A coaching field guide to

Team Topologies

Four team types. Three interaction modes. One organising principle: design your teams for a fast flow of change, and the architecture will follow.

> Based on **Team Topologies: Organizing Business and Technology Teams for Fast Flow** — Matthew Skelton & Manuel Pais (IT Revolution, 2019). Diagnostics and coaching prompts are added for facilitation use.

> *"Organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations."*
>
> MELVIN CONWAY, 1968

The team is the unit

Not the individual, not the project. Long-lived, stable teams own their software. Work is given to a team, never to a person.

Cognitive load is finite

A team can only hold so much. Software boundaries must be drawn to fit inside what the team can actually reason about.

Conway's Law runs both ways

Your org chart already shapes your architecture. Reverse the manoeuvre: design the teams you need for the architecture you want.

THE FOUR FUNDAMENTAL TEAM TYPES

Stream-aligned

Owns a single valuable stream of change, end to end. The default team type — most teams should be one.

Enabling

Specialists who raise the capability of stream-aligned teams, then deliberately leave.

Complicated-Subsystem

Deep specialist knowledge for one hard component. Rare, and only when the maths demands it.

Platform

An internal product of self-service capabilities that shrinks the load on everyone above it.

THE THREE INTERACTION MODES

Collaboration

Two teams work closely for a bounded period to discover something new. High bandwidth, high cost.

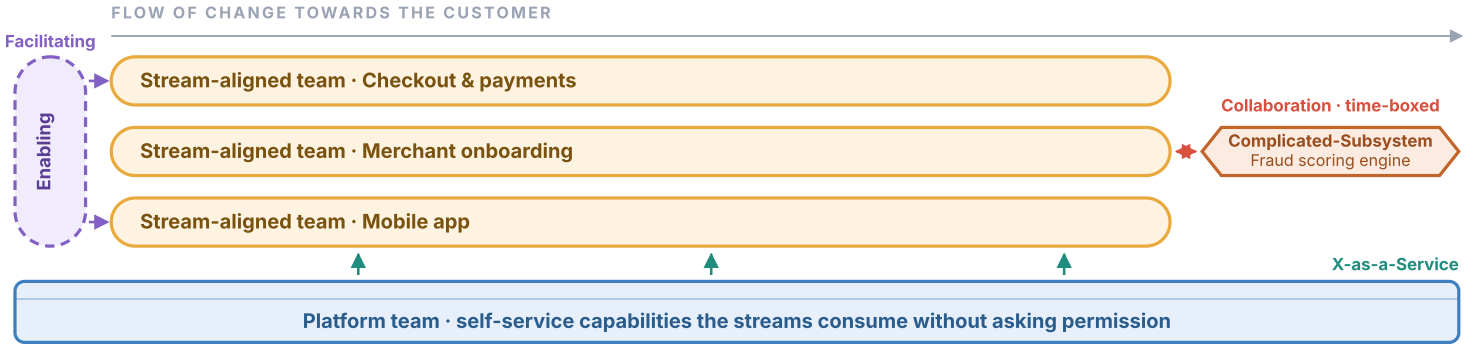
X-as-a-Service

One team consumes what another provides, with minimal talking. Predictable, clear ownership.

Facilitating

One team helps another learn or unblock. Time-boxed, and success means it ends.

ALL OF IT IN ONE PICTURE



The four fundamental team types

Skelton and Pais argue that almost every team in a technology organisation can — and should — be one of these four. Anything that is none of them is usually a silo, a hand-off, or a queue in disguise.

Stream-aligned

THE DEFAULT

MOST TEAMS

PURPOSE

Aligned to a single, valuable stream of work — a product, a service, a user journey, a persona, a market segment. It builds, ships and runs that stream itself, with the smallest possible number of hand-offs.

WHAT GOOD LOOKS LIKE

- Can go from idea to production without waiting on another team
- Owns the thing in production, including the pager
- Has slack capacity to improve, not just to deliver

SIZING

- 5–9 people, long-lived
- One complicated domain, **or** 2–3 simple ones

Anti-pattern: a "feature team" that needs three other teams' approval to deploy. That is a stream-*shaped* team on a hand-off-shaped org.

Enabling

CAPABILITY, NOT CAPACITY

A FEW

PURPOSE

A small band of specialists — testing, build engineering, architecture, security — who help stream-aligned teams over an obstacle they cannot clear alone. They increase autonomy by growing the other team's capability, not by doing the work for them.

WHAT GOOD LOOKS LIKE

- Engages for weeks or a few months, then moves on
- Spots gaps the stream teams have not noticed
- Measures success by what the team can now do alone

SIZING

- Small; serves several stream teams
- Engagement is always time-boxed

Anti-pattern: the permanent Centre of Excellence. An enabling team that never leaves becomes an ivory tower — and then a gate.

Complicated-Subsystem

WHERE DEEP EXPERTISE IS UNAVOIDABLE

RARE

PURPOSE

Owns a part of the system that genuinely requires specialist knowledge to change safely — a video codec, a pricing or risk model, a real-time matching engine, a face-recognition core. It exists so that stream-aligned teams do **not** have to carry that load.

WHAT GOOD LOOKS LIKE

- You could not staff it from the general engineering pool
- Serves consumers through a stable, documented interface
- Its existence measurably reduces load elsewhere

THE BAR TO CLEAR

- Complicated ≠ merely legacy or messy
- Justify it, or fold it back into a stream

Anti-pattern: relabelling an existing component silo. If the reason is "nobody else understands it", that is a documentation problem, not a topology.

Platform

AN INTERNAL PRODUCT

AS THIN AS POSSIBLE

PURPOSE

Provides internal services that stream-aligned teams consume mostly self-service, so they can deliver with substantial autonomy. The platform is a **product** with internal customers who could, in principle, go elsewhere.

WHAT GOOD LOOKS LIKE

- Adopted because it is better, never because it is mandated
- Has a roadmap, documentation, versioning and support
- Treats developer experience as the primary metric

TWO KEY IDEAS

- **Thinnest Viable Platform** — the smallest thing that helps
- A large platform is itself a fractal: it contains its own four team types

Anti-pattern: the mandated cost-centre platform. If teams have to use it and nobody asks whether they want to, you have built an internal monopoly.

The three interaction modes

Team types tell you who exists. Interaction modes tell you how they are allowed to touch each other — and this is where most organisations quietly leak their capacity.

CHAPTERS 6–7

Collaboration

Two teams, one problem, for now

USE WHEN

You are in genuine discovery. The interfaces are unknown, the technology is new, or the two teams must invent something neither can specify yet.

YOU ARE PAYING FOR IT WITH

- Higher cognitive load on *both* teams
- Blurred responsibility boundaries
- A short-term drop in delivery speed

COACH'S RULE

Collaboration is expensive and must be deliberate and temporary. Name an end date at the start, and name what you expect to have learned by it.

X-as-a-Service

Consume without conversation

USE WHEN

The thing being provided is well understood and you need predictable delivery. One team consumes an API, a component or a platform capability provided by another.

REQUIRES

- A clear, versioned, documented boundary
- Real ownership on the provider side
- Provider treating consumers as customers

COACH'S RULE

This is the mode you should be converging towards for most pairs of teams. If every interaction still needs a meeting, the service boundary is wrong.

Facilitating

Help them, then go

USE WHEN

One team needs to learn something, adopt a practice, or clear an impediment it cannot see around. The classic mode of an enabling team.

LOOKS LIKE

- Coaching, pairing, mentoring — not delivering
- Explicitly time-boxed engagement
- Gaps discovered and fed back to the wider org

COACH'S RULE

The measure of a facilitating engagement is what the receiving team can do *after* you leave. If they still need you, you have not finished — or you never started.

Which mode should each team type expect?

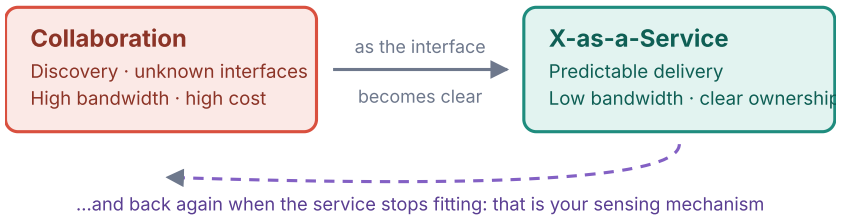
A team's type strongly predicts how it should interact. Deviations are not forbidden — they are a signal to investigate.

TEAM TYPE	COLLABORATION	X-AS-A-SERVICE	FACILITATING
Stream-aligned	Typical	Typical	Occasional
Enabling	Occasional	Rare	Typical
Complicated-Subsystem	Occasional	Typical	Rare
Platform	Occasional	Typical	Rare

Read the table as a diagnostic, not a rulebook. A platform team stuck permanently in Collaboration with its consumers is telling you its service boundary has not been found yet.

Modes are a sequence, not a setting

The healthy pattern is a deliberate move from discovery to service as understanding hardens.



Long-lived Collaboration between the same two teams is one of the strongest signals in the book that a boundary is drawn in the wrong place.

Cognitive load — the constraint everything else serves

Team Topologies borrows John Sweller's three types of cognitive load and turns them into an organisational design tool. If you only take one idea from the book into your coaching practice, take this one.

Three loads, three different responses

The mistake is treating all mental effort as the same thing. Each type has its own lever.

Intrinsic — REDUCE

Effort intrinsic to the problem space — the language, the framework, the fundamentals of the domain. **Lever:** training, pairing, good hiring, better technology choices.

Extraneous — ELIMINATE

Effort spent on the mechanics of the environment — how to deploy, which config flag, which ticket queue. It adds no value at all. **Lever:** automation and platform. This is what a platform is *for*.

Germane — PROTECT

Effort spent on the business domain itself — the part that produces value. **Lever:** clear it a path. Every hour of extraneous load you remove is an hour that can go here.

TYPICAL TEAM, TODAY



SAME TEAM, ON A GOOD PLATFORM



Intrinsic · Extraneous · Germane — the value-producing part

Total capacity does not grow. You are only ever choosing what it gets spent on.

A load audit you can run in twenty minutes

List every domain each team owns, mark it simple / complicated / complex, and read the result out loud.

TEAM	DOMAINS IT ACTUALLY OWNS	READ
Payments	Checkout *(complicated)* · Refunds *(complicated)* · Invoicing *(simple)*	Over the limit
Mobile app	iOS & Android client *(complicated)* · Feature flags *(simple)*	Within limits
Data services	Warehouse *(complicated)* · Reporting *(complicated)* · Ingest *(complicated)*	Split it

Nobody asked the Payments team to take on three domains. They accumulated, one reasonable-sounding ticket at a time.

Heuristics you can use in the room

1. Give a single team **one complicated or complex domain** — or two to three simple ones. Never two complicated ones.
2. If a team owns two complicated domains, split the team or split the domains. Adding people to the same team does not work: communication overhead grows faster than capacity.
3. Ask the team to size the domain themselves, in relative terms. Their answer is more reliable than your estimate.
4. When load is too high, the first move is to **take responsibilities away**, not to add headcount.
5. Design software boundaries to fit the load a team can hold — not the other way round.

Team size, and why it is not arbitrary

Trust does not scale linearly, and Dunbar's limits show up in organisations whether or not you design for them.

~5	~15	~50	~150
deep mutual trust	trust and memory of most	natural grouping boundary	limit of personal-history trust

Practical implication: keep teams at **5–9 people**, group them into tribes or divisions that respect the ~50 and ~150 thresholds, and expect to restructure ownership when you cross one. Crossing a Dunbar boundary without changing the topology is how organisations acquire silos they did not choose.

A note for coaches: cognitive-load conversations land far better than org-chart conversations. Start there.

Drawing the boundaries — fracture planes & the Team API

Two practical instruments: where to cut the system, and how a team makes itself consumable by the rest of the organisation.

CHAPTERS 4 & 3

Fracture planes — the natural seams along which to split a system

A fracture plane is a place where the system splits cleanly. Business-domain bounded context is the primary one; the rest are supporting seams to test against.

Business domain
The dominant plane. Split along bounded contexts, in the Domain-Driven Design sense.

Regulatory compliance
Isolate what is audited — PCI, HIPAA, financial reporting — so its constraints do not spread.

Change cadence
Things that change weekly should not be welded to things that change yearly.

Team location
Split along geography and time zones rather than pretending distance is free.

Risk
Separate high-risk from low-risk change so the safe path is not slowed to the pace of the dangerous one.

Performance isolation
Carve out components with sharply different load profiles or scaling needs.

Technology
A valid seam, but a weak one. Use it last — it is the plane that most often produces silos.

User persona
Split by who you serve — enterprise vs. self-serve, buyer vs. operator.

Coaching move: run these eight as a card sort against your current architecture. The planes that produce the same split are your strong candidates; a plane that cuts across every existing team boundary is telling you something.

The Team API

Everything a team deliberately exposes so others can work with it without needing a meeting.

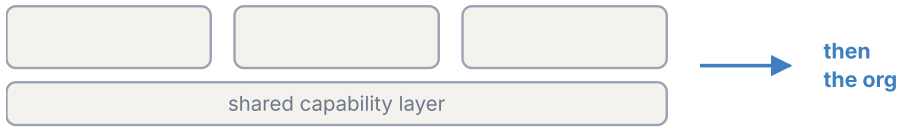
- **Code** — Runtime endpoints, libraries, clients the team produces
- **Versioning** — How change is signalled and what stability others can rely on
- **Wiki & documentation** — Especially how-to guides for the things the team owns
- **Principles & practices** — The team's preferred ways of working, made visible
- **Communication** — Which channels, what response expectations, who to ask
- **Work information** — What we are doing now, what is next, what we will not do
- **Purpose** — The stream, subsystem or platform this team exists to serve
- **Expected interaction modes** — Who we collaborate with, who we serve, who we facilitate

Designing the Team API forces a team to answer the question others are already asking behind its back: *what are you for, and how do I get something from you without interrupting you?*

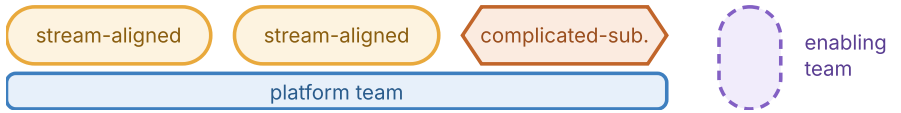
The Reverse Conway Manoeuvre

Conway's Law is not a warning. Treated deliberately, it is a design tool.

STEP 1 — DECIDE THE ARCHITECTURE YOU WANT



STEP 2 — BUILD THE TEAMS THAT WOULD PRODUCE IT



You cannot get a modular architecture out of an organisation that communicates as one undifferentiated blob, and you cannot get an integrated product out of teams that never speak. Choose the communication structure first; the software will comply.

Using this with a real organisation

A facilitation sequence, the anti-patterns you will meet on the way, and the questions that tend to open the conversation up.

COACHING TOOLKIT

A five-step sequence for a topology workshop

01
Map the streams
What are the actual valuable flows of change through this organisation? Not the projects — the streams that keep existing.

02
Measure the load
For each team, list the domains it owns and have the team size them. Find who is carrying two complicated domains.

03
Draw the real interactions
Whiteboard who talks to whom, and how often. Mark each edge as Collaboration, XaaS or Facilitating.

04
Name the target topology
Assign each team one of the four types. Argue about the ones that fit none — those are your findings.

05
Change one thing
Pick a single boundary and move it. Re-run in a quarter. Topologies evolve; big-bang reorgs destroy the trust they need.

Anti-patterns you will actually encounter

Most are not failures of intent. They are what happens when structure is allowed to accumulate rather than be designed.

- ✕ **Ad hoc team design**
  Teams created by whoever was available, split when someone left. Nobody owns the topology, so nobody can fix it.
- ✕ **More communication is always better**
  It is not. Unnecessary communication is a tax and a signal that a boundary is missing.
- ✕ **The mandated platform**
  Adoption enforced by policy instead of earned by quality. You will never learn whether it is any good.
- ✕ **Complicated-subsystem inflation**
  Every awkward component declared “specialist”. Usually a documentation and ownership problem wearing a topology costume.
- ✕ **Shuffling team members**
  Reassigning people between teams each quarter. Trust takes months to build and one reorg to destroy.
- ✕ **Splitting by function**
  Frontend, backend, DBA, QA, ops. Every unit of value now needs four queues to reach a customer.
- ✕ **The permanent enabling team**
  Facilitating that never ends becomes a dependency, then a gate, then an ivory tower.
- ✕ **The team that cannot deploy**
  Stream-aligned in name, blocked in practice. Check what a team can do without permission, not what it is called.

QUESTIONS THAT OPEN THE CONVERSATION

- Walk me through the last change you shipped. Who had to say yes?
- What do you own that you wish you did not?
- Which other team do you talk to most? Is that by design?
- If this team disappeared tomorrow, what would stop?
- What would you have to learn to stop depending on them?

SIGNALS IT IS TIME TO CHANGE THE TOPOLOGY

- Delivery slows although nothing technical has changed
- The same two teams have been “collaborating” for a year
- A team's domain list has quietly grown and nothing came off it
- You crossed roughly 50 or 150 people without redesigning ownership
- New starters take longer and longer to ship anything

“Organisation design is a continuous activity, not a one-off event. The goal is not the perfect topology — it is an organisation that can sense when its topology no longer fits, and change it without a crisis.”

A PARAPHRASE OF THE BOOK'S CLOSING ARGUMENT — USE IT TO END THE WORKSHOP