HOW TO STRUCTURE YOUR ENGINEERING TEAMS

by Kenneth Lange

First Edition

CONTENTS

Chapter 1

INTRODUCTION

The team structure you pick for your engineering organization will have a massive impact on its effectiveness and productivity. Research by J. Richard Hackman, professor at Harvard University, suggests that 60% of a team's performance is determined by its structure[1].

On the other hand, you only need to talk with a few software executives before you realize that they structure their teams in very different ways. Why is that? Why has the software industry not found a team structure that outperforms all others and established it as best practice? If team structure is the most important key to success, there should be a strong motivation for this to happen.

The short answer is that the right team structure depends on what is important to the company: is it speed to market, technical excellence, or something else?

While software companies structure their teams in very different ways, they tend to base their teams on one of three generic team structures, which they then adopt to their circumstances:

1. Technology teams: The horizontal approach where you organize your teams along the layers in your tech stack; for example, a frontend, a backend, and a database team. Each team is formed around a technology, such as Android, and all team members report to a manager who is also skilled in that technology.

2. Product teams: The vertical approach where you organize your teams according to the business areas in your solution; for example, a customer and an order team. Each team is cross-functional and has all the different skills needed to deliver its product.
3. Matrix teams: The mixed approach where you seek to organize teams along product and technology dimensions at the same time. In this team structure, the developers report to a development manager, but they are "lent out" to cross-functional product or project teams where they do their daily work.

These team structures can be customized in many different ways; for example, both Microsoft and Spotify have used matrix teams, but in vastly different ways; and companies can mix and match elements from each team structure. For example, some companies, like Instagram, use product teams for the majority of their teams, but have a technology team for their underlying technology platform. And some companies use a technology team when they introduce new technologies, such as AI or mobile apps, and later integrate the team members into the rest of their product teams.

In the next chapter we will dive into each of these team structures to explore their strengths and weaknesses, and see how world-class companies are using them and adjusting them to fit their special needs.

Chapter 2

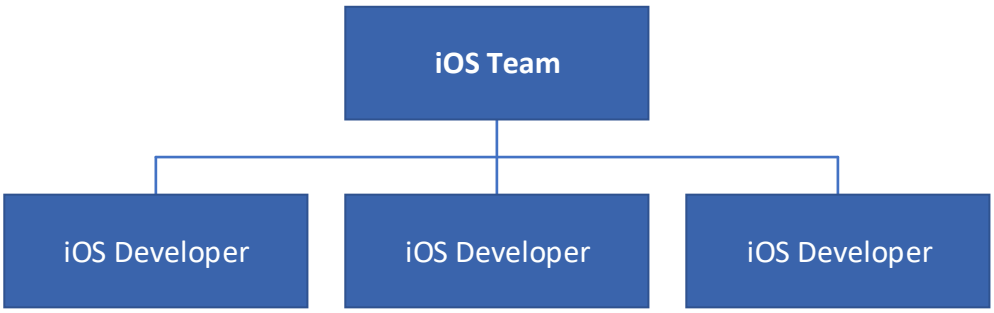
TEAM STRUCTURES

Team Structure 1: Technology Team

A technology team is a team of software developers who work with the same technology; for example, a small company may have a frontend and a backend technology team, and a larger company may have teams for individual technologies, such as Java EE or Oracle Databases.

A technology team consists exclusively of software developers who are specialized in a specific technology. This means that the technology team has no product managers, testers, or even developers working with other technologies. All team members report to a development manager who is also skilled in this technical area.

An example of a technology team is an iOS team, which develops apps for Apple devices. The team consists of a number of iOS developers who report to a development manager who has deep knowledge about developing iOS apps:



Technology teams, such as a frontend and a backend team, within the same company rarely share any code. Their code is typically written in different languages and frameworks, such as

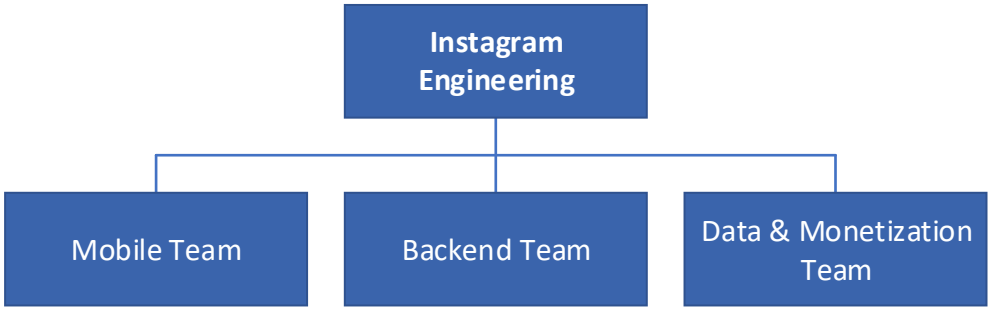
Swift or Python, and exists in different code repositories. The technical interaction among the teams usually happens through a REST API[2] or similar.

The reporting lines in such an organization are based on expertise. In other words, a mobile developer reports to a mobile development manager, and not a backend development manager or a product manager.

The managers in such a development organization are likely to be senior engineers who have been promoted to management positions and now also handle people management on their teams. A manager will still be writing code, or at least will have the ability to do so, and usually also handles the project management of the team's work and coordination with other teams.

The rock star in such an engineering organization is likely to be a technical expert, whose skill is measured by some technical standard, such as in-depth knowledge of the team's technology or his or her ability to write the most elegant and concise code.

A real-world example of using technology teams is early Instagram[3] (around 2015) where their engineering organization consisted of three technology teams:



As the company grew, Instagram eventually moved away from this structure as you will see in the following section about product teams.

Strengths

The primary strength of the technology team structure is technical excellence, which will be higher than in any of the other team structures.

A technology team's codebase is likely to be of a high quality, to take advantage of the latest advancements within the chosen technology, and to contain very little technical debt.

It can also be easier to recruit top-notch technical experts for such a team. For example, if an engineer is intensely passionate about Django then the idea of working in a Django team, reporting to a Django manager, and being surrounded by Django experts is an almost irresistible proposition.

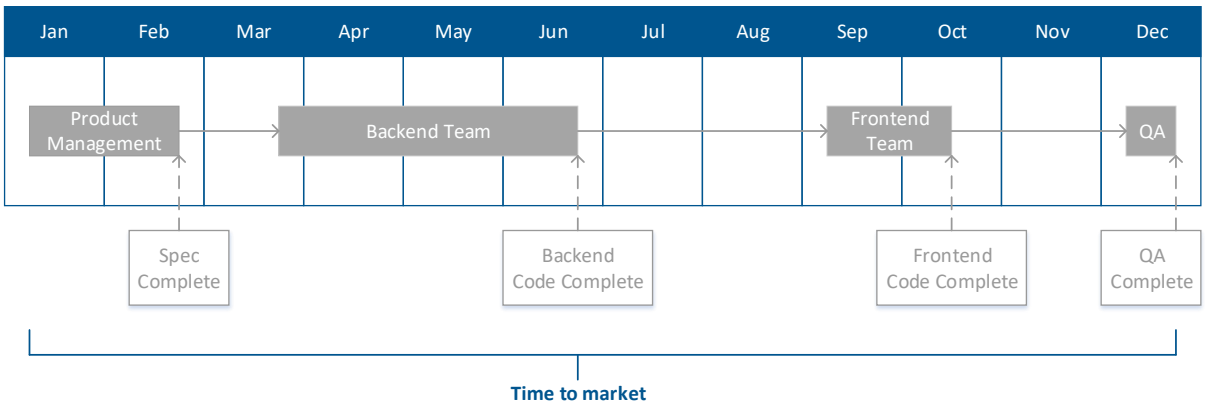
Finally, a technology team's manager is likely to be highly competent in the technical work that the team performs, which is a key to high job satisfaction according to recent research[4].

A technically competent manager can evaluate developers based on merit, rather than some measure that can be easily faked, such as who stays the longest in the office. The manager can also provide detailed coaching on how to write better code, and will have better awareness of when a developer is ready to be promoted to the next level

Weaknesses

A frequent problem in engineering organizations that use technology teams is that their time to market (for new features) tends to be slow.

The reason is that one team may finish its part of the new feature fast, but the next team might be busy with something else and thus unable to work on its part anytime soon – as shown in the diagram below:



This means that work backs up between the teams. The cost of a new feature may not be high in actual development time, but it can be very expensive in calendar time, and hence, may result in a slow speed to market.

This problem should not be underestimated. Speed to market is hugely important for most businesses. In lean thinking, unfinished features are expensive inventory that cost the company money, because they do not generate business value until they are in production and the end users start to benefit from them.

This team structure also nudges you toward phased or waterfall development where each team finishes its part of the work before passing it on to the next team. This discourages iterative

development and short feedback loops, and when mistakes start to occur, due to limited communication between the teams, the handovers are likely to become cumbersome and time-consuming, and can even lead to a destructive "us-versus-them" attitude between the teams which will discourage further collaboration.

In an attempt to overcome the weaknesses of technology teams, namely slow speed to market and poor cross-functional collaboration, some engineering organizations have introduced product teams that focus on product areas (or verticals) instead of technical layers (or horizontals). Their reasoning is that organizing around products will lead to improved collaboration among different roles, such as frontend and backend developers, because they will literally be on the same team. On top of that, they also expect that speed to market will increase, because when a product team takes on a new feature it has all the skills necessary to finish it.

Team Structure 2: Product Team

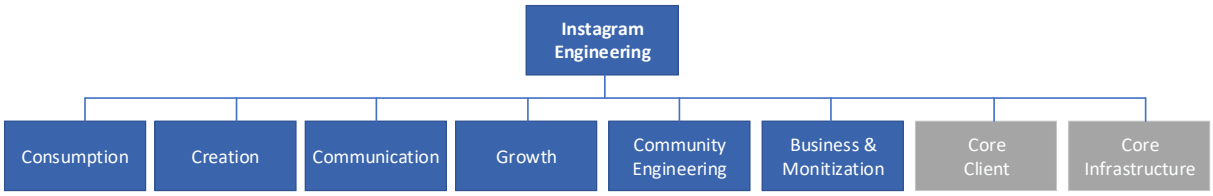
Product teams are organized around a company's product areas, such as customers or orders, instead of the technical layers in its tech stack, such as frontend and backend.

The motivation behind aligning team structure with product structure is to align the teams' success more closely with the company's success. In a product team, the measure of success is less likely to be technical excellence, as in a technology team, and more likely to be how well the product performs in the marketplace.

A product team needs all the different roles, such as developers and testers, that are required to develop and maintain its product area. A single manager is responsible for the team, and all team members, regardless of their role, report to him or her. So it doesn't matter whether a team member is a UX designer, a frontend developer, or a tester, they all report to the same line manager.

A key reason for having all team members report to the same manager is to simplify decision making – the buck stops at the line manager regardless of the functional area. Another reason is that it encourages people on the team to learn more about the business area that the product serves.

When Instagram moved away from technology teams, they adopted the product team structure that you can see in the org chart below:



Instagram organized their teams around product areas, such as content creation and communication. To handle cross-cutting concerns, they added to platform teams: the Core Client team for developing the container (or app shell) that the product teams develop their product areas within, and a Core Infrastructure team that handles servers and other infrastructure.

The line manager in this team structure is often called an engineering manager to show that he or she is not a manager for a specific technology area, such as mobile development, or a specific discipline, such as testing, but rather a manager responsible for all engineering within a product area.

Engineering organizations that use the product team structure tend to grow leaders who are good at bringing different disciplines together and making them build a unified product where all the pieces fit nicely together.

The pattern also encourages leaders to focus on building a product that actually solves a business problem: it aligns the manager's and the team's success much more closely with the company's success and it becomes much easier to define business-relevant KPIs, such as monthly active users, for the team.

Another interesting dynamic in companies that move from technology teams to product teams is that full-stack engineers with a good understanding of the product area tend to replace the technical expert, a specialist in a single technology, as the

rock stars of the engineering organization. The reason is that a specialist can typically only build part of a feature. For example, a Django expert can develop the backend functionality, but not the frontend technology, because it is written in React which he or she does not know. But the developer reports to a line manager who is responsible for the whole product, and not just a single technology, and is therefore more interested in shipping complete features fast, and hence is more likely to reward people who can deliver complete features.

This dynamic is further accelerated if it is a software-as-a-service company that uses continuous deployment and competes in a business where speed to market matters, which tends to be true for almost all businesses.

In many traditional companies, there is one department for development, which develops the software, and another department for operations, which runs the software.

While great improvements have been made in making cross-functional product teams within development departments, the walls between development and operations remain strong. This slows down speed to market for new features. That is, a feature may be completed by development, but if operations do not have time to deploy it to production, then it doesn't matter to the end users as the feature is still unavailable, just for a different reason.

Some companies for whom speed to market is business-critical, realized that they needed better collaboration between development and operations in order to increase their speed to market. Therefore, so they started to form DevOps teams with

both development and operations people, so the teams can deploy to production when a feature is ready.

This thinking is a natural continuation of the product team structure's goal of being end-to-end responsible for a product area, and as the world evolves slowly but surely toward software-as-a-service with microservices, continuous deployment, and serverless computing, this structure is likely to become increasingly popular.

Airbnb: Persona-based teams

Airbnb has made an interesting adjustment to the product team structure[5]: they have their product teams focus on specific personas, such as guest or host, instead of more traditional product areas, such as billing or booking. This makes it easy for Airbnb to empathize with its end users and establish KPIs to measure their satisfaction.

Amazon: Two-pizza teams and fitness functions

At Amazon, they use two-pizza teams which consist of the number of people you can feed with two pizzas, roughly 6 to 10 people[6]. The team is headed by a team lead, who agrees on a fitness function (a single key business metric) with the management team. The team lead, and his or her team, is then given autonomy to optimize for this fitness function in whatever way they want, and the team lead essentially works as a mini-CEO for the product area.

Scrum and product teams

Many companies use the Scrum framework[7], first introduced in the early 1990s, to structure their self-organizing, cross-functional teams.

A Scrum team consists of three, pre-defined roles:

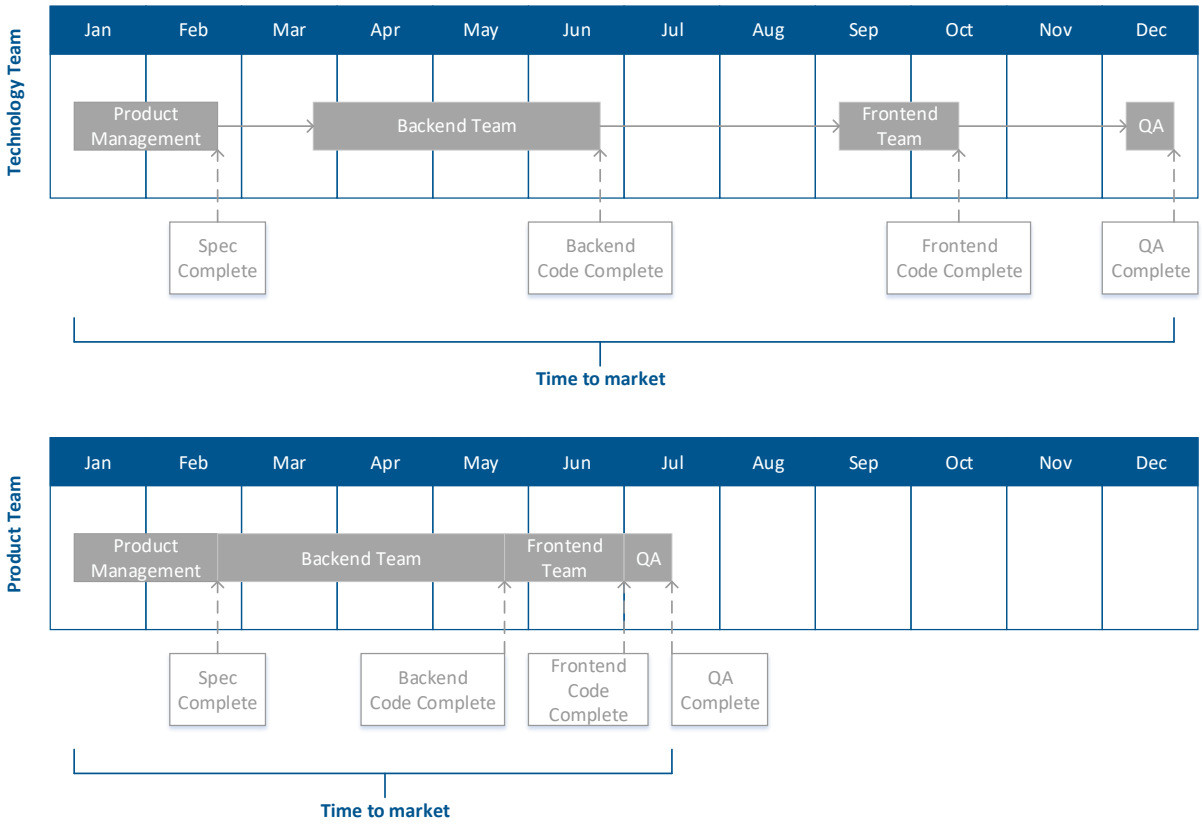
- Product Owner: Defines and prioritizes the development team's work.
- Development team: The people, such as software engineers, UX designers, QA specialists, who do the actual work.
- Scrum Master: Coaches the product owner and development team in working more effectively with Scrum and removes impediments that slow down the team.

Some companies combine the product team pattern with the Scrum methodology by letting the line manager be the Scrum Master. There has been much heated debate about whether this is a good idea: the strength is that it will be easier for the Scrum Master to remove impediments and coach the team; the weakness is that the Scrum Master can become too powerful, so there is no longer a self-organizing team of peers where the best ideas win. In the end, whether this setup is a good idea depends greatly on the company culture and the personality of the line manager / Scrum Master.

An equally heated debate concerns whether to let the Scrum Product Owner become the line manager for the team.

Strengths

A major strength of a cross-functional product team is that all the skills necessary to deliver new features are immediately available within the team, so there won't be any time gaps between the teams, as we saw with the technology teams. This reduces the time to market for new features:



Another advantage of product teams over technology teams is that the teams are much more closely aligned with business success. A product team is unlikely to feel successful if their product has just flunked a major public review or if the number of monthly active users is declining month after month – even if their code is so beautiful that it could have been used as an example in Clean Code[8].

It is also my experience that more and more software engineers no longer really fit the old computer geek stereotype who just

wants to be left alone and code. They want to see their product succeed in the market and see it make a positive difference in people's lives.

A product team is more likely to build a unified product because they have better collaboration across all the different disciplines that are needed to build a software product and they have a lower risk of an unhealthy "us-versus-them" culture.

These strengths contribute to a product team's ability to iterate very fast and launch new product features quickly.

A caveat is that many of these strengths can be nullified if the product team has strong dependencies outside the team's control. These external dependencies can be organizational, like external reviews or approvals, or technical, like an architecture that is a big ball of mud where any change to the codebase can have unexpected side effects in other modules, so all teams need to carefully coordinate their work.

Weaknesses

A serious risk with product teams, compared to technology teams, is that they may devote less attention to engineering excellence and their technical debts might increase to unmanageable levels.

There may be several reasons for this:

1. Engineers become so focused on market success that they lower their engineering standards. This is a risk if the product team is led by a strong and opinionated product manager.

2. Each team becomes a silo, reporting to a single line manager, and there can only be so many senior engineers on a single team. So, for example, if a team only has a junior Android engineer, who will make sure that the quality of his or her code is satisfactory, or make sure that he or she knows the best Android blogs to follow?

There is also a higher risk of code duplication. That is, several product teams may independently develop the same functionality in their individual codebases, which may or may not be a problem, depending on your belief system.

Another risk is that it can become difficult to move people between teams. If an engineer needs to move to another product team, he or she will also need to change line manager. So there may be resistance to the move: the engineer might like his or her current line manager and not want to start over with a new manager, and the manager might be an empire builder who is not willing to "give away" an engineer to another team. The consequence is that the company may not allocate its people to its highest priorities or greatest opportunities.

Another risk, compared to technology teams, is that recruitment can be tougher. It is easier to explain to a React developer that it will be a great idea to join a React team than it is to explain that it will be a great idea to join the life insurance team. This risk can be mitigated by explaining why this area is interesting from a technical point of view or how it makes a positive difference in the world.

Some companies brand their product teams, at least in job ads, as full-stack teams, and given it is cool to be a full-stack engineer, their thinking is that it will be easier to recruit people for a full-stack team than for the life insurance team.

From a manager perspective, my experience (having been both a development manager and an engineering manager) is that being an engineering manager responsible for a product area is a more demanding job. It is not because the job is difficult from a technical point of view; it is because the responsibility is much broader. You will essentially become a mini CTO or VP of Engineering for a small software company.

You will also have a direct impact on the business, which you cannot shy away from. That is, as a development manager for a technology team you can say that the product is perfect from a technical point of view and it is not your problem that it has been a failure in the market. Due to the broader scope of the role there are also many more things that can go wrong and you will be responsible for things outside your area of expertise.

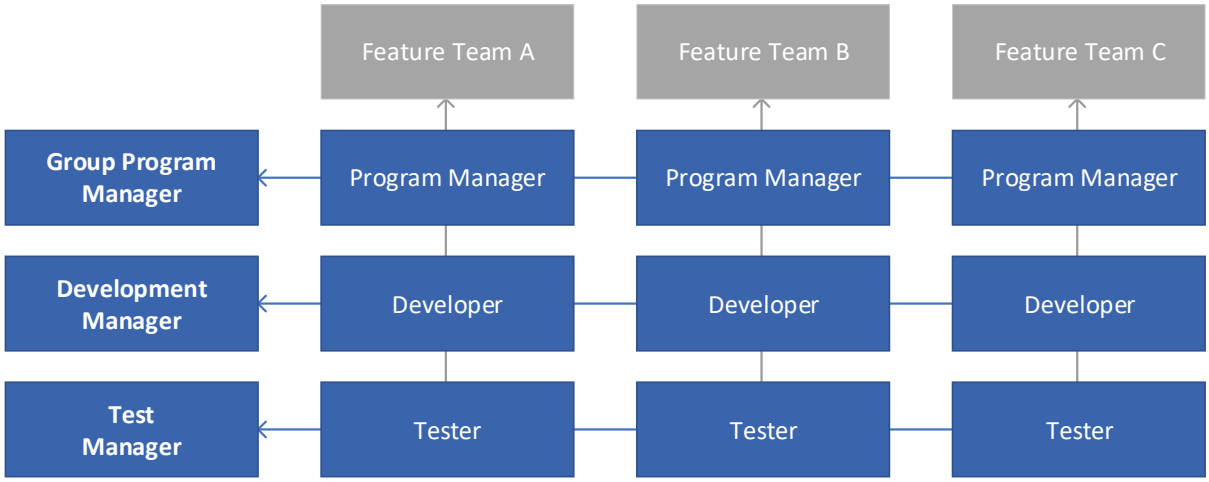
Finally, as a manager your technical skills will erode faster than in the other team structures. You will be responsible for multiple technologies, such as backend, frontend, and data, and for people management on top of that. The rapid pace of technological progress only accelerates this; for example, you were an expert in AngularJS and then they released Angular 2 and all your hard-earned skills became obsolete and you have no time to learn the new version. And this is not only happening in one layer of your tech stack, but in all layers, so keeping up with everything can become pretty tough.

In an attempt to keep the customer focus, but without losing their technical edge, some companies have introduced matrix teams that try to combine the best from both product and technology teams.

Team Structure 3: Matrix Team

A matrix team is a temporary product or project team that consists of specialists from different functional areas, such as product management, development, and testing. The idea behind the cross-functional nature of the matrix team is to increase collaboration between functions to make better products and faster releases.

An old-school example of this team structure is Microsoft Solutions Framework (MSF) which was popular back when Microsoft dominated the software industry around the turn of the millennium:



In MSF, a matrix team is called a feature team and focuses on a product area, and it will last for the duration of a product release or longer. For example, in Microsoft they had multiple feature teams working on a new release of Microsoft Excel, and one of those feature teams focused on Excel Macros.

A prototypical MSF feature team consists of one program manager[9], four developers, and two testers. The program manager (who would probably be called a product owner today)

writes functional specifications, prioritizes features, and handles coordination within the team.

But the program manager is not the developers' line manager. The developers report to a development manager who provides guidelines on how they should do their job by defining the development process, engineering practices, coding standards, and so on. The development manager is also responsible for people management, including hiring, promotions, training, salary, team transfers, and so forth. The other roles have the same structure: program managers report to a group program manager, and testers report to a test manager.

The underlying idea is that matrix teams encourage cross-functional collaboration because different specialists are working together in the same team and they have a shared goal of delivering a successful product. But at the same time the specialists report to a functional manager who ensures excellence within that area.

The matrix team structure is flexible and can be implemented in many different ways. One of the most common parameters to adjust is the power of the line manager versus the autonomy of the matrix team. Not surprisingly, older companies tend to favor a powerful line manager and younger companies tend to favor team autonomy. Another parameter that is often adjusted is the duration of the matrix team: is it a stable, permanent team, or a dynamic, task-oriented team?

Yammer: Task-oriented matrix teams

Yammer, a social network for enterprises, uses a modern variation of the matrix team pattern[10]. A key difference between Yammer and Microsoft as companies is that Yammer continuously deploy new features to production and do not have major product releases like Microsoft have for their flagship products.

Yammer's developers share the ownership of the entire codebase, so there is no such thing as "my code" or "my module". Each time a task needs to be performed on the codebase, Yammer establishes a temporary matrix team to perform this specific task. When the task is complete the code is released to production and the team is disbanded and developers are free again to join new ad-hoc teams to address new tasks.

Their thinking is that this is highly agile, and people will not be limited to work on a single product area but can quickly go wherever they are most needed.

The development manager is responsible for developers within his or her technical area, such as Ruby on Rails, Java, or React. However, the development manager no longer defines guidelines for how the developers should work, but instead acts as a coach who is focused on growing his or her developers into top-notch experts in their chosen technology.

Spotify: Product-oriented matrix teams

A different variation of the matrix team pattern is used at Spotify[11], an online music player, which prefers long-living, stable matrix teams, which they call "squads". Their reasoning is that it takes a long time to master a product area, such as Spotify Radio,

and mastery is needed to build awesome products for their users. On top of that, research shows that stable teams are more productive, by up to 60%, than volatile teams[12], because it takes time for a team to gel.

Compared to more traditional matrix organizations, Spotify empower their matrix teams and give them great autonomy, but they still have line managers, which they call "chapter leads", but with the important twist that the line manager is also an active member of a matrix team (for example, as a backend developer) to make sure he or she stays in touch with reality.

Strengths

The primary strength of the matrix team compared to the technology team is that it fosters much closer collaboration across functional disciplines. Now the developers and testers are part of the same team; The collaboration is further improved if the product owner has an exciting vision for the product area that unites the different functional disciplines. And similar to product teams, the matrix team has a lower time to market for new features than technology teams.

While the matrix team structure brings engineers closer to the business and makes it easier to see how they contribute to the success of the company, the engineers can still continue to seek mastery within their chosen technology and continue to report to a line manager who appreciates and understands their technical work. The line manager also enforces alignment and quality across teams, and the engineers will have a second opinion, and supporter, which is helpful if they have a powerful and persuasive

product owner on the team or they feel uncomfortable with the decisions taken within the team.

As with the product team structure, matrix teams also encourage developers and testers to learn more about the business domain that the matrix team is working in. This is really useful when it is a highly complex business domain with many counter-intuitive business rules. For simple domains, such as social networks or blogging, it is less important.

The matrix structure also scales well and can be used for delivering very large products. Microsoft released many of their most successful products, such as Windows and Microsoft Office, using this team structure. They were even able to compete with young startups, such as Netscape, while using this model. Obviously, Microsoft used some dirty business tricks to win the browser war against Netscape, but they would have been unable to compete with Netscape if they had failed to keep up with Netscape's development speed.

Weaknesses

A disadvantage of matrix teams, compared to product teams, is that decision-making can involve many more stakeholders, such as multiple line managers and product owners, which makes decision-making more cumbersome and time-consuming.

In theory, a matrix team has a high degree of autonomy, but in practice the line managers can enforce controls that limit the team's autonomy and the team will need to consult with the line managers before trying anything too radical.

There is also a risk that work processes inside the team will turn into small waterfalls with extensive handovers between the disciplines inside the team. This can happen when the line manager is not actually part of the team but defines the process that his or her people must follow within the team.

There is also a risk that the line managers may not see the big picture and may start to sub optimize for their functional area. For example, the test manager wants to introduce NASA-like quality controls, while the customers are actually happy with the current quality level and is much more interested in getting new features quicker at the current quality level.

Many developers who have worked in a matrix team feel like they have two managers, the development manager and the product owner, and they often receive conflicting signals about what is important. For example, the product owner says that the developer can skip unit testing to meet the deadline, but the development manager says that unit tests must be written for all new code – and the developer is caught in the middle. Unclear or overlapping responsibilities are a frequent source of conflict and frustration in matrix organizations.

Decision making related to how the team works may even turn into a lengthy process as multiple line managers may need to be involved in a single decision. For example, the development manager wants to introduce static code analysis and reduce the technical debts it reveals, which should be a pure development activity. However, the product owner feels that doing this initiative will delay the development activities already on the team's product roadmap, so she wants to be involved in the decision.

The test manager feels that it is an initiative related to quality, and hence, he should have a say in it, and wants to incorporate it as part of an overall test strategy.

As seen in the description of the three team structures, there isn't a single team structure that outperforms all the others in all dimensions; each comes with its own strengths and weaknesses, so which one should you choose?

Chapter 3

PICK THE RIGHT TEAM STRUCTURE

The right team structure for your software organization will depend on what is important to your company and its customers.

In my experience there are roughly four dimensions, or key result areas, that are important to software companies:

1. Product-market fit: The importance of delivering new features that match perfectly what the customer needs.
2. Speed to market: The time it takes to deliver new features and get them into the hands of end users.
3. Engineering excellence: The technical quality of the product, such as efficient code, few defects, and so forth. Plus, the need to introduce new technologies and to work with cutting-edge technologies.
4. Cross-team collaboration: The importance of collaborating across teams in the company. Are the teams independent or do they need to work closely together?

The order of importance of these dimensions depends on your company and its customers. For example, engineering excellence is likely to be more important for mission-critical software than for a startup still searching for product-market fit.

The different team structures will nudge you toward emphasizing some of the dimensions more strongly than others. But you have to be careful not to forget the remaining dimensions; that is, if you excel in product-market fit and speed to market, but are

disastrous in engineering excellence, with many critical defects, you are unlikely to be a success.

The arrows below are my subjective view of the strengths and weaknesses of each team structure when all other things are equal:

	Technology team	Product team	Matrix team
Product-market fit	→	↑	↗
Speed to market	↘	↑	↗
Engineering excellence	↑	↘	↗
Cross-team collaboration	↘	↘	↑

There are obviously many factors that can affect the dimensions; for example, great technology teams in a great company may have a faster speed to market than a so-so product team in a mediocre company. And a technology team may perform worse in the engineering excellence dimension if they have a legacy codebase full of technical debts.

Finding the right structure for your software teams is more of an art than a science, so the table above is not meant to represent a formula that can be blindly applied, but is a reference to inform your thinking. If speed to market is vital to your business, but you're using technology teams, maybe you should consider if that's still the right team structure for you. In addition, you may want to mix and match some of the team structures, as we saw Instagram did. For example, you might use product teams as your

default team structure, but use a technology team when you need to introduce a new technology in your company.

My subjective observation is that engineering excellence is becoming less important than it used to be; that is, it is still a hygiene factor that can sink the company if you do not deliver on it, but its *relative* importance, compared to the other dimensions, is declining. For example, in most business, tuning the code so it goes from good to great matters less than it used to, because of cloud computing and other technical advances.

This is the reason why we see a trend toward product and matrix teams on the technology versus product scale. Whether a company chooses the product or matrix team pattern depends on the company's culture; for a competitive American company, such as Amazon, the product team makes good sense; for a collaborative Scandinavian company, such as Spotify, the matrix team makes better sense, with a few essential adjustments.

Thank you for reading. I wish you the very best of luck with designing the right engineering organization for your company.

REFERENCES

---

[1] Collaborative Intelligence: Using Teams to Solve Hard Problems. J. Richard Hackman. Retrieved February 15, 2020.

[2] The Little Book on REST Services. Kenneth Lange. Retrieved February 15, 2020.

[3] How We Reorganized Instagram's Engineering Team While Quadrupling Its Size. James Everingham. Retrieved February 15, 2020.

[4] If Your Boss Could Do Your Job, You're More Likely to Be Happy at Work. Benjamin Artz et al. Retrieved February 15, 2020.

[5] Engineering Culture at Airbnb. Mike Curtis. Retrieved February 15, 2020.

[6] Amazon's "two-pizza teams": The ultimate divisional organization. Jason Crawford. Retrieved February 15, 2020.

[7] The Scrum Guide. Retrieved February 15, 2020.

[8] Clean Code: A Handbook of Agile Software Craftsmanship. Robert C. Martin. Retrieved February 15, 2020.

[9] How to be a program manager. Joel Spolsky. Retrieved February 15, 2020.

[10] Why Yammer Believes the Traditional Engineering Organizational Structure is Dead. First Round Review. Retrieved February 15, 2020.

[11] Scaling Agile @ Spotify with Tribes, Squads, Chapters & Guilds. Kniberg et al. Retrieved February 15, 2020.

[12] Creating Great Teams: How Self-Selection Lets People Excel. Sandy Mamoli and David Mole. Retrieved February 15, 2020.

How to Structure Your Engineering Teams

The team structure you pick for your engineering organization will have a massive impact on its effectiveness and productivity. Research from Harvard University shows that team structure is the single most important factor in determining team performance.

But how do you pick the right team structure for your engineering organization? The short answer is that it depends on what is important to your company: is it speed to market, technical excellence, or something else?

This little book provides an overview of the most popular team structures and shows their strengths and weaknesses using real-world examples from Instagram, Airbnb, Yammer, Spotify, and other well-known companies.

About the Author

Kenneth Lange is Chief Technology Officer at Tia Technology where he leads the development of the most popular digital insurance platform in Europe.

You can reach him on Twitter (@KennethLange) where he tweets about technology leadership and related topics.