Team Topologies by Matthew Skelton and Manuel Pais

Contents

Short Summary

My top 8 takeaways and a 'so what' are below.

Top 8 Takeaways

1. Conway's law states that organisation structures need to align to architecture or…. the architecture will end up aligning to the organisation structure.
2. Dunbar's number says that: (1) 5 people is the number of people you can hold a close relationship, (2) 15 people is the number of people you can have deep trust with, (3) 50 people is the number of people you can have mutual trust with and (4) 150 people is the number of people for whom you can remember their capabilities.
3. Brooke's law states that adding more people to a project doesn't immediately increase the output. This is because of legacy colleagues needing to teach new colleagues, learning how to work together emotionally (forming, storming, norming, performing)
4. In high trust organisations, reorganisations could happen annually, in low trust organisations, reorganisations shouldn't happen more than every 18 months to 2 years.
5. Teams should not be given more than their cognitive load can cope. Cognitive load is (1) intrinsic (e.g. programming language), (2) extraneous (e.g. how to set up and run unit tests), (3) germane (e.g. banking domain knowledge).
6. There are four fundamental team topologies: (1) Complicated sub system team - delivering for a complicated sub-system until it has stabilised, (2) Enabling team - supports the stream aligned teams, (3) Stream aligned team - delivering a steady flow of features and (4) Platform team - aligned to a platform (e.g. cloud or IOT or channel).
7. There are three types of collaboration (1) Collaboration, (2) X as a service and (3) Facilitating. The four team topologies, collaborate using these three types of collaboration.
8. Next steps for getting started with team topologies:
    a. Start with the team
    b. Identify suitable streams of change
    c. Identify the thinnest viable platform
    d. Identify capability gaps in team coaching, mentoring, service management and documentation
    e. Share and practice different interaction modes and explain principles behind new ways of working

So what?

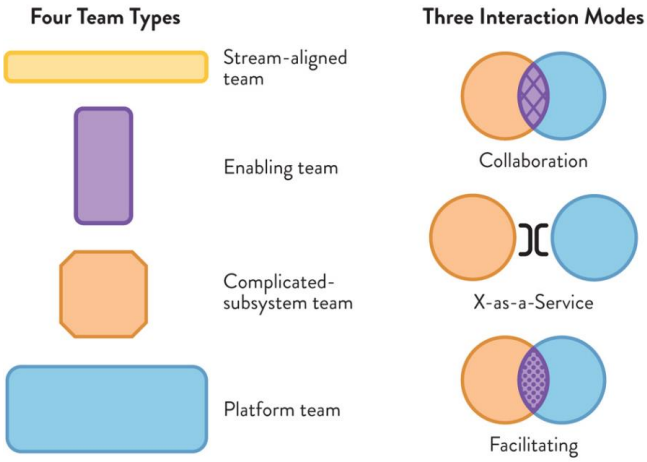
My main so what messages from this book are:

1. Don't change your org more than annually if you are high trust or 18 months if low trust organisation
2. Make sure your org and your architecture are aligned
3. Org charts aren't as helpful as team topologies for explaining how organisations will work and communicate
4. There's a lot in this book so I'd say a key point is to try and keep it simple whilst leveraging some of the key concepts. Don't tie yourself in knots.

Detailed Summary

Contents:

Summary:

- Part 1 Teams as the means of delivery
    - Chapter 1: The problem with org charts
        - Summary:
            - Conway's law suggests major gains from designing software architectures and team interactions together, since they are familiar forces.
            - Team topologies clarify team purpose and responsibilities increasing the effectiveness of their interrelationships.
            - Team topologies take a humanistic approach to building software systems while setting up organisations for strategic adaptability.
        - Key
            - 

        - Org chart with actual lines of communication

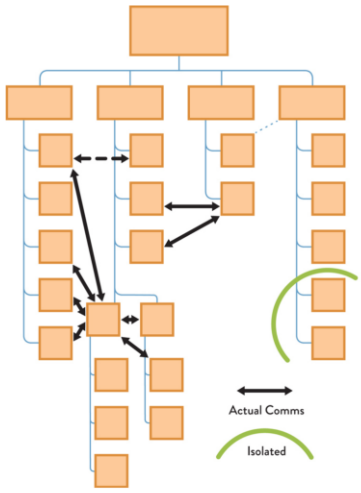


Figure 1.1: Org Chart with Actual Lines of Communication
In practice, people communicate laterally or "horizontally" with people from other reporting lines in order to get work done. This creativity and problem solving needs to be nurtured for the benefit of the organization, not restricted to optimize for top-down/bottom-up communication and reporting.

- Org chart thinking is the problem.
- Thinking beyond the org chart gives you:
  - Formal structure – facilitates compliance.
  - Informal structure – the realm of influence between individuals
  - Value creation structure – how work actually gets done based on interpersonal and inter team reputation
- Key points from organisation design book: Creating high performing and adaptable enterprises.
  - Design when there is a compelling reason.
  - Develop options for deciding on a design.
  - Choose the right time to design.
  - Look for clues that things are out of alignment.
  - Stay alert to the future.
- Obstacles to fast flow:

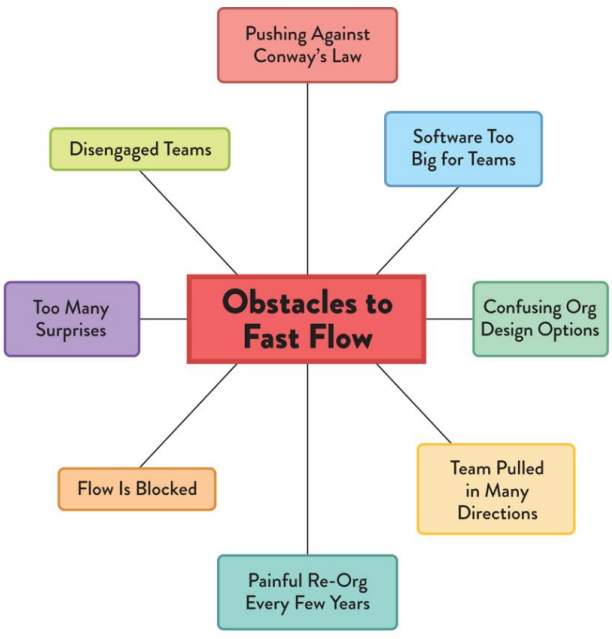


Figure 1.2: Obstacles to Fast Flow

- Chapter 2: Conway's Law and why it matters.
  - Summary:

- Organisations are constrained to produce designs that reflect communication paths.
- The design of the organisation 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 produce modular, decoupled systems

▪ Inter team communication.

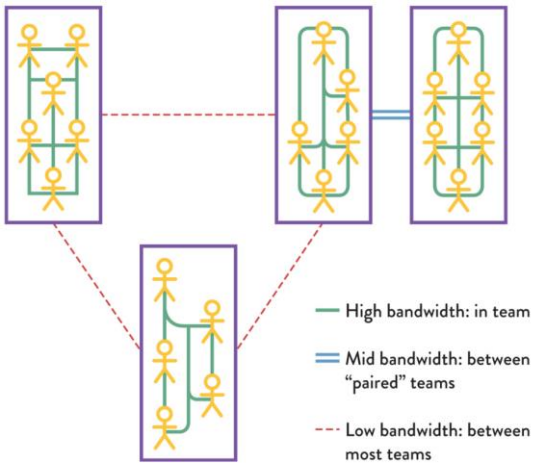


Figure 2.5: Inter-Team Communication
Communication within teams is high bandwidth. Communication between two "paired" teams can be mid bandwidth. Communication between most teams should be low bandwidth.

▪

▪ Conway’s law is about the impact of the software architecture on an organisation.

▪ The below diagram aligns the four teams to the four microservices.

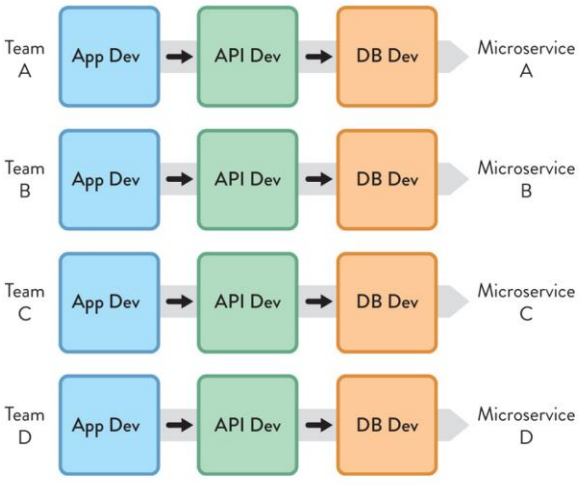


Figure 2.4: Team Design for Microservices Architecture with Independent Services and Data Stores
An organization design that anticipates the homomorphic force behind Conway's law to help produce a software architecture with four independent microservices. (Again, this is basically the diagram in Figure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

▪

- Chapter 3: Team-First thinking
  - Summary
    -
    - The team is the most effective means of software delivery, not individuals.

- Limit the size of multi team groupings within the organisation 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.

- Dunbar's number:
  - Around 5 people – limit of people with whom we can hold close personal relationships and working memory.
  - Around 15 people – limit of people with whom we can experience deep trust.
  - Around 50 people – limit of people with whom we can have mutual trust.
  - Around 150 people – limit of people whose capabilities we can remember.
  - A single team: around five to eight people – in high trust organisations no more than 15 people.
  - Families (tribes) grouping of teams of no more than fifty people – in high trust organisations, groupings of no more than 150 people.
  - Divisions / streams / profit and loss lines: groupings of no more than 150 or 500 people.

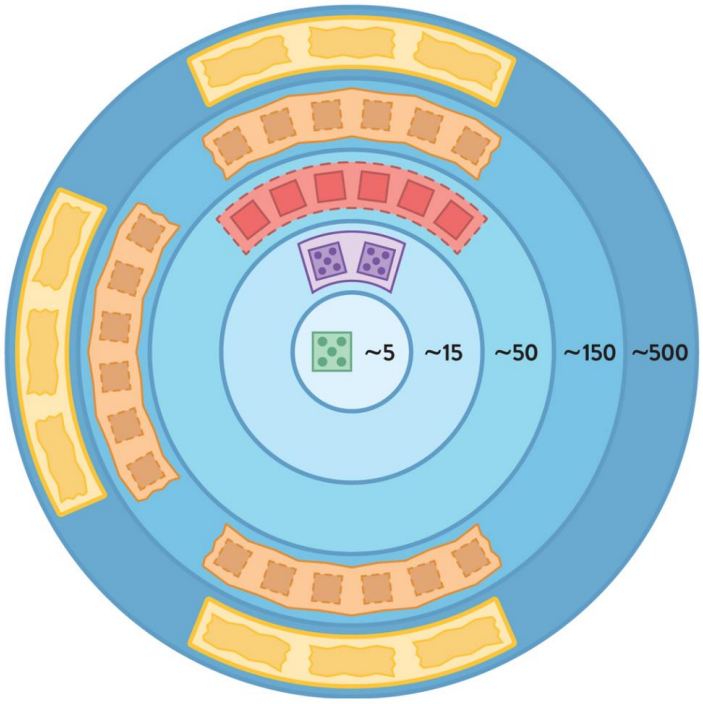


Figure 3.1: Scaling Teams Using Dunbar's Number
Organizational groupings should follow Dunbar's number, beginning with around five people (or eight for software teams), then increasing to around fifteen people, then fifty, then 150, then 500, and so on.

- Teams can take time to form and be effective. Typically, a team can take from two weeks to three months to become a cohesive unit.
- From the 'mythical man month' – adding new people to a team doesn't immediately increase its capacity (as described with Brooke's law). Instead, it reduces capacity in the initial stage.
- The initial slowing of a team when new folks are onboarded is due to:

        - Teaching the new team members which takes up time from the old team members.
        - Emotional adaptation which involves understanding how to accommodate each other's points of view and work habits (storming from the Tuckman model)
        - For these reasons, teams should not be changed regularly but only when required. In a high trust organisation, teams could change once a year without major detrimental effects on team performance. In lower trust organisations people should remain in the same team for eighteen months to two years.
- There are three types of cognitive load
    - Intrinsic cognitive load – relates to aspects of the task fundamental to the problem space
    - Extraneous cognitive load – relates to the environment in which the tasks is being done
    - Germane cognitive load – relates to aspects of the task that need special attention for learning or high performance
    - For example: A web application developer could have knowledge of the computer language and programming plus details of the commands needed to instantiate a dynamic testing environment plus specific aspects of the business domain that the application developer I am programming.
- Software boundary size should be matched to team cognitive load.
- Team members need a team first mindset
    - Be on time to stand-ups and meetings
    - Keep discussions and investigations on track
    - Encourage a focus on team goals
    - Help unblock other team members before starting on new work
    - Mentor new and less experienced folks
    - Avoid 'winning' arguments and instead agree to explore options
- No more than one complicated or complex domain per team

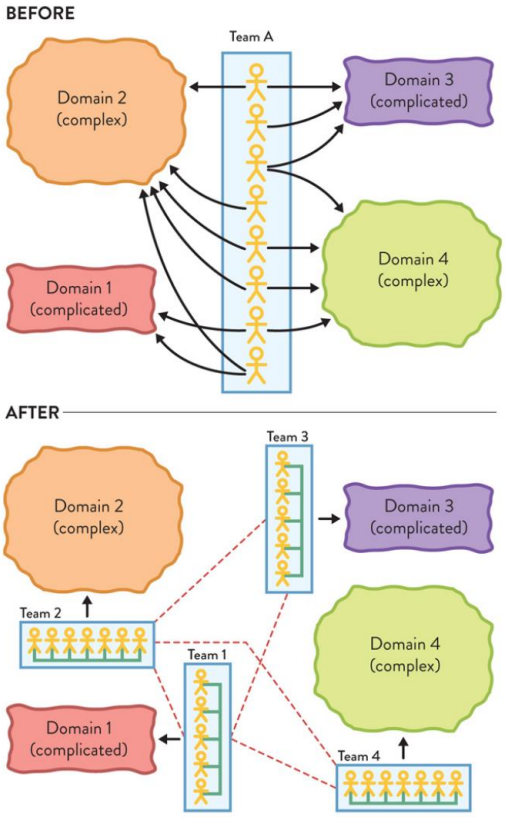
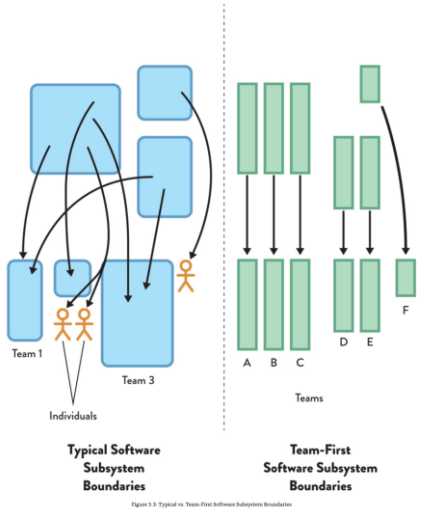


Figure 3.2: No More than One Complicated or Complex Domain per Team

- Typical team's vs team first design



- To increase the size of a software subsystem or domain for which a team is responsible, the cognitive capacity should be lessened by:
    - Provide a team first working environment
    - Minimise team distractions including meetings, emails, set up a dedicated person to field queries
    - Change the management style by communicating goals and outcomes and not the 'how'
    - Increase the quality of developer experience for other teams using your teams code and APIs
    - Use a platform that is explicitly designed to reduce cognitive load for teams building software on top of it

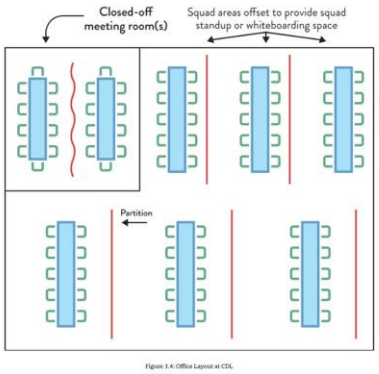
- Define team APIs including:
    - Code: Runtime endpoints, libraries, clients, UI produced by the team
    - Versioning: how the team communicates changes to its code and services
    - Wiki and documentation: especially how to guides for the software owned by the team
    - Practices and principles the teams preferred ways of working
    - Communication – the team's approach to remote communication tools, such as chat tools and video conferencing
    - Work information: what the team is working on now, what's coming next and overall priorities in the short to medium term
    - Other – anything else that the other teams need to use to interact with the team
- Office layout:
    - 

Figure 3.4: Office Layout at CDL

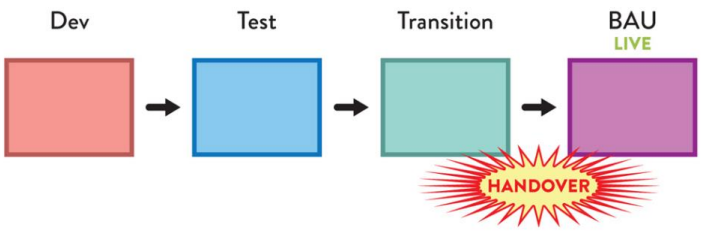
- Part 2 Team Topologies that work for flow
    - Chapter 4: Static team topologies
        - Summary
            - Ad hoc or constantly changing team design slows down software delivery
            - There is no single definitive team topology but several inadequate topologies for any one organisation
            - 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.
        - This is a definition of a traditional flow of change. No information flows back. This is non-ideal topology.
            - 

Figure 4.1: Organization not Optimized for Flow of Change
Traditional flow of change in an organization not optimized for flow, with a series of groups owning different activities and handing over the work to the next team. No information flows back from the live systems into teams building the software.

- Organisation optimised for flow of change:

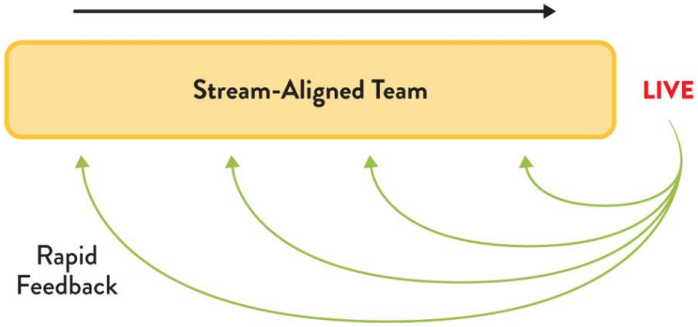


Figure 4.2: Organization Optimized for Flow of Change
Organizations set up for fast flow avoid hand-offs by keeping work within the stream-aligned team, and they ensure that the rich set of operational information flows back into the team.

- TIP

SRE teams are not essential; they are optional.

That's right: not every development team at Google uses SRE. "Downscale the SRE support if your project is shrinking in scale, and finally let your development team own the SRE work if the scale doesn't require SRE support," said Jaana B. Dogan, SRE at Google.[10]

- Relationship between SRE and Application Team

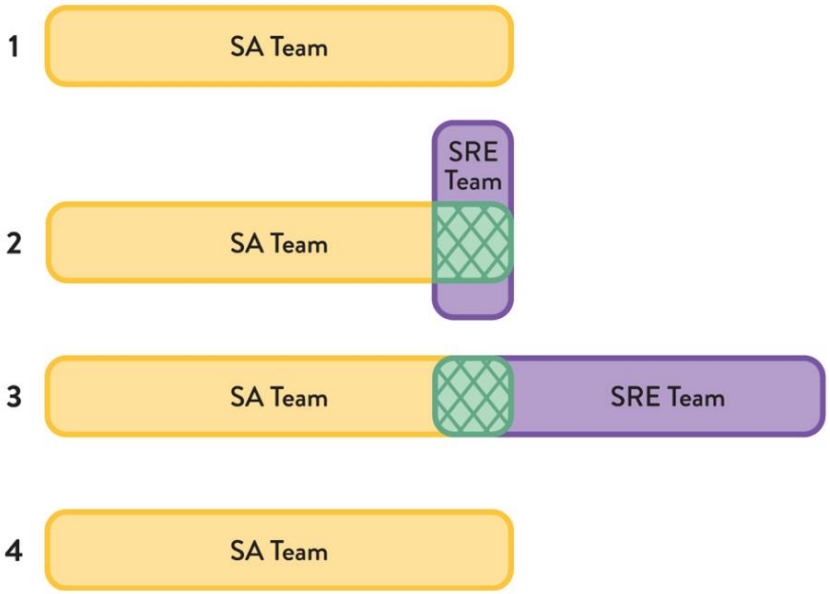


Figure 4.3: Relationship between SRE Team and Application Team

- 2x2 showing org size / software scale vs Engineering maturity

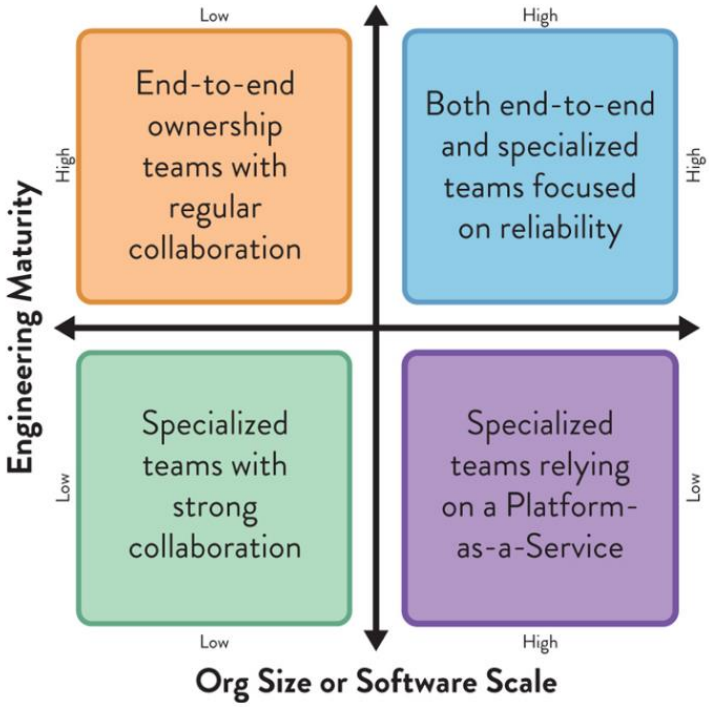


Figure 4.4: Influence of Size and Engineering Maturity on Choice of Topologies
Organization size (or software scale) and engineering discipline influence the effectiveness of team interaction patterns.

-
- Chapter 5: The four fundamental Team Topologies
  - Summary
    - The four fundamental team topologies simplify modern software team interactions
    - Mapping common industry team types to the fundamental topologies sets up organisations for success, removing grey 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: team of teams
  - Four fundamental team topologies:

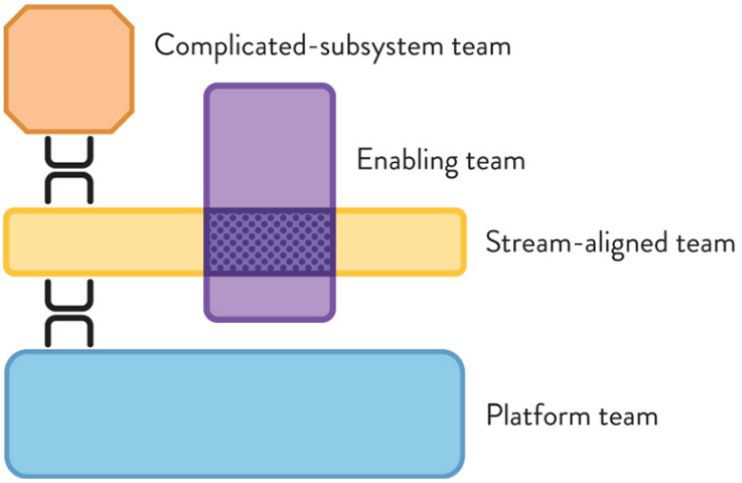


Figure 5.1: The Four Fundamental Team Topologies

-

- Stream aligned teams
    - A stream aligned team aims to produce a steady flow of feature delivery
    - … is quick to course correct based on feedback
    - …. Uses an experimental approach to product evolution, expecting to constantly learn and adapt
    - … has minimal (ideally zero) hand offs of work to other teams
    - … is evaluated on the sustainable flow of change it produces
    - … must have time and space to address code quality changes
    - …. Proactively and regularly reaches out to the supporting fundamental topologies teams
    - …. Members feel they have achieved or are in the path to achieving autonomy, mastery and purpose, the three key components of engaged knowledge workers according to Daniel Pink.
- Enabling Teams
    - An enabling team proactively seeks to understand the needs of stream aligned teams establishing regular checkpoints and jointly agreeing when more collaboration is needed
    - … stays ahead of the curve in keeping abreast of new approaches, tooling and practices in their area of expertise, well before an actual need is expected from stream aligned teams
    - …. Acts as a messenger of both good news and bad news.
    - Occasionally the enabling team might act as a proxy for external or internal services that are currently too difficult for stream aligned teams to use directly
    - An enabling team promotes learning not only insight=de the enabling team but across stream aligned teams, acting as a curator that facilitates appropriate knowledge sharing inside the organisation
- Complicated sub-system team
    - A complicated sub system team is mindful of the current stage of development of the subsystem and acts accordingly: high collaboration with stream aligned teams during early exploration and development phases; reduced interaction and focus on the subsystem interface and feature evolution and usage during later stages when the subsystem has stabilised
    - The complicated subsystem team correctly prioritises and delivers upcoming work respecting the needs of the stream aligned teams that use the complicated subsystem
- Platform teams
    - A platform team uses strong collaboration with stream aligned teams to understand their needs
    - … relies on fast prototyping techniques and involves stream aligned team members for fast feedback on what works and what does not
    - … has a strong focus on usability and reliability for their services and regularly assess if the services are still fit for purpose and usable

- … leads by example using the services they provide internally, partnering with stream aligned teams and enabling teams and consuming lower level platforms whenever possible
- … understands that adoption of internal new services like new technologies is not immediate but instead evolves along an adoption curve

▪ Platform composed of several fundamental team topologies

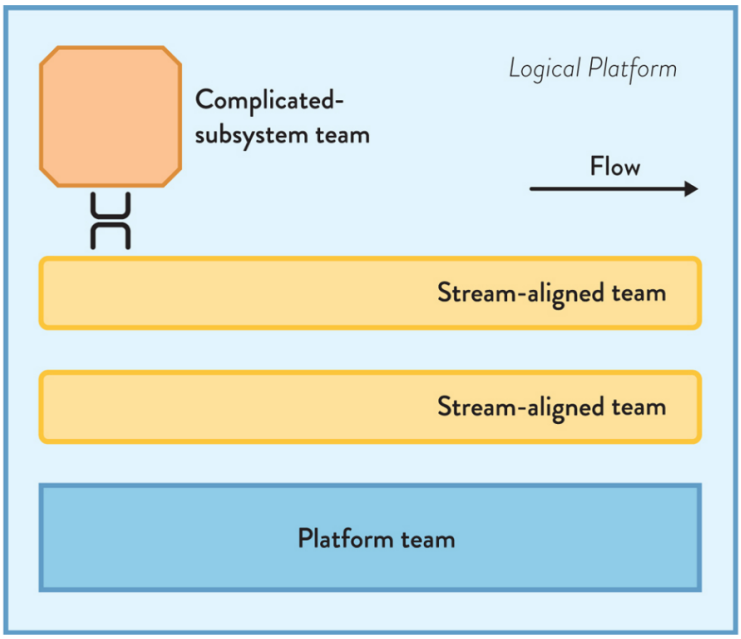


Figure 5.2: Platform Composed of Several Fundamental Team Topologies
In a large organization, the platform is composed of several other fundamental team topologies: stream-aligned Dev teams, complicated-subsystem teams, and a lower-level platform.

- Avoid team silos in the flow of change. Silos such as: Testing / QA, DBA, UX, Architecture, Data processing.
- A good platform is just big enough.
- Cognitive load reduction and accelerated product development
- Compelling, consistent, well chosen constraints
- Built on an underlying platform
- Manage a live product or service
- Convert common team types to the fundamental team topologies
- Move to mostly stream aligned teams for longevity and flexibility
- Infrastructure teams to platform teams
- Traditional infrastructure team organisation:

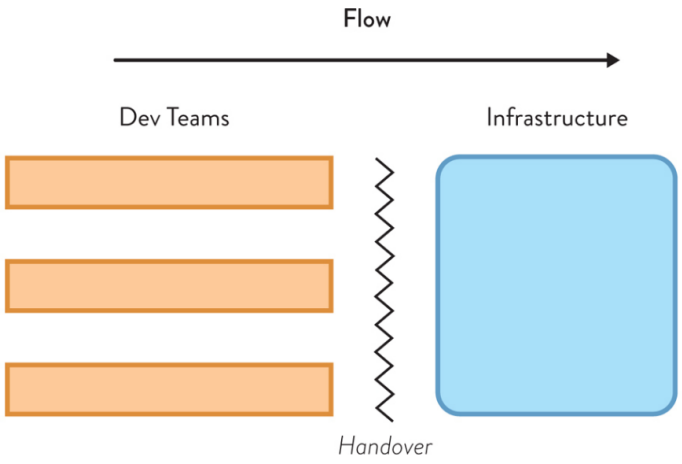


Figure 5.3: Traditional Infrastructure Team Organization
Many traditional infrastructure teams (on the right) blocked flow by being responsible for all changes to production infrastructure, including application changes, frequently gated by ITIL change processes. Work from Dev teams (on the left) was handed over to infrastructure or Ops teams for deployment, preventing flow.

- Component teams to platform or other team types
- Tooling teams to enabling teams or part of the platform
- Converting support teams
- Support team aligned to stream of change.

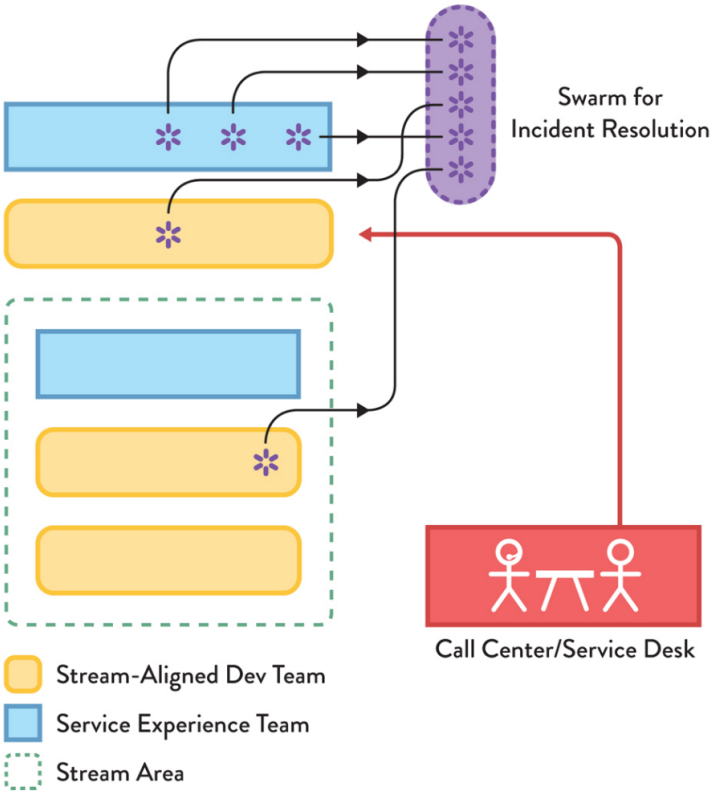


Figure 5.4: Support Teams Aligned to Stream of Change
The new model for support teams: aligned to the flow of change, usually paired with one or more stream-aligned Dev teams. Incidents are handled with dynamic "swarming."

- Converting architecture and architects
- Summary: use loosely coupled module groups of four specific team types.

- Chapter 6: Choose Team-First Boundaries
  - Summary:
    - 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
- A team first approach to software responsibilities and boundaries
- Hidden monoliths and coupling
- Application monolith
- Joined at the database monolith
- Monolith builds (Rebuild everything)
- Monolithic (coupled) releases
- Monolithic model (Single view of the world)
- Monolithic thinking (standardisation)
- Monolithic workplace (open plan office)
- Software boundaries or ‘fracture planes’
- Fracture plane: business domain bounded context
- Fracture plane: regulatory compliance
- Fracture plane: Change cadence
- Fracture Plane: Team Location
- Fracture Plane: Risk
- Fracture Plane: Performance isolation
- Fracture plane: technology
- Fracture Plane: User personas
- Natural ‘Fracture Planes’ for your specific organisation or technologies
- Real world example: Manufacturing
- Mobile, cloud and IOT Technology fracture plane scenario

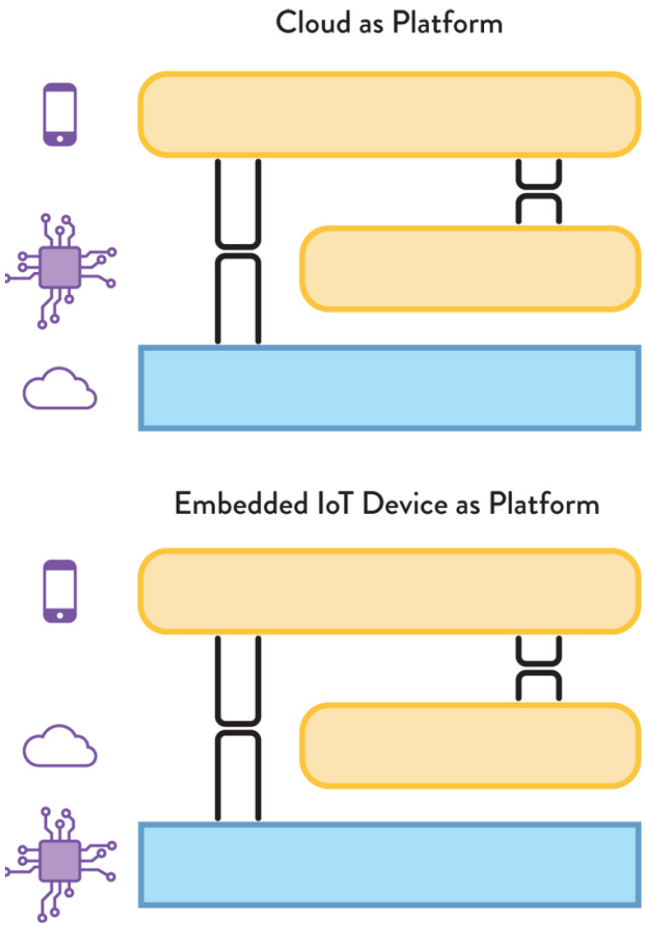


Figure 6.1: Mobile, Cloud, and IoT Technology Fracture Plane Scenario

- Part 3 Evolving team interactions for innovation and rapid delivery
    - Chapter 7: Team interaction modes
        - Summary
            - Choose specific team interaction modes to enhance software delivery.
            - Choose between three team interaction modes – collaboration, X-as-a-Service and facilitating – 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 defects problems.
        - Well defined interactions are key to effective teams
        - Collaboration vs X as a service

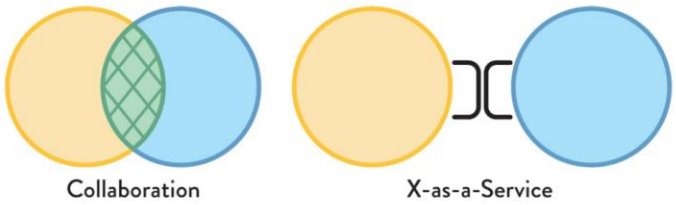


Figure 7.1: Collaboration vs. X-as-a-Service
Collaboration means explicitly working together on defined areas. X-as-a-Service means one team consumes something "as a service" from another team.

- The three essential team interaction modes
- The three team interaction modes:

Figure 7.2: The Three Team Interaction Modes
Collaboration mode is shown with diagonal cross-hatching, X-as-a-Service mode is shown with brackets, and facilitating is shown with dots.

  - Collaboration: working closely together with another team
  - X as a service: consuming or providing something with minimal collaboration
  - Facilitating: Helping or being helped by another team to clear impediments.
- Team interaction modes scenario:

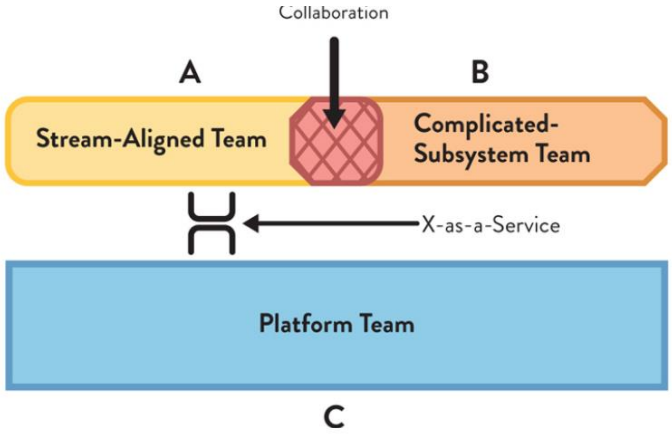


Figure 7.3: Team Interaction Modes Scenario
Stream-aligned Team A collaborates with complicated-subsystem Team B (shown with cross-hatching) while also consuming the platform provided by Team C, using the X-as-a-Service mode (shown with brackets).

- Collaboration: Driver of innovation and rapid discovery but boundary blurring

Table7.1: Advantages and Disadvantages of Collaboration Mode

Advantages	Disadvantages
• Rapid innovation and discovery • Fewer hand-offs	• Wide, shared responsibility for each team • More detail/context needed between teams, leading to higher cognitive load • Possible reduced output during collaboration compared to before
Constraint: A team should use collaboration mode with, at most, one other team at a time. A team should not use collaboration with more than one team at the same time.	
Typical Uses: Stream-aligned teams working with complicated-subsystem teams; stream-aligned teams working with platform teams; complicated-subsystem teams working with platform teams	

- X as a service: clear responsibilities with predictable delivery but needs good product management.

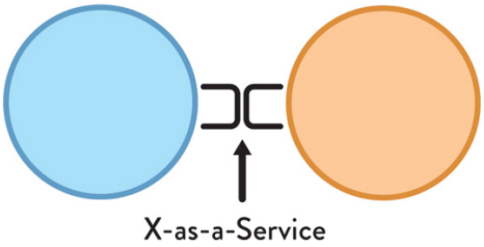


Figure 7.4: X-as-a-Service Team Interaction Mode
In this case, the team on the right is providing something "as a service" to the team on the left (perhaps an API, some developer tooling, or even an entire platform).

- Facilitating: Sense and reduce gaps in capabilities

Table7.3: Advantages and Disadvantages of Facilitation Mode

Advantages	Disadvantages
• Unblocking of stream-aligned teams to increase flow • Detection of gaps and misaligned capabilities or features in components and platforms	• Requires experienced staff to not work on "building" or "running" things • The interaction may be unfamiliar or strange to one or both teams involved in facilitation
Constraint: A team should expect to use the facilitating interaction mode with a small number of other teams simultaneously, whether consuming or providing the facilitation.	
Typical Uses: An enabling team helping a stream-aligned, complicated-subsystem, or platform team; or a stream-aligned, complicated-subsystem, or platform team helping a stream-aligned team.	

- Team behaviours for each interaction mode
- Team behaviours for collaboration node: High interaction and mutual respect.
- Team behaviours for X as a service mode: Emphasizes the user experience
- Team behaviours for facilitating mode: Help and be helped
- Choosing suitable team interaction modes
- Primary interaction modes for the four fundamental team topologies.

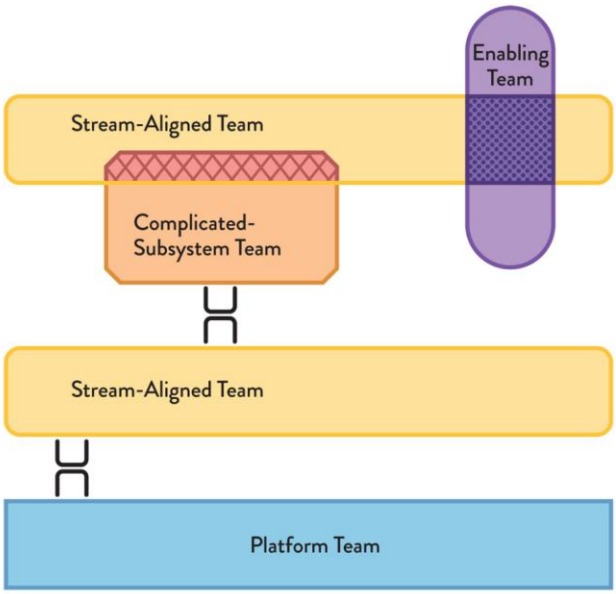


Figure 7.5: Primary Interaction Modes for the Four Fundamental Team Topologies
Stream-aligned teams use X-as-a-Service or collaboration; enabling teams use facilitation; complicated-subsystem teams use X-as-a-Service; platform teams use X-as-a-Service for teams that consume the platform.

The typical and occasional team interaction modes of the fundamental team topologies can be mapped to the essential team interaction modes as shown in Table 7.4.

Table 7.4: Team interaction modes of the fundamental team topologies

	Collaboration	X-as-a-Service	Facilitating
Stream-aligned	Typical	Typical	Occasional
Enabling	Occasional		Typical
Complicated-subsystem	Occasional	Typical	
Platform	Occasional	Typical	

- Choosing basic team organisation
- Team interaction modes at IBM around 2014

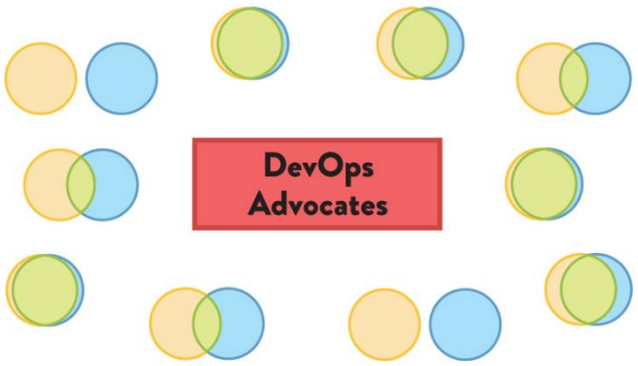


Figure 7.6: Team Interaction Modes at IBM around 2014
Team interaction modes at IBM around 2014, with a team of "DevOps advocates" coordinating and facilitating learning and team changes.

- Use the reverse Conway manoeuvre with collaboration and facilitating interactions
- Discover effective APIs between team by deliberate evolution of team topologies

        - Choose team interaction modes to reduce uncertainty and enhance flow
        - Change the team interaction mode temporarily to help a team grow experience and empathy
        - Use awkwardness in team interactions to sense missing capabilities and misplaced boundaries
    - Chapter 8: Evolve team structures with organisational sensing
        - Summary:
            - 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
            - Export multiple simultaneous team topologies to meet different needs.
            - Recognise triggers for organisation change
            - Treat operations as high fidelity sensory input for self steering
        - How much collaboration is right for each team interaction?
        - Accelerate learning and adoption of new practices
        - Collaboration between cloud and embedded teams

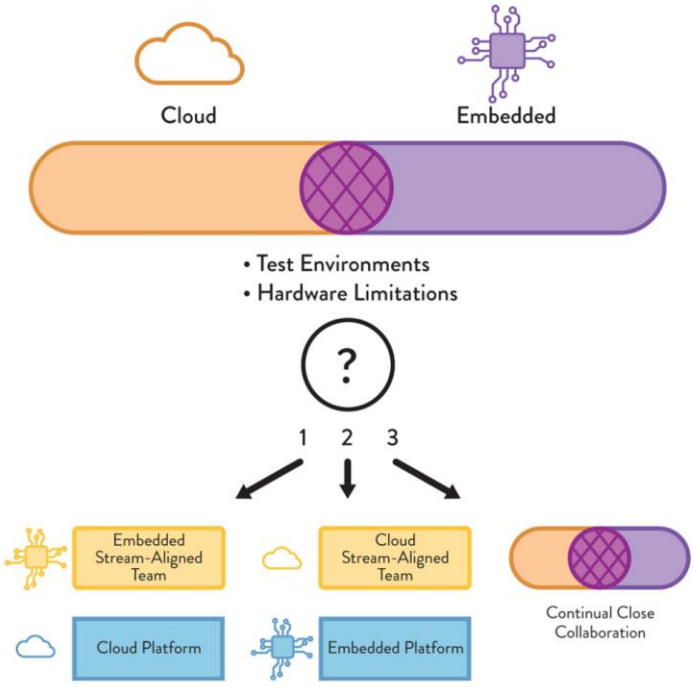


Figure 8.1: Collaboration between Cloud and Embedded Teams
Two teams ("cloud" and "embedded") collaborate to share practices and increase awareness. The results will include heightened awareness of the options for future team interactions: (1) treat the cloud software as a platform for the embedded team to use, (2) treat the embedded devices as a platform for the cloud team to use, or (3) continue with close collaboration.

        - System build and platform build team at TransUnion. Then showing System build and platform build team collaboration at trans union

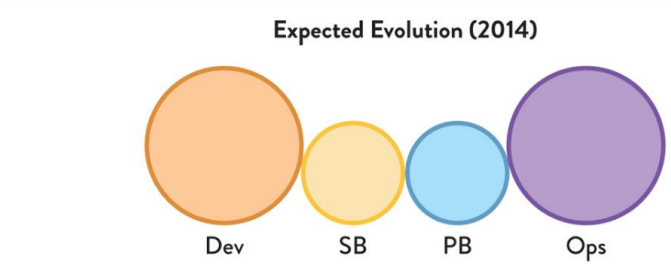


Figure 8.2: System-Build and Platform-Build Team at TransUnion
A team from Dev (SB) and a team from Ops (PB) exploring close interactions.

We drew out a timeline of the team evolution, inspired by the DevOps Team Topologies patterns. We expected to need to focus on awareness or operability within Dev teams to begin with, so our initial team evolution had the SB team collaborating closely with the Dev teams and the PB team. This really helped improve things, like deployment automation, metrics, logging, and other operational aspects (see Figure 8.3 on page 158).

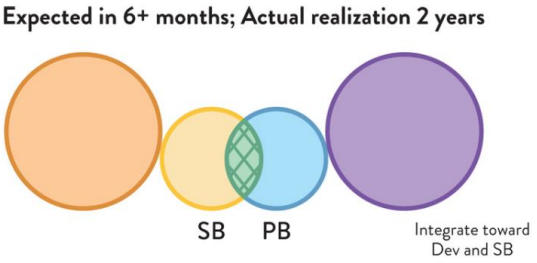


Figure 8.3: System-Build and Platform-Build Team Collaboration at TransUnion
The two teams, SB and PB, collaborating closely.

Back in 2014, we expected to have evolved the SB and PB teams to a nice enabling team within twelve months. As it happened, this transition took quite a bit longer than we initially thought (three years longer!) as we started to move our services to Azure; but by early 2018, we had made this work (see Figure 8.4 on page 158). The benefits to the business

-
  - Expected in 12+ months, actual realisation 4 years. Then showing 2018

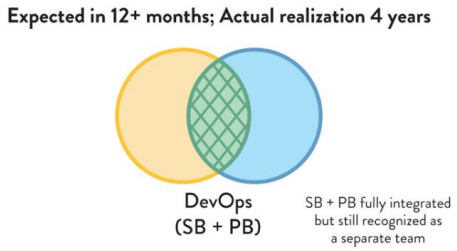


Figure 8.4: System-Build and Platform-Build Teams Merged at TransUnion
The SB and PB teams merged, helping to bring Dev and Ops together.

By late 2018, we had gone even further, finally merging the SB team back into the Dev teams and PB into the Ops teams, bringing higher levels of operational awareness and accountability, and leaving Ops to manage the underlying platform, with some strategic infrastructure running in Azure Cloud (see Figure 8.5).

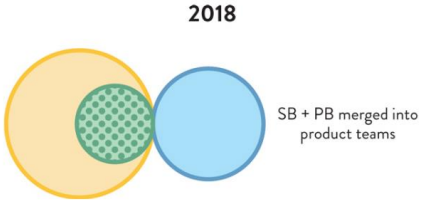


Figure 8.5: System-Build and Platform-Build Teams Merged Back into Dev and Ops at TransUnion
The SB and PB teams merged back into Dev and Ops, providing Platform-as-a-Service.

-
- Evolution of team topologies

Figure 8.6: Evolution of Team Topologies
The evolution of Team Topologies from close collaboration to limited collaboration (discovery) through to X-as-a-Service for established, predictable delivery.

- Evolution of team topologies in an enterprise

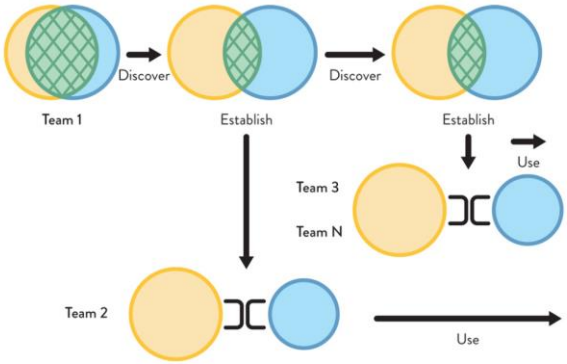


Figure 8.7: Evolution of Team Topologies in an Enterprise
Team 1 continues to collaborate with a platform team, discovering new patterns and ways of using new technologies. This discovery activity eventually enables Team 2 to adopt an X-as-a-Service relationship with the platform team. Later, Teams 3 and beyond adopt a later version of the platform, using it as a service without having to collaborate closely with the platform team.

- Combining team topologies for greater effectiveness
- Triggers for evolution of team topologies
- Trigger: Software has grown too large for one team
- Trigger: Delivery cadence is becoming slower
- Trigger: Multiple business services rely on a large set of underlying services
- Example of a platform wrapper

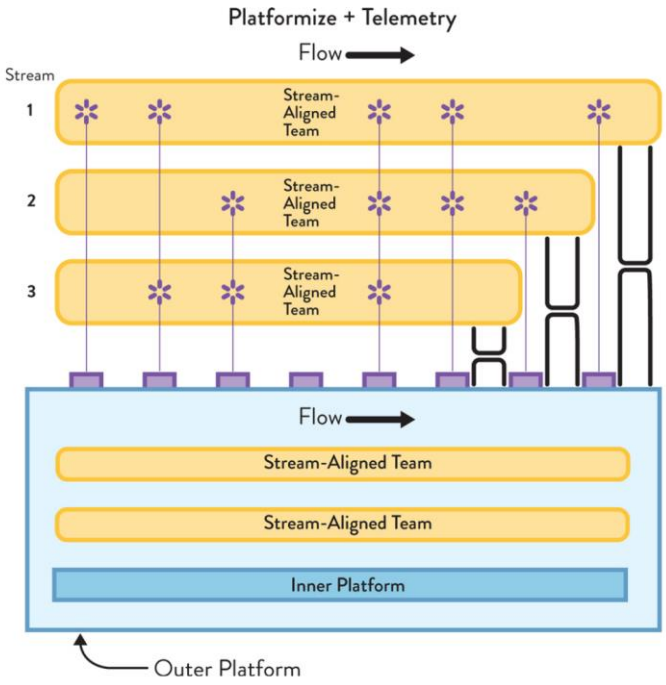


Figure 8.8: Example of a "Platform Wrapper"
Increase flow predictability in higher-level business services (streams) through the use of a "platform wrapper" to "platformize" the lower-level services and APIs, allowing the streams to treat all their dependencies as a single platform with a holistic roadmap and consistent DevEx. The streams also have rich telemetry to track flow and resource usage of the platform.

- Self steer design and development
- Treat teams and team interactions as senses and signals
- IT operations as high value sensory input to development
- New service and business as usual teams

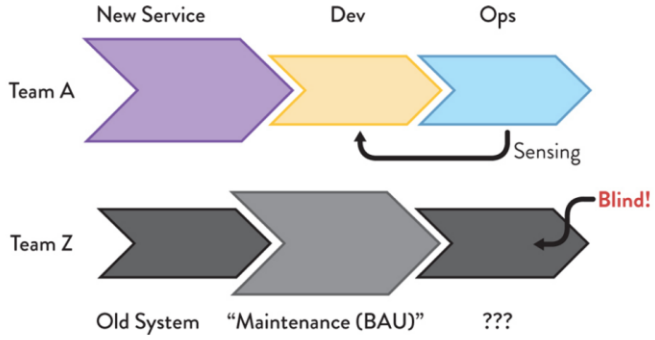


Figure 8.9: New-Service and "Business as Usual" (BAU) Teams
Having separate teams for "new stuff" and BAU tends to prevent learning, improvements and ability to self-steer. It is a non-cybernetic approach.

- Side by side new service and BAU teams

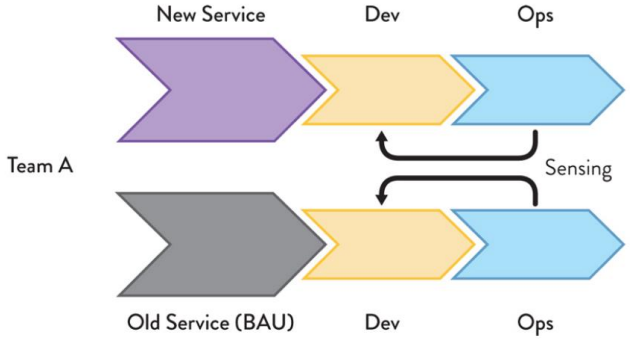


Figure 8.10: Side-by-Side New Service and BAU Teams
A cybernetic approach to maintaining older systems has a single stream-aligned team (or pair of teams) developing and running the new service and the older systems, enabling the team to retro-fit newer telemetry to the older system and increase the fidelity of the sensing from both systems.

- Conclusion: The next generation digital operating model
  - Summary
    - Combine a team first approach with Conway's law, the four fundamental topologies, team interaction modes, topology evolution and organisational sensing
    - Get started; begin with the team, identify the streams, identify the thinnest viable platform, identify capability gaps and practice team interactions.
  - Core ideas of team topologies

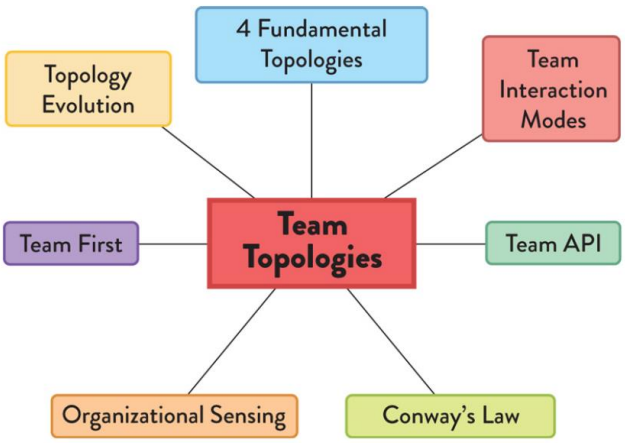


Figure 9.1: Core Ideas of Team Topologies

- Four team types and three interaction modes
- Team first thinking: Cognitive load, team API and Team sized architecture
- Strategic application of Conway's law
- Evolve organisation design for adaptability and sensing
- Team topologies alone are not sufficient for IT effectiveness
- Next steps how to get started with team topologies
    - Start with the team
    - Identify suitable streams of change
    - Identify the thinnest viable platform
    - Identify capability gaps in team coaching, mentoring, service management and documentation
    - Share and practice different interaction modes and explain principles behind new ways of working