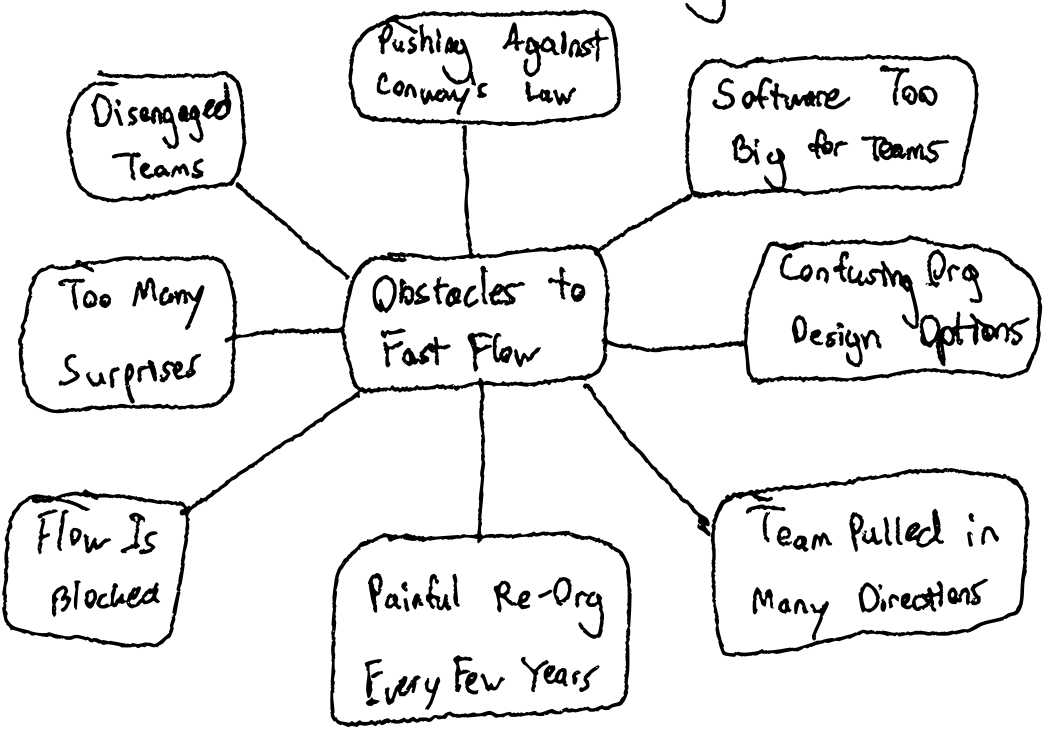
Ch.1 - The Problem With Org Charts

Org charts → static

Actual lines of communication differ from what is defined in org chart

Like docs with software, an org chart is quickly out of date with reality
- static representations (e.g matrix mgmt)

Conway's Law - strong correlation between an org's real communication paths & the resulting software architecture



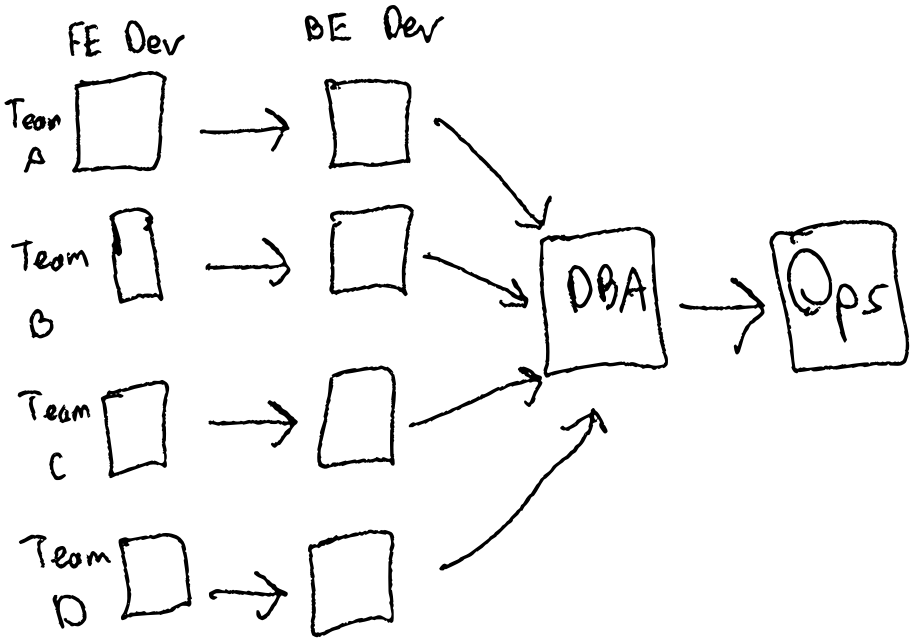
Ch. 2 - Conway's Law & Why It Matters

"whenever the code architecture & the org architecture are at odds, the code architecture wins."

The Reverse (or Inverse) Conway Maneuver

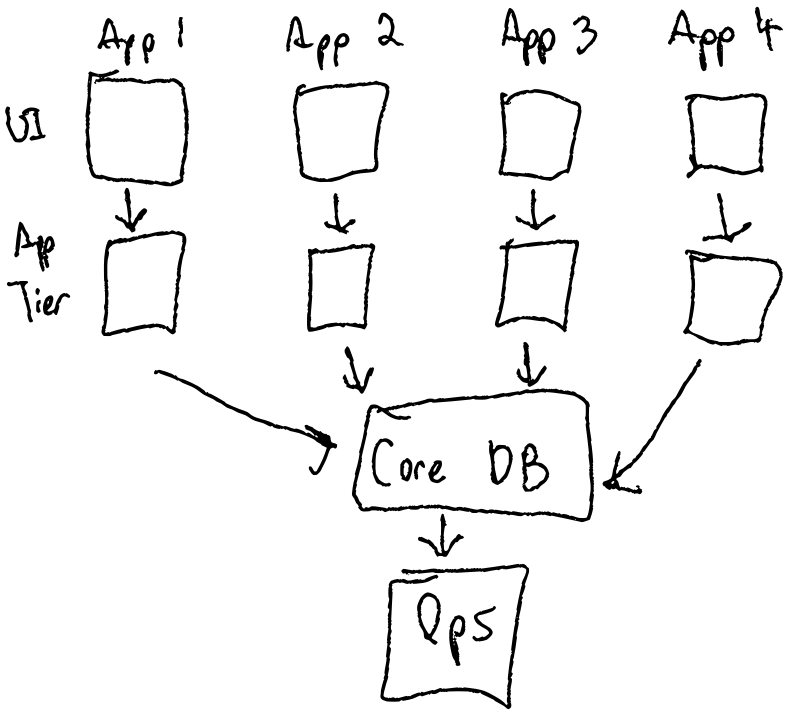
Orgs should evolve their team & organizational structure to achieve the desired architecture.

Ex.

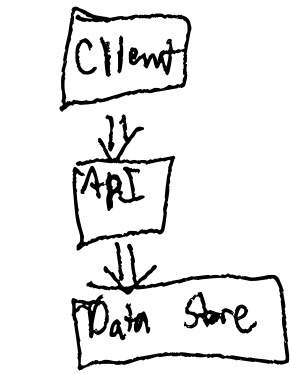


This team structure leads to shared DB.

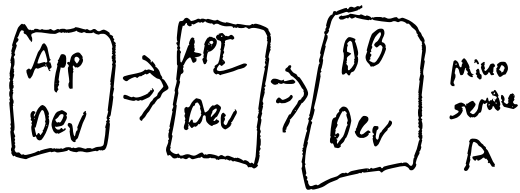
Leads to this type of architecture in code:



To Move to microservices architecture, put DB dev in team:



. . .



Team assignments are the first draft of the architecture

Restrict Unnecessary Communication

Focused communication between specific teams

Use "fracture plane" patterns to split up software into separate repos when necessary

Everyone does not need to communicate w/ everyone

- This mandate of "everyone must attend the stand up" or "be present in meetings" leads to monolithic, coupled design

Tool choices drive communication patterns

- Be mindful of the effect of shared tools on the way teams interact

Chapter 3 - Team-First Thinking

Use small, long-lived teams as the standard

Team - stable grouping of 5-9 people who work toward a shared goal as a limit

Dunbar
- 15 - limit of people one can trust deeply
- 5 - can be known & trusted closely
- 50 - people with whom we can have mutual trust
- 150 - people whose capabilities we can remember

Teams take time to form & be effective
- 2 weeks to 3 months

Tuckman Teal Performance Model
1) Forming - assembling for 1st time
2) Storming - working through differences
3) Norming - evolving ways of working together
4) Performing - high state of effectiveness

Team-first mindset
- reward whole team, not individuals

Cognitive Load

3 types

1) - intrinsic - relate to aspects of the task fundamental to the problem space

2) Extraneous - relates to the environment in which the task is being done

3) Germane - relates to the aspects of the task that need special attention for learning or high performance

Measure cognitive load by domain complexity

One complex domain per team

"Minimize cognitive load for others"

Design "Team APIs" and Facilitate Team Interactions

- Include code, documentation, & user experience

Facilitate team interactions for trust, awareness, & learning

- set space & time for teams & people to inter-communicate and learn

Design physical & virtual environments to help team interactions

Ch. 4 - Static Team Topologies

Consciously design teams instead of accidentally or haphazardly

Anti-pattern - ad-hoc team design
- shuffling team members

How can we reduce/avoid handovers between teams in the main flow of change?
where should the boundaries be in the software system in order to preserve system viability and encourage rapid flow?

Given:
- our skills
- constraints
- cultural & engineering maturity
- desired software architecture.
- business goals

which team topology will help us deliver results safer & faster?

DevOps topologies

- No one-size-fits-all approach, but:
  there are several anti-patterns detrimental to success

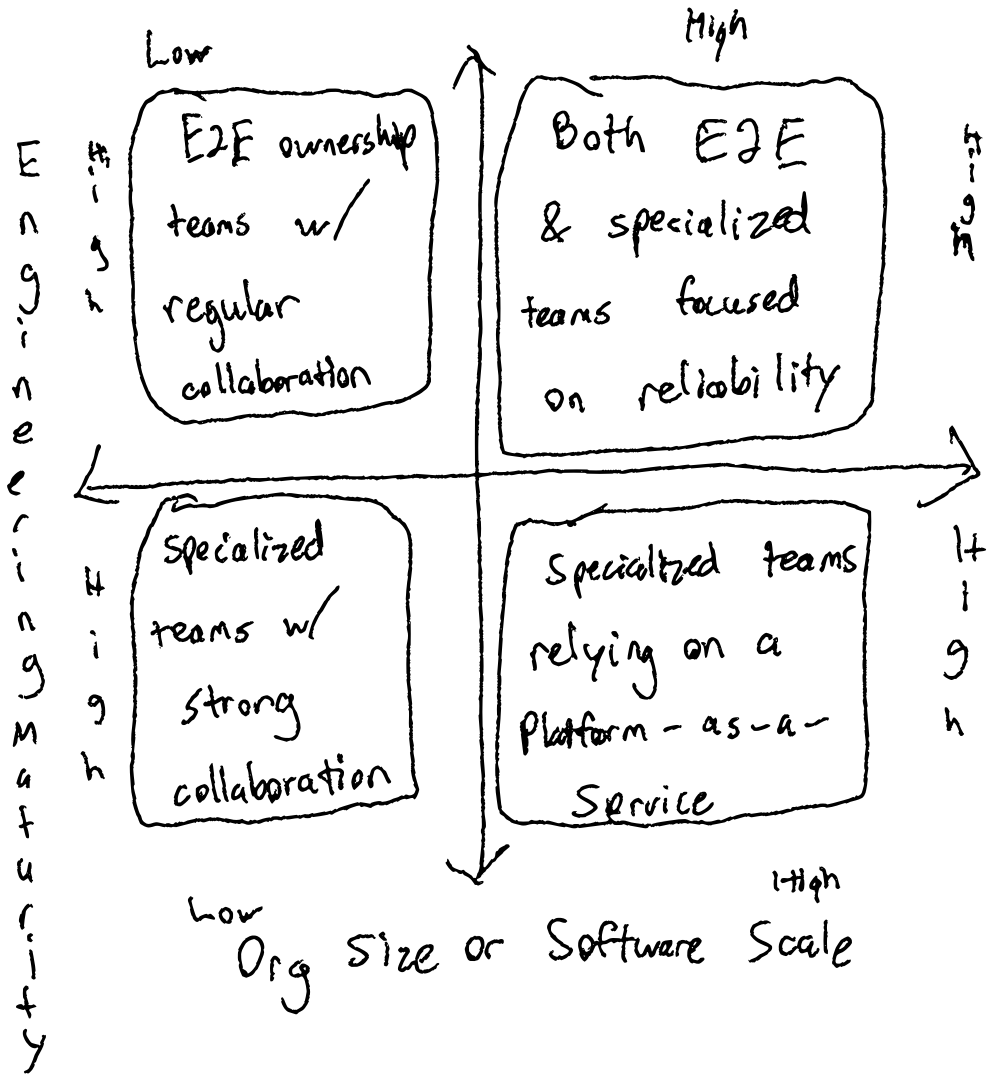
Cross-functional team ✓

"Communication conduits" help with inter-team communication

Need non-blocking teams / dependencies
- self-service capabilities

Split responsibility of designing cloud infra process and the actual provisioning & updates to application resources

SRE teams are optional, not essential



Summary: Adopt & evolve team topologies that match your current context

Ch.5 - The 4 Fundamental Team Topologies

4 team topologies

1) Stream-aligned team
2) Enabling team
3) Complicated-subsystem team
4) Platform team

No "Ops" team - teams are responsible for live operation - no handovers

Stream-aligned team

Stream - continuous flow of work aligned to a business domain or capability
This team type is the primary type in a company
Other team types exist to relieve the burden of this type
Each Stream should have steady, expectable work that is ready to prioritize
These teams need a mix of capabilities to work from requirements to production
"Stream-aligned" handles more than a "feature" or "product" so no feature or product teams

Stream-Aligned Team Behaviors

- Produce steady flow of feature delivery
- Quick to correct course based on feedback
- Use experimental approach
- Have minimal (ideally zero) hand-off work to other teams
- Evaluated on the sustainable flow of change it produces
- Have time & space to address code quality changes
- Proactively reaches out & teaches teams supporting them
- Have achieved or are in the path to achieving "autonomy, mastery, and purpose"

Enabling Teams

- Composed of specialists in a given technical (or product) domain who help bridge capability gaps
  - ex. UX, arch, testing, build engineering, CD, deployments,

Behaviors

- Seek to understand needs of stream-aligned teams
- Keep abreast of new approaches & techniques
- Act as messenger of both good & bad news
- Proxy for external services
- Promotes learning internally & across stream-aligned teams

Complicated-Subsystem Teams

- Build & maintain a part of the system that heavily depends on specialist knowledge
  - Reduce cognitive load of stream-aligned teams

Platform Teams

- Enable Stream-aligned teams to deliver work w/ substantial autonomy

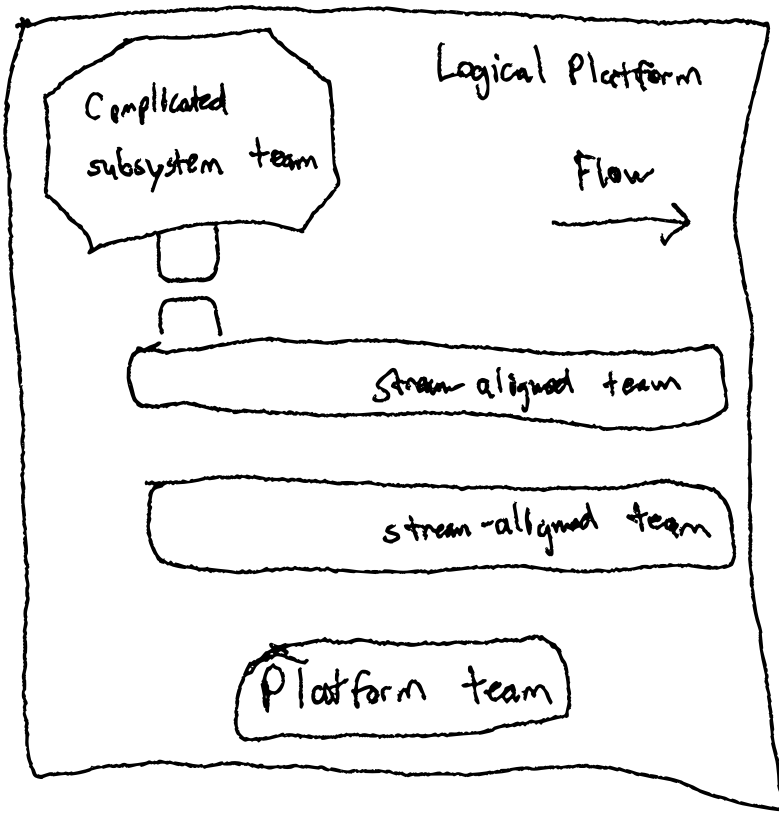
Platform - a foundation of self-service APIs, tools, services, knowledge, & support which are arranged as a compelling internal product. Autonomous delivery teams can make use of the platform to deliver product features at a higher pace, with reduced coordination.

- Ease of use
- Pure service providers for domain teams
- Focus on a *smaller* # of services of acceptable quality over numerous services w/ many resilience and quality problems
- Use internal pricing to regulate demand to avoid everyone asking for premium level service

Platform Team Expected Behaviors

- Strong collaboration w/ stream-aligned teams to understand needs
- Fast prototyping
- Strong focus on usability & reliability for services
- Lead by example
- Understands adoption evolves along a curve

Compose platform from groups of other fundamental teams



Avoid team silos

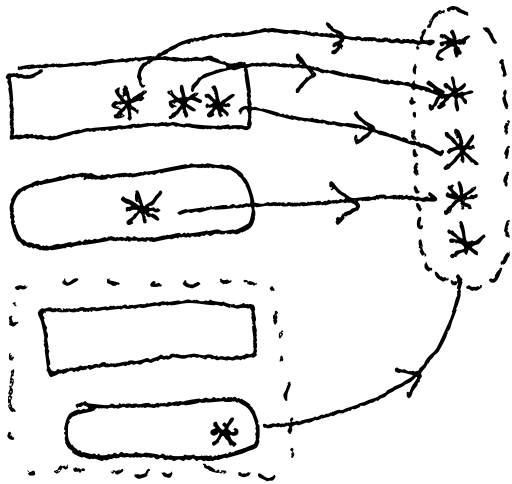
Convert common team types to the fundamental team topologies

Infrastructure team → Platform Team

Component team → Platform / Other type

Tooling team → Enabling team / part of the platform

Support teams → handle incidents w/ swarming



Architecture → Enabling team

Ch. 6 Choose Team-First Boundaries

Many kinds of monoliths

Application Monolith - single large app w/ many dependencies
- deployed as a unit

Joined-at-the-DB Monolith
- several apps/services coupled to same DB schema
- DB is viewed as core business engine by org
- DBA team(s) manage DB & are bottleneck to delivery

Monolithic Builds (rebuild everything)
- Gigantic CI build to get new version of component

Monolithic Releases
- set of smaller components bundled together into a "release"
- Components need each other for testing, so they all get deployed to same environment

Monolithic Model (single view of the world)
- Single domain language & representation across many different contexts

Monolithic Thinking (Standardization)
- One size fits all thinking for teams
- Restricts ability to choose tech & processes

Monolithic Workplace (Open-Plan Office) - single office layout pattern
- Need collocation of purpose AND bodies

Software Boundaries or "Fracture Planes"

Fracture Plane - Natural seam in the software system that allows to split the system into multiple parts easily

Best to align software boundaries w/ different business domains

Business Domain Bounded Context

Regulatory Compliance
- Isolate compliance aspect from rest of system

Change Cadence
- Split pieces off to not slow all parts of system

Team Location
- Best - full collocation or fully distributed
- Next best - Split system to teams in different locations

Risk
- Compliance
- # of users
- Marketing-driven changes

Performance Isolation
- Split system based off perf demands

Tech

User Personas
- Role-based
- Experience level

Natural Fracture Planes
- Can we provide or consume this subsystem as a service?

Ch.7 - Team Interaction Modes

Well-defined interactions are key

1) Collaboration - work closely w/ another team
2) X-as-a-service - consuming or providing something with minimal collaboration
3) Facilitating - helping (or being helped by) another team to clear impediments

Intermittent collaboration gives the best results

Formalize the way in which teams interact

Collaboration

+ Rapid innovation & 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

Don't collaborate with more than one team at a time

X-as-a-service

Service boundary must be well-chosen & well-implemented w/ good service management practice from providing team

+ Clarity of ownership w/ clear responsibility boundaries
+ Reduced detail/context needed between teams → reduced cognitive load
- Slower innovation of boundary or API
- Danger of reduced flow if the boundary API is not effective

Works w/ many teams simultaneously

Facilitating

Reduce gaps in capabilities

+ 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

Team Behaviors for Each Mode

Collaboration → high interaction & mutual respect

X-as-a-Service → emphasize user experience

Facilitating → help & be helped

Interaction modes of team topologies

	Collaboration	X-as-service	Facilitation
stream-aligned	Typical	Typical	Occasional
Enabling	Occasional	—	Typical
Complicated Subsystem	Occasional	Typical	—
Platform	Occasional	Typical	—

Use reverse Conway maneuver w/ Collaboration & Facilitating interactions

Discover effective APIs between teams by deliberate evolution of team topologies

Use the collaboration mode to discover viable X-as-a-service interactions
- Collaborate on potentially ambiguous interfaces until the interfaces are proven stable and functional

Use awkwardness in team interactions to sense missing capabilities and misplaced boundaries

Change team interaction mode temporarily to help a team grow experience & empathy

Ch.8 - Evolve Team Structures w/ Organizational Sensing

Reduce any ongoing collaboration between teams to explicit valuable activity

Change interaction modes of teams regularly, depending on what teams need to achieve

Triggers for Evolution of Team Topologies

Software has grown too large for one team

Delivery cadence is becoming slower

Multiple Business Services Rely on a large set of underlying services

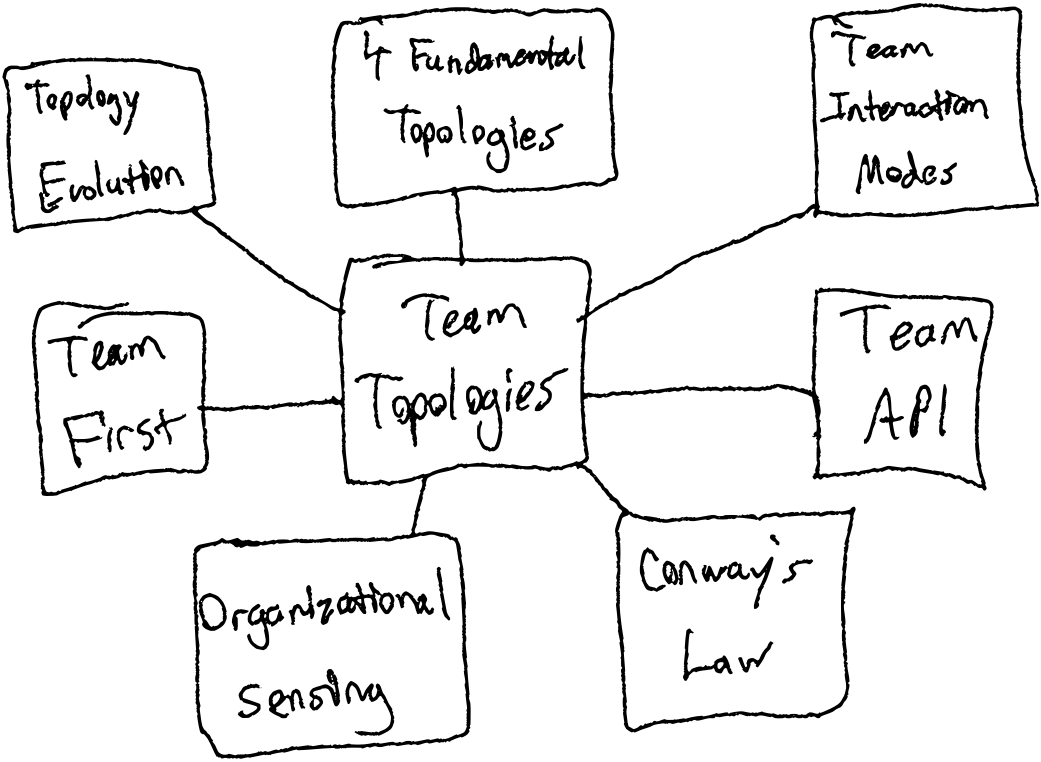
Treat teams & interactions as senses and signals

If Operations as high-value sensory input to development

3 ways of Devops

1) Systems thinking - optimize for fast flow across the whole org
2) Feedback loops - development informed and guided by Operations
3) Culture of continual experimentation and learning - sensing and feedback for every team interaction

Conclusion - The Next-Gen Digital Operating Model



How to get started w/ Team Topologies

1) Get started with the team
2) Identify suitable streams of change
3) Identify a Thinnest Viable Platform (TVP)
4) Identify capability gaps in team coaching, mentoring, service management, and documentation
5) Share & practice different interaction modes & explain principles behind new ways of working