Summary of Team Topologies

Organizing business and technology teams for fast flow

A book by Matthew Skelton and Manuel Pais

PART I - Teams as the Means of Delivery

1 The Problem with Org Charts: Rethink Team Structures, Purpose, and Interactions

"*Organizations should be viewed as complex and adaptive organisms rather than mechanistic and linear system.*"
- Naomi Stanford, Guide to Organisation Design

Developing and operating software effectively for modern, interconnected systems and services requires organizations to consider many different dimensions. Historically, most organizations have seen software development as a kind of manufacturing to be completed by separate individuals arranged into functional specialties, with large projects planned up front and with little consideration for sociotechnical dynamics. This led to the prevailing problems depicted in the picture below.

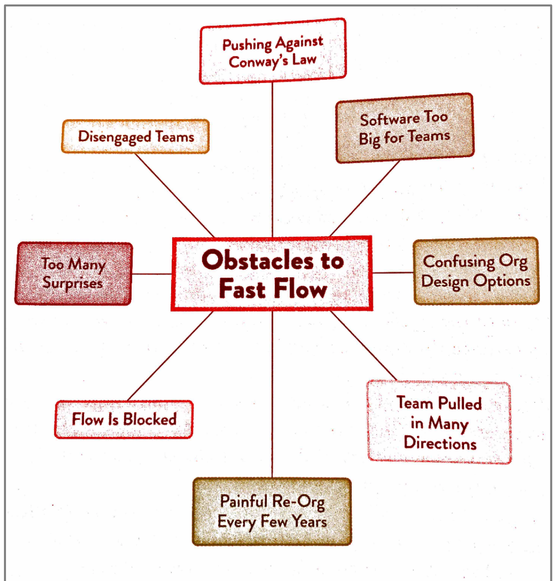
Obstacles to Fast Flow
Pushing Against Conway's Law
Disengaged Teams
Software Too Big for Teams
Too Many Surprises
Confusing Org Design Options
Flow Is Blocked
Team Pulled in Many Directions
Painful Re-Org Every Few Years

Figure 1.2: Obstacles to Fast Flow

The Agile, Lean IT, and DevOps movements helped demonstrate the enormous value of smaller, more autonomous teams that were aligned to the flow of business, developing and releasing in small, iterative cycles, and course correcting based on feedback from users.

Lean IT and DevOps also encouraged big strides in telemetry and metrics tooling for both systems and teams, helping people building and running software to make proactive, early decisions based on past trends, rather than simply responding to incidents and problems as they arose.

However, traditional organizations have often been limited in their ability to fully reap the benefits of Agile, Lean IT, and DevOps due to their organizational models. It's no surprise that there is a strong focus on the more immediate automation and tooling adoption, while cultural and organizational changes are haphazardly addressed. The latter changes are much harder to visualize, let alone to measure their effectiveness. Yet having the right team structure, approach, and interaction in place, and understanding their need to evolve over time is a key differentiator for success in the long run.

In particular, traditional org charts are out of sync with this new reality of frequent (re)shaping of teams for collaborative knowledge work in environments filled with uncertainty and novelty. Instead, we need to take advantage of Conway's law (organizational design prevails over software architecture design), cognitive load restrictions, and a team-first approach in order to design teams with clear purposes and promote team interactions that prioritize flow of software delivery and strategic adaptability.

The goal of Team Topologies is to give you the approach and mental tools to enable your organization to adapt and dynamically find the places and timing when collaboration is needed, as well as when it is best to focus on execution and reduce communication overhead.

KEY TAKEAWAYS

- Conway's law suggests major gains from designing software architectures and team interactions together, since they are similar forces.
- Team Topologies clarifies team purpose and responsibilities, increasing the effectiveness of their interrelationships.
- Team Topologies takes a humanistic approach to building software systems while setting up organizations for strategic adaptability.

2 Conway's Law and Why It Matters: Conway's Law Is Critical for Efficient Team Design in Tech

“[Conway's law] creates an imperative to keep asking: Is there a better design that is not available to us because of our organization?"
- Mel Conway, Toward Simplifying Application Development, in a Dozen Lessons

Conway's law tells us that an organization's structure and the actual communication paths between teams persevere in the resulting architecture of the systems built. They void the attempts of designing software as a separate activity from the design of the teams themselves.

The effects of this simple law are far reaching. On one hand, the organization's design limits the number of possible solutions for a given system's architecture. On the other hand, the speed of software delivery is strongly affected by how many team dependencies the organization design instills.

Fast flow requires restricting communication between teams. Team collaboration is important for gray areas of development, where discovery and expertise is needed to make progress. But in areas where execution prevails - not discovery - communication becomes an unnecessary overhead.

One key approach to achieving the software architecture (and associated benefits like speed of delivery or time to recover from failure) is to apply the reverse Conway maneuver: designing teams to match the desired architecture. The authors provided a simple example (see picture below) where an organization could avoid a monolithic database by embedding database skills in the application team, so that they had sufficient autonomy to maintain a separate data store (perhaps relying on a centralized DBA team for recommendations on database design or synchronization with other databases).

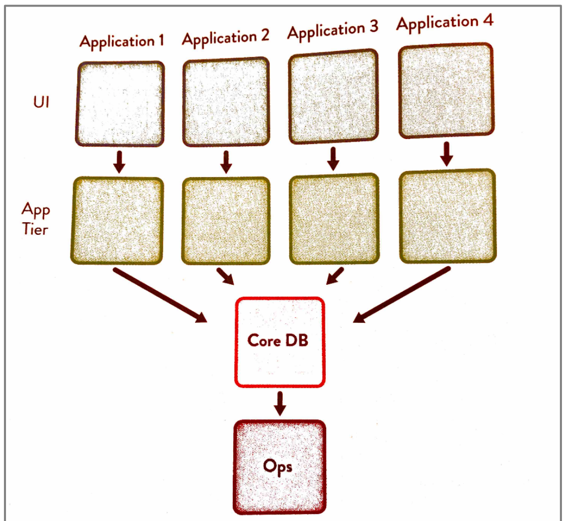

Figure 2.2: Software Architecture from Four-Team Organization
Four separate applications, each with a separate user interface (UI) and a back-end application tier that communicate with a single shared database. This reflects and matches the team communication architecture from Figure 2.1; the diagram has simply been rotated ninety degrees.

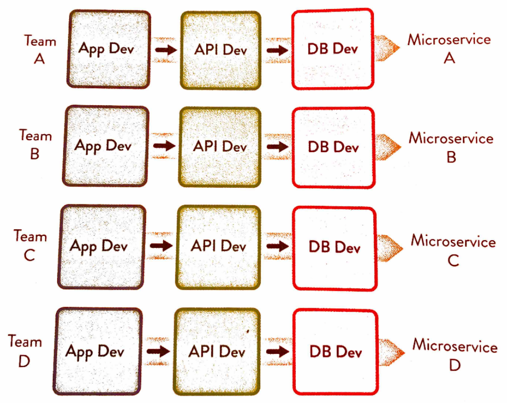

Figure 2.4: Team Design for Microservices Architecture with Independent Services and Data Stores
An organization design that anticipates the homomorphic force behind Conway's law to help produce a software architecture with four independent microservices. (Again, this is basically the diagram in Figure 2.3 rotated ninety degrees.)

In short, by considering the impact of Conway's law when designing software architectures and/or reorganizing team structures, you will be able to take advantage of the isomorphic force at play, which converges the software architecture and the team design.

KEY TAKEAWAYS

- Organizations are constrained to produce designs that reflect communication paths.
- The design of the organization constrains the "solution search space," limiting possible software designs.
- Requiring everyone to communicate with everyone else is a recipe for a mess.
- Choose software architectures that encourage team-scoped flow.
- Limiting communication paths to well-defined team interactions pro- duces modular, decoupled systems.

3 Team-First Thinking: Limit Teams' Cognitive Load and Facilitate Team Interactions to Go Faster

Disbanding high-performing teams is worse than vandalism: it is corporate psychopathy.
- Allan Kelly, Project Myopia

In a fast-changing and challenging context, teams are more effective than groups of individuals. Successful organizations - from the US military to corporations large and small - treat the team as the fundamental means of getting work done. Teams are generally small, stable, and long lived, allowing team members the time and space to develop their working patterns and team dynamics.

Importantly, due to limits on team size (Dunbar's number), there is an effective upper limit on the cognitive load that a single team can bear. This strongly suggests a limit on the size of the software systems and complexity of team should work with. The team needs to own the system domains or subsystems they are responsible for. Teams working on multiple codebases lack ownership and, especially, the mental space to understand and keep the corresponding systems healthy.

The team-first approach provides opportunities for many kinds of ple to thrive in an organization. Instead of needing a thick skin or resilience in order to survive in an organization that atomizes individuals, people in a team-first organization have the space and support to develop their skills and practices within the context of a team.

Crucially, because communication between individuals is de-emphasized in favor of communication between teams for day-to-day work, the organization supports a wide range of communication preferences, from those people who communicate best one to one to those who like large group conversations. Furthermore, the effect of previously destructive individuals is curtailed. This humanistic approach is a huge benefit of choosing teams first.

KEY TAKEAWAYS
- The team is the most effective means of software delivery, not individuals.
- Limit the size of multi-team groupings within the organization based on Dunbar's number.
- Restrict team responsibilities to match the maximum team cognitive load.
- Establish clear boundaries of responsibility for teams.
- Change the team working environment to help teams succeed.

PART II - Team Topologies that Work for Flow

4 Static Team Topologies: Adopt and Evolve Team Topologies that Match Your Current Context

“*Instead of structuring teams according to technical know-how or activities, organize teams according to business domain areas.*”

-Jutta Eckstein, "Feature Teams-Distributed and Dispersed," in Agility Across Time and Space

Setting up new team structures and responsibilities reactively, triggered by the need to scale a product, adopt new technologies, or respond to new market demands, can help in the present moment but often fails to achieve the speed and efficiency of well thought-out topologies.

Because those decisions are often made on an individual team basis, they lack consideration for important organization-wide factors, like technical and cultural maturity, organization size, scale of the software, engineering disciple, or inter-team dependencies. The result is team structures optimized for problems that are temporary or limited in scope, rather than adaptive to new problems over time.

The "DevOps team" anti-pattern is a quintessential example. On paper, it makes sense to bring automation and tooling experts in house to accelerate the delivery and operations of our software. However, this team can quickly become a hard dependency for application teams if the DevOps team is being asked to execute steps on the delivery path of every application, rather th helping to design and build self-service capabilities that application teams can rely on autonomously.

It is critical to explicitly consider the different aspects at play and adopt topologies that work given the organizational context (which tends to evolve slowly), rather than adapting those that solve a particular problem or need in a given moment in time.

In particular, within a DevOps context the DevOps Topologies can help shed some light on which topologies work well for which contexts. Forward- thinking organizations take a multi-stage approach to their team design, understanding that what works best today might not necessarily be the case in a few years, or even months from now.

KEY TAKEAWAYS

- Ad hoc or constantly changing team design slows down software delivery.
- There is no single definitive team topology but several inadequate topologies for any one organization.
- Technical and cultural maturity, org scale, and engineering discipline are critical aspects when considering which topology to adopt.
- In particular, the feature-team/product-team pattern is powerful but only works with a supportive surrounding environment.
- Splitting a team's responsibilities can break down silos and empower other teams.

5 The Four Fundamental Team Topologies: Use Loosely Coupled, Modular Groups of Four Specific Team Types

"*The architecture of the system gets cemented in the forms of the teams that develop it.*"
- Ruth Malan, "Conway's Law"

Many organizations struggling with rapid, sustainable software delivery have a wide range of different types of teams, each with different (usually poorly defined) responsibilities. To avoid this problem, restrict teams to just four fundamental types:

- stream aligned,
- enabling,
- complicated subsystem,
- and platform.

This focuses the organization on team interaction patterns that are known to promote flow at both personal and organizational levels.

Organizations developing and running non-trivial software systems today need to optimize their teams for a safe and rapid flow of change strongly informed by how live production systems work (or fail). This means that the majority of teams need to be loosely coupled, aligned to the flow of change (the "stream"), and capable of delivering a useful increment in the product, service, or user experience for which they are responsible.

Helping **stream-aligned teams** achieve this high rate of flow are:

- **enabling teams** (which identify impediments and cross-team challenges, and simplify the adoption of new approaches),
- **complicated-subsystem teams** (if needed, to bring deep specialist expertise to specific parts of the system),
- and **platform teams** (which provide the underlying "substrate" on which stream-aligned teams can build and support software products and services with minimal friction).

This standardization on the types and responsibilities of teams building and running software systems helps to increase flow by ensuring that most teams are stream aligned, with supporting capabilities and skills provided by enabling, complicated-subsystem, and platform teams.

In turn, the platform itself is run as a product or service, with well-established software-product-management techniques used to prioritize work, regular interaction with customers of the platform (mostly stream- aligned teams), and a strong focus on UX and DevEx. The platform itself may be composed of internal stream-aligned teams, enabling teams, complicated-subsystem teams, and even lower-level platform teams, using the same team types and interactions that are used by the teams consuming the platform.

The focus on empowering stream-aligned teams to achieve fast flow helps to drive decisions at all levels of the organization and provides the overarching mission for all teams.

KEY TAKEAWAYS

- The four fundamental team topologies simplify modern software team interactions.
- Mapping common industry team types to the fundamental topologies sets up organizations for success, removing gray areas of ownership and overloaded/underloaded teams.
- The main topology is (business) stream-aligned; all other topologies support this type.
- The other topologies are enabling, complicated-subsystems, and platform.
- The topologies are often "fractal" (self-similar) at large scale: teams of teams.

6 Choose Team-First Boundaries: Choose Software Boundaries to Match Team Cognitive Load

“*When code doesn't work... the problem starts in how teams are organized and [how] people interact*.”
 - Eric Evans, Domain-Driven Design

When optimizing for flow, stream-aligned teams should be responsible for a single domain. This is a challenge when domains are hidden in monolithic systems that include many different responsibilities and are mostly driven by technology choices, providing functionalities across multiple areas of business.

We need to look for natural ways to break down the system (fracture planes) that allow the resulting parts to evolve as independently as possible.
Consequently, teams assigned to those parts will experience more autonomy and ownership over them.

Looking to align subsystem boundaries with (mostly independent) segments of the business is a great approach, and the domain-driven design methodology supports that approach nicely. But we need to beware and sense for other fracture planes, such as change cadence, risk, regulatory compliance, and so on. Often, a combination of fracture planes will be required.

Finally, we need to be aware of different types of monoliths impeding flow and causing unnecessary dependencies between teams. While we typically think of system architecture as a monolith, there are other, more subtle ways in which coupling creeps in, even when the system architecture is already modular (for example, shared databases, coupled builds and/or releases, and more). As Amy Phillips puts it, "If you have microservices but you wait and do end-to-end testing of a combination of them before a release, what you have is a distributed monolith."

When considering subsystem boundaries, the main aim should be to find software fracture planes that align to business domain bounded contexts, because most of these bounded contexts will map to streams of change that are natural for the organization. This, in turn, means that business-domain boundaries can be aligned to stream-aligned teams, helping to focus on flow across the organization.

Fracture planes can be chosen around specific challenges, e.g.:

- Technology
- Regulation
- Performance
- Geographic location of staff
- User personas

The fracture planes help avoid hand-offs between teams and promote flow.

In all cases, it is essential to make software segments team sized so that teams can effectively own and evolve their software in a sustainable way.

KEY TAKEAWAYS

- Choose software boundaries using a team-first approach.
- Beware of hidden monoliths and coupling in the software-delivery chain.
- Use software boundaries defined by business-domain bounded contexts.
- Consider alternative software boundaries when necessary and suitable.

PART III Evolving Team Interactions for Innovation and Rapid Delivery

7 Team Interaction Modes: Three Well-Defined Team Interaction Modes

"*Technologies and organizations should be redesigned to intermittently isolate people from each other's work for best collective performance in solving complex problems.*"
- Ethan Bernstein, Jesse Shore, and David Lazer, "How Intermittent Breaks in Interaction Improve Collective Intelligence."

An effective, modern organization building and running software is a product of the interactions between teams. Yet many organizations fail to define what good team interactions look like, resulting in confusion, annoyance, and ineffectiveness. Simply defining a set of teams with responsibility boundaries is not enough to produce an effective sociotechnical system; it is also necessary to define sensible and effective interactions between teams.

In this chapter, the authors have shown how three core team interaction modes provide the clarity needed for all team interactions within the organization:

- Collaboration: two teams work closely together for a defined period to discover new patterns, approaches, and limitations. Responsibility is shared and boundaries blurred, but problems are solved rapidly and the organization learns quickly.
- X-as-a-Service: one team consumes something (such as a service or an API) provided "as a service" from another team. Responsibilities are clearly delineated and if the boundary is effective, the consuming team can deliver rapidly. The team providing the service seeks to make their service as easy to consume as possible.
- Facilitating: one team helps another team to learn or adopt new approaches for a defined period of time. The team providing the facilitation aims to make the other team self-sufficient as soon as possible, while the team receiving the facilitation has an open-minded attitude to learning.

The combination of well-defined team types and well-defined team interactions provides a clear and powerful way to promote team-based organizational effectiveness, avoiding the ambiguities and conflicts that many organizations experience.

KEY TAKEAWAYS

- Choose specific team interaction modes to enhance software delivery.
- Choose between three team interaction modes to help teams provide and evolve services to other teams:
  - **Collaboration** can be a powerful driver for innovation but can also reduce flow.
  - **X-as-a-Service** can help other teams deliver quickly but only if the boundary is suitable.
  - **Facilitating** helps to avoid cross-team challenges and detects problems.

8 Evolve Team Structures with Organizational Sensing

“*The design...is almost never the best possible, [so] the prevailing system concept may need to change. Therefore, flexibility of organization is important to effective design.*”
- Mel Conway, "How Do Committees Invent?"

The rapid pace of change in technology, markets, customer and user demands, and regulatory requirements means successful organizations need to expect to adapt and evolve their organization structure on a regular basis. However, organizations that build and run software systems need to ensure that their team interactions optimize for flow, Conway's law, and a team-first approach (including team cognitive load). By deploying the four fundamental team topologies with the three core team interaction modes, organizations gain crucial clarity of purpose for their teams on an ongoing basis. Teams understand:

- how, when, and why they need to collaborate with other teams;
- how, when, and why they should be consuming or providing something “as a service”;
- and how, when, and why they should provide or seek facilitation with another team.

Thus, an organization should expect to see different kinds of interactions between different kinds of teams at any given time as the organization responds to new challenges.

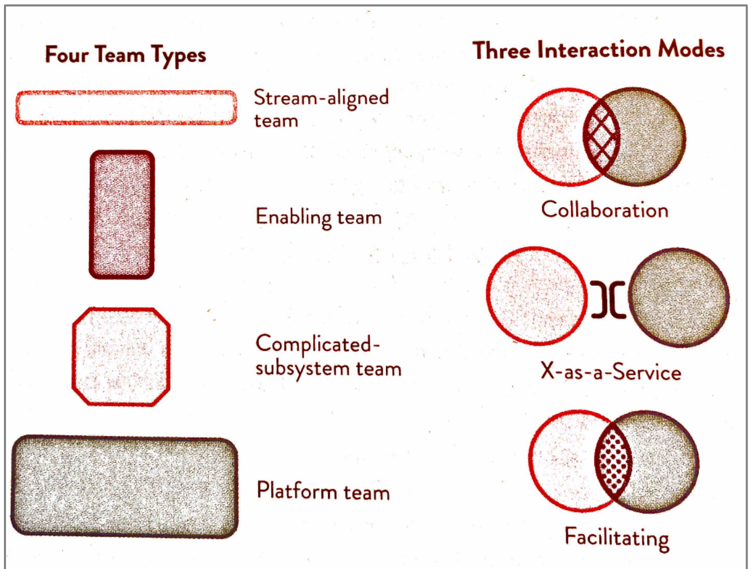

Figure 0.1: The Four Team Types and Three Interaction Modes

The combination of well-defined teams and well-defined interaction modes provides a powerful and flexible organizational capability for structural adaptation to changing external and internal conditions, enabling the organization to "sense" its environment, modify its activities, and focus to fit.

KEY TAKEAWAYS

- Use different team topologies simultaneously for strategic advantage.
- Change team topologies and team interactions to accelerate adoption of new approaches.
- Differentiate between explore, exploit, sustain, retire phases using team topologies.
- Expect multiple, simultaneous team topologies to meet different needs. Recognize triggers for organization change.
- Treat operations as high-fidelity sensory input for self-steering