that you're accepting.

Let's look at these tradeoffs in more detail:

Product-aligned vs technology-aligned

With **product-aligned** teams, each team is responsible for some piece of your public-facing product. E.g., if you're building a music player, you might have a playlists team, a music ingest team, etc. Another way of looking at it: if an outsider looking at your product could draw a reasonably accurate org chart, you have a product-aligned org.

Technology-aligned organizations break down based more on backend technology. So you might have a database team, a backend software team, a frontend web team, a mobile app team, etc.

Generally, product-aligned teams deliver better products more rapidly. Again, Conway's Law is inescapable; if delivering a new feature requires several teams to coordinate, you'll struggle compared to an org where a single team can execute on a new feature.

It's not always easy to build product-aligned teams, especially at scale. Larger products can be hard to break down along product lines to a point that's granular enough to afford well-sized teams. And, there's often foundation engineering that doesn't align to a single product feature.

Larger organizations often end up needing a mix: a set of platform teams that build and maintain underlying infrastructure, shared libraries, and tooling; and a set of product teams that directly work on the product.

Stable vs project-based teams

On **stable teams**, staff join teams and stay indefinitely. Teams are long-lived, and projects come to teams: teams may work on different things over time, but the people stay the same.

Project-based teams form to tackle a specific project, and dissolve when it ends. Teams are ad-hoc and purpose-driven. Each project gets a different assortment of staff.

Stable teams are the norm within product organizations, while project-based teams are more common at consulting shops or within service organizations.

Generally, stable teams produce better results. It takes teams a while to "gel" and work at their highest productivity as they work through the forming–storming–norming–performing phases of team cohesion. My experience is that it takes at least 6 months to reach the "performing" level, and sometimes much more. If staff switch teams more often than that, teams never reach their full potential.

The main upside of project-based teams is more flexibility of skill-set and staffing levels. This is why consulting organizations tend towards project teams: it's nearly impossible to predict what sort of skill/staffing your next client will need, so forming a new team for each new client can be worth the tradeoff in lost productivity. But, consulting orgs that can

manage stable teams – e.g. by having a pool of teams, and having projects assigned to the most appropriate team – usually perform better.

Occasionally, product orgs will want to create ad hoc teams to tackle certain important strategic projects. An example from my time at Heroku: we were an organization with stable teams, but when we decide to build Private Spaces, we decided that project was important enough (and technically challenging enough) to warrant building a special team to focus on that project, even though it cut across the org in a complex way.

This can be an effective move, especially if the team is expected to last long enough to get through its growth pains. Dedicating a new team to a special project is a great way to break organizational inertia and get something new shipped. It also has some downsides: there's often competition to get on to these "special" teams, resentment from those who aren't chosen, and reluctance for people to fold back to their original teams when the project spins down. (We saw both of these at Heroku, but the trade-off in execution was worth it.)

Single- vs interdisciplinary

Most teams aren't pure-engineering; they need other skills. Design & project management are the big ones. But even just within engineering, in a product-aligned team some teams could need staff with a pretty wide variety of skills - front-end, backend, mobile dev, etc.

Single-disciplinary teams group staff by skill-set – e.g.e.g., a backend team that uses Python/Django, a frontend JS/React team, Android and iOS teams, etc. Product and Design are separate teams/orgs, and those staff float and sometimes support multiple teams.

Interdisciplinary teams combine all the pieces a team needs to execute under one roof. So a single team might have backend and frontend developers, designers and product managers, content producers and editors – whatever might be needed to ship.

Generally, interdisciplinary teams outperform organizations built with single-discipline teams. This is for the same reasons I've talked about above: Conway's Law and team cohesion. If Design is a separate function from Engineering, you'll pay an organizational cost every time your org needs to ship a feature that requires both engineering and design (read: all of them). This often leads features where the design feels "slapped on" at the last minute, or stuff that looks pretty but is functionally broken.

However, executing on interdisciplinary can be really tough. For one, teams can get unwieldy: a complex product might require so many different skill-sets that it might require way too many people to form a single team. There might be a way to decompose that product, but sometimes there isn't.

Interdisciplinary teams also have challenges around reporting chains; see the next section.

Reporting chains: delivery-aligned or skills-aligned

In a multi-disciplinary and/or product-aligned team, how does reporting work? If there's a designer on an integrated team, who do they report to? The manager of the delivery team, or a designer?

In **skills-aligned reporting chains**, people report to managers who match their skill-set. Engineers report to engineering managers (who roll up to engineering directors and VPs of engineering); product managers report to other product managers (and up to a VP of product); designers report to designers, and so on.

With **delivery-aligned reporting chains** staff reports to the manager of their delivery team, regardless of discipline. So a delivery team of designers, engineers, and product managers all report to the same person (who might be from any of those disciplines).

Generally, delivery-aligned is more effective. Organizations are trying to optimize for delivery, and having one team – including its manager – wholely responsible for delivery is highly effective. Skills-aligned teams can struggle with actually shipping: when they miss deadlines it can be easier to point fingers at some other team than accept responsibility.

The biggest problem with skills-aligned teams is that they often devolve into matrix management. People end up formally reporting to some manager, but work day-to-day work under a different project manager or team lead. (Or, worse, multiple other project leads). Since managers don't see day-to-day work, they won't be able to give effective feedback or understand their team's performance. At best, they can ask project leads how things are going and relay that information, but games of telephone don't make for great feedback. You'll see this most clearly during performance reviews, which are incredibly difficult and fraught because managers don't have first-hand knowledge of their staff's performance!

Usually the objection to delivery-aligned teams goes like this: "hey, I'm an engineer, I can't report to a project manager! How could they possibly understand my job!?" This is – to use a technical term – bullcrap.

The fact is, *management* is the most important skill of a manager. We know that great engineers don't necessarily make great managers; that's because it's a pretty different set of skills. The skills of management (feedback, coaching, conflict resolution, team building, etc) translate quite well across disciplines.

Now, it's true that domain experts need domain-specific guidance and feedback, and that a manger from a different discipline can't always provide that feedback. That is, I can help a Python programmer debug a broken tests, but I can't give a designer guidance on selecting a color pallette. (Well, I can, but nobody will be happy with the outcome!)

So, organizations with delivery-aligned teams need to create other channels for domain-specific guidance and feedback. This usually takes the form of "guilds" or other kinds of communities-of-practice.

The most effective teams

If you're trying to optimize for delivery, then **stable**, **multi-disciplinary**, and **product-aligned** should the principles that guide how you form teams. There are reasons (some of them good) to deviate from these principles, but the tradeoff will always be that it'll be more difficult to ship.

Thanks to Aidan Feldman for asking me a question about engineering organizations that inspired this post.

Published January 5th, 2021.

Table of Contents:

Tags: engineering management organization structure

Previous by date:

AI is catching up to the hype

The Algorithm thinks these articles are similar:

- A reading list for new engineering managers
- Measuring Hiring Manager Effectiveness
- The Innovation/Execution Spectrum