TEAM TOPOLOGIES

ORGANIZING BUSINESS AND TECHNOLOGY TEAMS FOR FAST FLOW

MATTHEW SKELTON and MANUEL PAIS

Foreword by Ruth Malan
**CONTENTS**

Figures & Tables x
Case Studies & Industry Examples xi
Foreword by Ruth Malan xv
Preface xvii

**PART I  TEAMS AS THE MEANS OF DELIVERY**

Chapter 1: The Problem with Org Charts 3
  Communication Structures of an Organization 4
  Team Topologies: A New Way of Thinking about Teams 9
  The Revival of Conway’s Law 9
  Cognitive Load and Bottlenecks 11
  Summary: Rethink Team Structures, Purpose, and Interactions 13

Chapter 2: Conway’s Law and Why It Matters 15
  Understanding and Using Conway’s Law 15
  The Reverse Conway Maneuver 18
  Software Architectures that Encourage Team-Scoped Flow 21
  Organization Design Requires Technical Expertise 23
  Restrict Unnecessary Communication 24
  Beware: Naive Uses of Conway’s Law 26
  Summary: Conway’s Law Is Critical for Efficient Team Design in Tech 29

Chapter 3: Team-First Thinking 31
  Use Small, Long-Lived Teams as the Standard 32
  Good Boundaries Minimize Cognitive Load 39
  Design “Team APIs” and Facilitate Team Interactions 46
  Warning: Engineering Practices Are Foundational 55
  Summary: Limit Teams’ Cognitive Load and Facilitate Team Interactions to Go Faster 56

**PART II  TEAM TOPOLOGIES THAT WORK FOR FLOW**

Chapter 4: Static Team Topologies 61
  Team Anti-Patterns 62
  Design for Flow of Change 63
  DevOps and the DevOps Topologies 65
Successful Team Patterns 67
Considerations When Choosing a Topology 72
Use DevOps Topologies to Evolve the Organization 75
Summary: Adopt and Evolve Team Topologies that Match Your Current Context 77

Chapter 5: The Four Fundamental Team Topologies 79
Stream-Aligned Teams 81
Enabling Teams 86
Complicated-Subsystem Teams 91
Platform Teams 92
Avoid Team Silos in the Flow of Change 99
A Good Platform Is “Just Big Enough” 100
Convert Common Team Types to the Fundamental Team Topologies 104
Summary: Use Loosely Coupled, Modular Groups of Four Specific Team Types 109

Chapter 6: Choose Team-First Boundaries 111
A Team-First Approach to Software Responsibilities and Boundaries 112
Hidden Monoliths and Coupling 112
Software Boundaries or “Fracture Planes” 115
Real-World Example: Manufacturing 121
Summary: Choose Software Boundaries to Match Team Cognitive Load 123

PART III EVOLVING TEAM INTERACTIONS FOR INNOVATION AND RAPID DELIVERY
Chapter 7: Team Interaction Modes 131
Well-Defined Interactions Are Key to Effective Teams 132
The Three Essential Team Interaction Modes 133
Team Behaviors for Each Interaction Mode 137
Choosing Suitable Team Interaction Modes 144
Choosing Basic Team Organization 146
Choose Team Interaction Modes to Reduce Uncertainty and Enhance Flow 149
Summary: Three Well-Defined Team Interaction Modes 151
**Chapter 8: Evolve Team Structures with Organizational Sensing**

- Accelerate Learning and Adoption of New Practices 155
- Constant Evolution of Team Topologies 159
- Combining Teams Topologies for Greater Effectiveness 164
- Triggers for Evolution of Team Topologies 165
- Self Steer Design and Development 170
- Summary: Evolving Team Topologies 175

**Conclusion: The Next-Generation Digital Operating Model**

- Four Team Types and Three Interaction Modes 178
- Team-First Thinking: Cognitive Load, Team API, Team-Sized Architecture 179
- Strategic Application of Conway’s Law 180
- Evolve Organization Design for Adaptability and Sensing 181
- Team Topologies Alone Are Not Sufficient for IT Effectiveness 181
- Next Steps: How to Get Started with Team Topologies 183

**Glossary**

187

**Recommended Reading**

189

**References**

191

**Notes**

203

**Index**

207

**Acknowledgments**

215

**About the Authors**

216
FIGURES & TABLES

FIGURES

0.1: The Four Team Types and Three Interaction Modes xx
1.1: Org Chart with Actual Lines of Communication 6
1.2: Obstacles to Fast Flow 12
2.1: Four Teams Working on a Software System 19
2.2: Software Architecture from Four-Team Organization 20
2.3: Microservices Architecture with Independent Services and Data Stores 21
2.4: Team Design for Microservices Architecture with Independent Services and Data Stores 22
2.5: Inter-Team Communication 25
3.1: Scaling Teams Using Dunbar’s Number 34
3.2: No More than One Complicated or Complex Domain per Team 44
3.3: Typical vs. Team-First Software Subsystem Boundaries 45
3.4: Office Layout at CDL 52
4.1: Organization not Optimized for Flow of Change 64
4.2: Organization Optimized for Flow of Change 65
4.3: Relationship between SRE Team and Application Team 71
4.4: Influence of Size and Engineering Discipline on Team Interaction Patterns 73
5.1: The Four Fundamental Team Topologies 80
5.2: Platform Composed of Several Fundamental Team Topologies 96
5.3: Traditional Infrastructure Team Organization 105
5.4: Support Teams Aligned to Stream of Change 107
6.1: Mobile, Cloud, and IoT Technology Fracture Plane Scenario 124
7.1: Collaboration vs. X-as-a-Service 133
7.2: The Three Essential Team Interaction Modes 134
7.3: Team Interaction Modes Scenario 135
7.4: X-as-a-Service Team Interaction Mode 138
7.5: Primary Interaction Modes for the Four Fundamental Team Topologies 145
7.6: Team Interaction Modes at IBM around 2014 146
8.1: Collaboration between Cloud and Embedded Teams 156
8.2: System Build and Platform Build Team at TransUnion 158
8.3: System Build and Platform Build Team Collaboration at TransUnion 158
8.4: System Build and Platform Build Teams Merged at TransUnion 158
8.5: System Build and Platform Build Teams Merged Back Into Dev and Ops at TransUnion 159
8.6: Evolution of Team Topologies 160
8.7: Evolution of Team Topologies in an Enterprise 160
8.8: Example of a “Platform Wrapper” 168
8.9: New-Service and “Business as Usual” (BAU) Teams 173
8.10: Side-by-Side New Service and BAU Teams 174
9.1: Core Ideas of Team Topologies 178

TABLES
Table 7.1: Advantages and Disadvantages of Collaboration Mode 137
Table 7.2: Advantages and Disadvantages of X-as-a-Service Mode 140
Table 7.3: Advantages and Disadvantages of Facilitating Mode 141
Table 7.4: Team Interaction Modes of the Fundamental Team Topologies 144

CASE STUDIES & INDUSTRY EXAMPLES

Chapter 1
Industry Example: OutSystems (Part 1)—Miguel Antunes, R&D Principal Software Engineer, OutSystems 11

Chapter 2
Industry Example: Adidas—Fernando Cornago, Senior Director Platform Engineering, and Markus Rautert, Vice President Platform Engineering and Architecture, Adidas 16

Chapter 3
Industry Example: OutSystems (Part 2)—Miguel Antunes, R&D Principal Software Engineer, OutSystems 42
Industry Example: IKEA—Albert Bertilsson, Solution Team Lead, and Gustaf Nilsson Kotte, Web Developer, IKEA 46
Case Study: Team-Focused Office Space at CDL—
Michael Lambert, Head of Development, and Andy Rubio,
Development Team Leader, CDL 50

Case Study: Stream-Aligned Office Layout for
Flow-Based Collaboration at Auto Trader—
Dave Whyte, Operations Engineering Lead, and
Andy Humphrey, Head of Customer Operations, Auto Trader 53

Chapter 4

Industry Example: Spotify—Henrik Kniberg, Agile/Lean Coach, and Anders Ivarsson, Organizational Coach, Spotify 63

Industry Example: Feature Teams Supported by Cross-Subsystem Functions at Ericsson—Wolfgang John, Research Leader, Ericsson 68

Industry Example: DevOps Team Topologies at a Healthcare Organization—Pulak Agrawal, DevOps Manager and Technology Architect, Accenture

Case Study: Evolution of Team Topologies at TransUnion (Part 1)—
Ian Watson, Head of DevOps, TransUnion 76

Chapter 5

Case Study: Strictly Independent Service Teams at Amazon 82

Case Study: Engineering Enablement Team within a Large Legal Organization—Robin Weston, Engineering Leader, BCG Digital Ventures 88

Case Study: Sky Betting & Gaming—Platform Feature Teams (Part 1)—
Michael Maibaum, Chief Architect, Sky Betting & Gaming 94

Case Study: Evolving Highly Responsive IT Operations at Auto Trader—Dave Whyte, Operations Engineering Lead, and Andy Humphrey, Head of Customer Operations, Auto Trader 97

Chapter 6

Case Study: Finding Good Software Boundaries at Poppulo—
Stephanie Sheehan, VP of Operations, and Damien Daly, Director of Engineering, Poppulo 121

Chapter 7

Case Study: Team Interaction Diversity at IBM around 2014—
Eric Minick, Program Director for Continuous Delivery, IBM 146
Chapter 8

Case Study: Adoption of Kubernetes to Drive Organizational Change at uSwitch—Paul Ingles, Head of Engineering, uSwitch 155

Case Study: Evolution of Team Topologies at TransUnion (Part 2)—Dave Hotchkiss, Platform Build Manager, TransUnion 157

Case Study: Sky Betting and Gaming—Platform Feature Teams (Part 2)—Michael Maibaum, Chief Architect, Sky Betting & Gaming 162
Figure 0.1: The Four Team Types and Three Interaction Modes
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.
Figure 1.2: Obstacles to Fast Flow

- Pushing Against Conway’s Law
- Disengaged Teams
- Painful Re-Org Every Few Years
- Flow Is Blocked
- Too Many Surprises
- Confusing Org Design Options
- Software Too Big for Teams
- Team Pulled in Many Directions
Figure 2.1: Four Teams Working on a Software System
Four separate teams consisting of front-end and back-end developers work on a software system. Front-end devs communicate only with back-end devs, who communicate with a single DBA for the database changes.
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 2.3: Microservices Architecture with Independent Services and Data Stores
A microservices-based architecture with four separate services, each with its own data store, API layer, and front-end client.
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.)
Communication within teams is high bandwidth. Communication between two “paired” teams can be mid bandwidth. Communication between most teams should be low bandwidth.
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.
Figure 3.2: No More than One Complicated or Complex Domain per Team

Before: a larger team is spread thin across four domains (two complicated and two complex) and struggles to perform well. Intra-team morale is negatively affected, with frequent context switches and individual disengagement. After: with multiple smaller teams each focusing on a single domain, motivation rises and the team delivers faster and more predictably. Low bandwidth inter-team collaboration allows solving occasional issues affecting two or more domains.
Figure 3.3: Typical vs. Team-First Software Subsystem Boundaries
Closed-off meeting room(s)

Squad areas offset to provide squad standup or whiteboarding space

Figure 3.4: Office Layout at CDL
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.
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.
Figure 4.3: Relationship between SRE Team and Application Team
Organization size (or software scale) and engineering discipline influence the effectiveness of team interaction patterns.

**Figure 4.4: Influence of Size and Engineering Maturity on Choice of Topologies**

End-to-end ownership teams with regular collaboration

Both end-to-end and specialized teams focused on reliability

Specialized teams with strong collaboration

Specialized teams relying on a Platform-as-a-Service
Figure 5.1: The Four 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.
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.
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.”
Figure 6.1: Mobile, Cloud, and IoT Technology Fracture Plane Scenario
With three very disparate technologies (mobile, cloud, and IoT), an organization must decide on an approach to fracture planes that makes sense based on the cognitive load and the change cadence in each area.
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.

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.

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).
### Table 7.1: Advantages and Disadvantages of Collaboration Mode

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Rapid innovation and discovery</td>
<td>• Wide, shared responsibility for each team</td>
</tr>
<tr>
<td>• Fewer hand-offs</td>
<td>• More detail/context needed between teams, leading to higher cognitive load</td>
</tr>
<tr>
<td></td>
<td>• Possible reduced output during collaboration compared to before</td>
</tr>
</tbody>
</table>

**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
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).

### Table 7.2: Advantages and Disadvantages of X-as-a-Service Mode

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Clarity of ownership with clear responsibility boundaries</td>
<td>• Slower innovation of the boundary or API</td>
</tr>
<tr>
<td>• Reduced detail/context needed between teams, so cognitive load is limited</td>
<td>• Danger of reduced flow if the boundary or API is not effective</td>
</tr>
</tbody>
</table>

**Constraint:** A team should expect to use the X-as-a-Service interaction with many other teams simultaneously, whether consuming or providing a service.

**Typical Uses:** Stream-aligned teams and complicated-subsystem teams consuming Platform-as-a-Service from a platform team; stream-aligned teams and complicated-subsystem teams consuming a component or library as a service from a complicated-subsystem team.
Table 7.3: Advantages and Disadvantages of Facilitation Mode

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Unblocking of stream-aligned teams to increase flow</td>
<td>• Requires experienced staff to not work on “building” or “running” things</td>
</tr>
<tr>
<td>• Detection of gaps and misaligned capabilities or features in components and platforms</td>
<td>• The interaction may be unfamiliar or strange to one or both teams involved in facilitation</td>
</tr>
</tbody>
</table>

**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.

Table 7.4: Team interaction modes of the fundamental team topologies

<table>
<thead>
<tr>
<th></th>
<th>Collaboration</th>
<th>X-as-a-Service</th>
<th>Facilitating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stream-aligned</td>
<td>Typical</td>
<td>Typical</td>
<td>Occasional</td>
</tr>
<tr>
<td>Enabling</td>
<td>Occasional</td>
<td></td>
<td>Typical</td>
</tr>
<tr>
<td>Complicated-subsystem</td>
<td>Occasional</td>
<td>Typical</td>
<td></td>
</tr>
<tr>
<td>Platform</td>
<td>Occasional</td>
<td>Typical</td>
<td></td>
</tr>
</tbody>
</table>
**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.
Team interaction modes at IBM around 2014, with a team of “DevOps advocates” coordinating and facilitating learning and team changes.
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.
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.

Expected Evolution (2014)

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

Expected in 6+ months; Actual realization 2 years

Integrate toward Dev and SB

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.

Expected in 12+ months; Actual realization 4 years

SB + PB fully integrated but still recognized as a separate team
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.
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.

Team 1
Discover → Establish

Team 2

Team 3
Establish

Team N
Use

Use

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.
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.
Having separate teams for “new stuff” and BAU tends to prevent learning, improvements and ability to self-steer. It is a non-cybernetic approach.

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.
Figure 9.1: Core Ideas of Team Topologies
**API (application programming interface):** A description and specification for how to interact programmatically with software.

**Application monolith:** A single, large application with many dependencies and responsibilities that possibly exposes many services and/or different user journeys.

**Bounded context:** A unit for partitioning a larger domain (or system) model into smaller parts, each of which represents an internally consistent business domain area.

**Brooks’s law:** Law coined by Fred Brooks which states that adding new people to a team doesn’t immediately increase the capacity of a team.

**Cognitive load:** The amount of working memory being used.

**Collaboration mode:** Team(s) working closely together with another team.

**Complicated-subsystem team:** Responsible for building and maintaining a part of the system that depends heavily on specialist knowledge.

**Conway’s law:** Law coined by Mel Conway that states that system design will copy the communication structures of the organization which designs it.

**Domain complexity:** How complex the problem is that is being solved via software.

**Dunbar’s number:** Coined by anthropologist Robin Dunbar, which states that fifteen is the limit of people one person can trust; of those, only around five can be known and trusted closely.

**Enabling team:** Team(s) composed of specialists in a given technical (or product) domain; they help bridge the capability gap.

**Extraneous cognitive load:** Relates to the environment in which the task is being done (e.g., “How do I deploy this component, again?” “How do I configure this service?”).

**Facilitating mode:** Team(s) helping (or being helped by) another team to clear impediments.

**Flow of change:** A stream of related updates or alterations to a software service or system, usually aligned to user goals or other core focus of the business.
**fracture plane:** a natural “seam” in the software system that allows it to be easily split into two or more parts.

**germane cognitive load:** relates to aspects of the task that need special attention for learning or high performance (e.g., “How should this service interact with the ABC service?”).

**intrinsic cognitive load:** relates to aspects of the task fundamental to the problem space (e.g., ”What is the structure of a Java class?” “How do I create a new method?”).

**joined-at-the-database monolith:** composed of several applications or services all coupled to the same database schema, making them difficult to change, test, and deploy separately.

**monolithic build:** uses one gigantic continuous integration (CI) build to get a new version of a component.

**monolithic model:** software that attempts to force a single domain language and representation (format) across many different contexts.

**monolithic release:** a set of smaller components bundled together into a “release.”

**monolithic thinking:** “one-size-fits-all” thinking for teams that leads to unnecessary restrictions on technology and implementation approaches between teams.

**monolithic workplace:** a single office layout pattern for all teams and individuals in the same geographic location.

**organizational sensing:** teams and their internal and external communication are the “senses” of the organization (sight, sound, touch, smell, taste).

**platform team:** enables stream-aligned teams to deliver work with substantial autonomy.

**reverse Conway maneuver:** organizations should evolve their team and organizational structure to achieve the desired architecture.

**stream-aligned team:** a team aligned to a single, valuable stream of work.

**team API:** an API surrounding each team.

**Team Topologies:** model for organizational design that provides a key technology-agnostic mechanism for modern software-intensive enterprises to sense when a change in strategy is required (either from a business or technology point of view).

**thinnest viable platform:** a careful balance between keeping the platform small and ensuring that the platform is helping to accelerate and simplify software delivery for teams building on the platform.

**X-as-a-Service mode:** consuming or providing something with minimal collaboration.
Key Management Concepts and Practices for Reliable, Fast Flow


Key Practices and Approaches for Organizations, Software, and Systems

- *Thinking in Promises* by Mark Burgess (Sebastopol, California: O’Reilly Media, 2015).

Key Engineering Practices that Enable Fast Flow


References | 51
References


O’Dell, Chris. “You Build It, You Run It (Why Developers Should Also Be on Call).” Skelton Thatcher.com (blog), October 18, 2017. https://skeltonthatcher.com/blog/build-run-developers-also-call/.


Foreword
1. Conway, “How Do Committees Invent?.”

Preface
2. Skelton, “How Different Team Topologies Influence DevOps Culture.”

Chapter 1
10. Lewis, “Microservices and the Inverse Conway.”

Chapter 2
1. “DevOps Over Coffee – Adidas;” Fernando Cornago, person email communication with the authors, March 2019.
3. Kniberg and Ivarsson, “Scaling Agile @ Spotify.”
4. Kniberg and Ivarsson, “Scaling Agile @ Spotify.”
5. Forsgren et al., Accelerate, 63.
8. Hastie, “An Interview with Sam Guckenheimer on Microsoft’s Journey to Cloud Cadence.”
9. Ben Treynor, as quoted in Niall Murphy, “What is ‘Site Reliability Engineering’?”
10. Dogan, “The SRE Model.”
15. Pulak Agrawal, personal communication with the authors, March 2019.
16. Pulak Agrawal, personal communication with the authors, March 2019.

Chapter 5
3. Lane, “The Secret to Amazon’s Success—Internal APIs;” Hoff, “Amazon Architecture.”
4. Crawford, “Amazon’s Two-Pizza Teams;” Munns, “Chris Munns, DevOps @ Amazon.”
7. Pink, Drive, 49.
10. DeMarco and Lister, Peopleware, 212.
11. Webber, Building Successful Communities of Practice, 11.
14. Neumark, “DevOps & Product Teams—Win or Fail?”
16. Womack and Jones, Lean Thinking.
21. Shibata, “How to Build a Platform Team Now!”
22. Shibata, “How to Build a Platform Team Now!”
24. Shibata, “How to Build a Platform Team Now!”
26. Forsgren et al., Accelerate, 68.

Chapter 6
1. Forsgren et al., Accelerate, 63.
2. Forsgren et al., Accelerate, 66
4. Evans, Domain-Driven Design.

**Chapter 7**

2. Rother, *Toyota Kata*, 236.

**Chapter 8**

1. Forsgren et al., *Accelerate*, 63.
2. Ingles, “Convergence to Kubernetes.”
3. Ingles, “Convergence to Kubernetes,”
5. Kotter, “Accelerate!”

**Conclusion**

15. Cohn, “Nine Questions to Assess Scrum Team Structure.”

Chapter 3
3. Rozovsky, “Re:Work—The Five Keys to a Successful Google Team.”
8. Lewis, “Microservices and the Inverse Conway Manoeuvre.”
9. Munns, “Chris Munns, DevOps @ Amazon.”
29. Jeff Bezos, as quoted in Lane, “The Secret to Amazon’s Success.”
31. Kniberg and Ivarsson, “Scaling Agile @ Spotify.”
32. Kniberg and Ivarsson, “Scaling Agile @ Spotify.”
33. Forsgren et al., *Accelerate*, 181.
34. Jeremy Brown, personal communication with the authors, March 2019.

Chapter 4
2. Kniberg and Ivarsson, “Scaling Agile @ Spotify.”
# INDEX

## A

**Accelerate: The Science of Dev Ops** (Forsgren, Humble, & Kim), 18, 64, 86, 112, 114, 154  
Accenture, 75–76  
"Acquisition Community Team Dynamics: The Tuckman Model vs. the DAU Model" (Knight), xx  
ad hoc team design, 62  
adaptive structuration theory, xix  
Adidas, 16  
adoption of new practices, 155–159  
**Agile IT Organization Design** (Narayan), 173  
Agrawal, Pulak, 75–76  
Amazon, 32, 35, 82–83, 112  
anti-patterns, 62  
Antunes, Miguel, 11  
APIs  
defined, 187  
effective, 148  
team, 47–56  
application monolith, 113, 187  
Auto Trader, 53–55, 97–99  
awkward team interactions, 150–151  
AWS, 49  
Axelrod, Robert, 49  
Azure, 69, 101

## B

Balena.io, 101  
basic team organization, 146–148  
BCG Digital Ventures, 88–90  
Beer, Stafford, 103  
benched bay approach, 51  
Bertilsson, Albert, 47  
Bezos, Jeff, 49, 82–83  
Boone, Mary, xxi  
Borland Delphi, 101  
Bottcher, Evan, 92  
bottlenecks, 11–12  
boundaries, 39–47  

domain limitations, 42–45  
misplaced, 150–151  
relative domain complexity, 41–42  
responsibility restriction, 39–41  
software boundary size, 45–47  
team-first, 111–126  
bounded context, 115–116, 187  
**Brain of the Firm** (Beer), 103  
Brooks, Fred, 35  
Brooks’s law, 40, 187  
Brown, Jeremy, 53  
Burgess, Mark, 49, 142  
business as usual teams, 173–174  
business domain bounded context, 115–116  
business process management, 169

## C

capabilities  
missing, 150–151  
self-service, 69  
capability gaps, 184–185  
“Capturing the Complexity in Advanced Technology Use: Adaptive Structuration Theory” (DeSanctis and Poole), xxi  
case studies  
complicated-subsystems teams, 94–95, 97–99  
DevOps Topologies, 75–77  
enabling teams, 88–90  
fraction planes, 121–125  
organizational sensing, 154–155, 157–159, 162–164  
software boundaries, 121–125  
static topologies, 75–77  
stream-aligned teams, 82–83  
team APIs, 50–52, 53–55  
team interaction modes, 146–147  
team types, 82–83, 88–90  
team-first boundaries, 121–125  
team-first thinking, 50–52, 53–55  
CDL, 50–52  
change cadence, 117  
cloud teams, 69–70

cognitive load, 11–12, 39–47  
defined, 187  
domain identification, 43  
domain limitations, 42–45  
ecosystem tuning, 46  
extraneous, 40, 187  
“Eyes On, Hands Off,” 46  
germane, 40, 188  
heuristics for domain assignment, 43  
intrinsic, 40, 188
cognitive load (continued)
  relative domain complexity, 41–42
  responsibility restriction, 39–41
  software boundary size, 45–47
  types of, 39–40
Cohn, Mike, 24
collaboration mode, 9, 133, 135–137, 153–155
defined, 187
  and evolution of team topologies, 159–161
  and reverse Conway maneuver, 147–148
  right amount of collaboration, 153–154
  team behaviors for, 142–143
  and viable X-as-a-service interactions, 149
commodity systems, 18
communication
  focused, 24–26
  inter-team, 25–26, 27
  paths, 17
  structures, 4–8
  tool choice and, 27
  unnecessary, 24–26
communities of practice vs. enabling teams, 90
compatible support systems, 69
complicated-subsystems teams, 9, 28, 91–92
  case study, 94–95, 97–99
  defined, 187
  expected behaviors, 94
  platform composition, 96–97
component teams, 28, 105–106
continuous delivery (CD), 11
cross-team testing, 23
database administrator team, 18–19
domain complexity, xxi, 187
domain identification, 43
domain limitations, 42–45
Drucker, Peter, 170
Dunbar, Robin, 32
Dunbar’s number, 32, 187
Dynamic Reteaming (Helfand), 48, 118, 150
Eckstein, Jutta, 86, 92
ecosystem tuning, 46
enabling teams, 9, 86–90, 106–107
  case study, 88–90
  vs. communities of practice, 90
monolithic shared databases, 16
naive uses of, 26–28
organization design, 16–17, 23–24
reorganizations, 28
reverse Conway maneuver, 10, 18–21,
  147–148, 188
software architecture as flows of change, 23
  team assignments, 22
  team-scoped flow, 21–23
  tool choice and communication, 27
  understanding and using, 15–17
  unnecessary communication, 24–26
  version compatibility, 22
Cornago, Fernando, 16
coupled releases, 113
credit reference agencies, 76–77
Cybernetics: Or Control and Communication in the
  Animal and the Machine (Wiener), xxii
defined, 187
expected behaviors, 87–88
engineering maturity, 73–74
environment design, 50
environmental scanning, 171
Ericsson, 68
Evans, Eric, 115
evolution triggers, 162–164, 165–170
delivery cadence slowing down, 166–167
multiple business services relying on one large set of underlying services, 167–169
software too large for one team, 165–166
expected behaviors
complicated-subsystems teams, 94
enabling teams, 87–88
stream-aligned teams, 85–86
“Eyes On, Hands Off,” 46
F
facilitating mode, 9, 134, 140–144
defined, 187
and reverse Conway maneuver, 147–148
feature teams, 67–68
*The Five Dysfunctions of a Team: A Leadership Fable* (Lencioni), xx
flow, enhancing, 148–151
flow of change, 12, 63–64, 187
focused communication, 24–26
Forrester, Russ, xxii
Forsgren, Nicole, 18, 64, 86
four team types, 178–179
Fowler, Martin, 115–116
fracture planes, 115–123
business domain bounded context, 115–116
case study, 121–125
change cadence, 117
defined, 187
natural, 121
performance isolation, 119
regulatory compliance, 116–117
risk, 118–119
technology, 120
user personas, 120–121
Fried, Jason, 55
G
germane, 40
Google, 70
Google Cloud, 72
group chat prefixes, 55–56
*Guide to Organisation Design* (Stanford), 38
guilds, 49
H
Hansson, David Heinemeier, 55
Harvard Business School, 16
healthcare organizations, 75–76
Helfand, Heidi, 48, 118, 150
Hering, Mirco, 120
heuristics for domain assignment, 43
hidden monoliths, 112–114
high cohesion, 22
high-trust organizations, 32
homomorphic force, 10, 19–20
Horizons, 36
Hotchkiss, Dave, 157–159
“How Do Committees Invent?” (Conway), 9–10
Humble, Jez, 18, 36, 64, 86
Humphrey, Andy, 53–55, 97–99
I
IBM, 146–147
IBM 8086 processor, 101
Ikea, 47
influences for book, xxii–xxii
infrastructure automation, 11
infrastructure teams, 105
ING Netherlands, 53
Ingles, Paul, 155
interaction mode key, xx
interaction modes, 179, 185
intermittent collaboration, 133
Internet of Things (IoT), 84, 101, 123–125, 156–157, 171
inter-team communication, 25–26, 27
Ivarsson, Anders, 50
J
Java Virtual Machine, 101
Jay, Graylin, 41
joined-at-the-database monolith, 113, 188
K
Kelly, Allan, 10, 24, 35, 101, 148
Kim, Gene, 18, 64, 86, 172
Kim, Kyung Hee, 136, 138
Kniberg, Henrik, 25, 50, 100
Knight, Pamela, xxii, 36
Kotte, Gustaf Nilsson, 47
Kotter, John, 161
L
Lambert, Michael, 50–52
“A Leader’s Framework for Decision Making” (Snowden and Boone), xx
*Lean Enterprise* (Humble, Molesky, & O’Reilly), 36
Lencioni, Patrick, xxii
Lewis, James, xxii, 10, 34
Linux, 101
Lister, Timothy, 38
log-aggregation tools, 27–28
loose coupling, 22
Luo, Jiao, 80
M
MacCormack, Alan, 16
Maibaum, Michael, 94–95, 162–164
Make Space* (Doorley & Witthoft), 53
Malan, Ruth, xx, 17, 23, 148
McChrystal, Stanley, 31, 46
Microsoft, 69
Millett, Scott, 116
Minick, Eric, 146–147
misplaced boundaries, 150–151
missing capabilities, 150–151
“A Model for Team-Based Organization Performance” (Forrester and Drexler), xx
The Modern Firm (Roberts), 23
Molesky, Joanne, 36
“monolithic build, 188
monolithic model, 114, 188
monolithic rebuilds, 113
monolithic release, 113, 188
monolithic shared databases, 16
monolithic thinking, 114, 188
monolithic workplace, 114, 188
monoliths, 112–114
application, 113, 187
hidden, 112–114
joined-at-the-database, 113, 188
multi-layer viable-systems model, 103
The Mythical Man-Month (Brooks), 35

N
Narayan, Sriram, 172, 173
.Net Framework, 101
Neumark, Peter, 92–93
The New Hacker’s Dictionary (Raymond), 10
new practices, adoption of, 155–159
Nokia, 38–39
non-blocking dependencies, 68–69
Nygard, Michael, 22, 116

O
onion concept, 34
open-plan office, 114
Ops team, 80–81
O’Reilly, Barry, 36
org chart thinking, 3–14
bottlenecks, 11–12
cognitive load, 11–12
collaboration mode, 9
communication structures, 4–8
complicated-subsystem teams, 9
Conway’s law, 9–11
enabling teams, 9
facilitating mode, 9
platform teams, 9
problems with, 5–7
stream-aligned teams, 9
team interaction modes, 9
Team Topologies model, 9
team types, 9
thinking beyond, 7–8
X-as-a-Service mode, 9
organization design, 16–17, 23–24
organization design evolution, 181
organization size, 73–74
organizational sensing, 64–65, 153–175
adoption of new practices, 155–159
business as usual teams, 173–174
business process management, 169
case study, 154–155, 157–159, 162–164
collaboration mode, 153–155, 159–161
environmental scanning, 171
evolution triggers, 165–170
rapid learning, 155–159
self-steering design and development, 170–174
team topologies combination, 164–165
team topologies evolution, 159–164, 165–170
X-as-a-Service mode, 153–154, 161
Out of the Crisis (Deming), 38
OutSystems, 11, 42

P
Pais, Manuel, 66
Payment Card Industry Data Security Standard (PCI DSS), 117
Pearce, Jo, 40
Peopleware (DeMarco & Lister), 38
performance isolation, 119
The Phoenix Project, 166
Fierce, Robert A., 136, 138
Pink, Dan, 11
Pivotal, 149–150
Pivotal Cloud Foundry (PCF), 48–49, 101
platform composition, 96–99
platform teams, 9, 92–96, 105–106, 188
platforms, 100–104
cognitive load reduction, 101–102
constraints, 102
managed as a live product or service, 103–104
multi-layer viable-systems model, 103
product development acceleration, 101–102
thinnest viable, 101, 184
underlying, 102–103
Poole, Marshall Scott, xix
Poppulo, 121–123
Prezi, 92–93
principle of overlapping measurement, 143
The Principles of Product Development Flow (Reinertsen), 23, 143
product teams, 68–69
Project Myopia (Kelly), 35
promise theory, 142

R
rapid learning, 155–159
Rautert, Markus, 16
Raymond, Eric, 10
“Real Life Agile Scaling” (Kniberg), 25
rebuild everything, 113
Red Hat Open Innovation Labs, 53
regulatory compliance, 116–117
Reinertsen, Don, 23, 143
relative domain complexity, 41–42
Remote: Office Not Required (Fried & Hansson), 55
Rensin, Dave, 72
reorganizations, 28
responsibilities, splitting, 74
responsibility restriction, 39–41
risk, 118–119
Roberts, John, 23
Rother, Mike, 134
Rubio, Andy, 50–52
self-service capabilities, 69
self-steering design and development, 170–174
Sheehan, Stephanie, 121–123
shuffling team members, 62
silo reduction, 74
single view of the world, 114
site reliability engineering (SRE), 70–72
Skelton, Matthew, 66
Sky Betting & Gaming, 94–95, 162–164
Snowden, Dave, xix
software architecture as flows of change, 23
software boundaries, 115–123
business domain bounded context, 115–116
case study, 121–125
change cadence, 117
defined, 187
natural, 121
performance isolation, 119
regulatory compliance, 116–117
risk, 118–119
team location, 118
technology, 120
user personas, 120–121
software boundary size, 45–47
software ownership, 36–37
software scale, 73–74
Sosa, Manuel, 24
Spotify, 49, 50, 75
squads, 50, 75
standardization, 114
Stanford, Naomi, 24, 38, 171
static topologies, 61–78
ad hoc team design, 62
anti-patterns, 62
case study, 75–77
cloud teams, 69–70
compatible support systems, 69
credit reference agencies, 76–77
dependencies, 74–75
DevOps, 65–67
DevOps Topologies, 66–67, 75–77
engineering maturity, 73–74
feature teams, 67–68
flow of change, designing for, 63–64
healthcare organizations, 75–76
non-blocking dependencies, 68–69
organization size, 73–74
organizational sensing, 64–65
product teams, 68–69
self-service capabilities, 69
shuffling team members, 62
silo reduction, 74
site reliability engineering, 70–72
software scale, 73–74
splitting responsibilities, 74
team intercommunication, 64–65
team patterns, successful, 67–72
technical and cultural maturity, 72–73
topology choice considerations, 72–75
wait time between teams, 74–75
stream-aligned teams, 9, 81–86, 104, 188
capabilities within, 83–84
case study, 82–83
expected behaviors, 85–86
streams of change, suitable, 183–184
support teams, 80–81, 107–109
Sussna, Jeff, 85, 161, 172
Sweller, John, 39–40
team APIs, 47–56
benched bay approach, 51
case study, 50–52, 53–55
defined, 48, 188
environment design, 50
group chat prefixes, 55–56
guilds, 49
squads, 50
team interactions, 49
virtual environment design, 53–56
workspace design, 50–56
team assignments, 22
team behaviors, 141–144
team habits, 134–135
team interaction modes, 9, 131–152
awkward team interactions, 150–151
basic team organization, 146–148
case study, 146–147
choosing suitable, 143–145
collaboration mode, 133, 135–137, 142–143,
147–148, 149
effective APIs, 148
enhancing flow, 148–151
facilitating mode, 134, 140–144, 147–148
intermittent collaboration, 133
misplaced boundaries, 150–151
missing capabilities, 150–151
principle of overlapping measurement, 143
promise theory, 142
reducing uncertainty, 148–151
reverse Conway maneuver, 147–148
team behaviors for, 141–144
team habits, 134–135
temporary changes to, 149–150
well-defined team interactions, 132–133
X-as-a-Service mode, 133, 137–140, 143, 149
team interactions, 49, 132–133
team intercommunication, 64–65
team lifespans, 35–36
team location, 118
Team of Teams (McChrystal), 31, 46
team patterns, successful, 67–72
cloud teams, 69–70
compatible support systems, 69
feature teams, 67–68
non-blocking dependencies, 68–69
product teams, 68–69
self-service capabilities, 69
site reliability engineering, 70–72
team silos, 99–100
team size, 32
team topologies
capability gaps, 184–185
combining, 164–165
component teams to platform teams, 105–106
converting architecture and architects, 109
converting common to fundamental team
topologies, 104–109
Conway’s law, 15–29, 180–181
defined, 188
evolving, 159–170
four fundamental. See team types
four team types, 178–179
how to get started with, 183–185
infrastructure teams to platform teams, 105
interaction modes, sharing and practicing, 185
org chart thinking, 3–14
organization design evolution, 181
organizational sensing, 153–175
static, 61–78
stream-aligned teams, 104
suitable streams of change, 183–184
support teams, 107–109
team interaction modes, 131–152
team types, 79–110
team-first boundaries, 111–126
team-first thinking, 31–57, 179–180
thinnest viable platform, 184
three interaction modes, 179
tooling teams to enabling teams, 106–107
team topologies combination, 164–165
team topologies evolution, 159–170
adopting different interaction modes, 159–162
case study, 162–164
combining team topologies, 164–165
delivery cadence slowing down, 166–167
evolution triggers, 165–170
multiple business services relying on one large
set of underlying services, 167–170
software too large for one team, 165–166
Team Topologies model, 9
team types, 9, 79–110
case study, 82–83, 88–90
complicated-subsystems teams, 91–92
converting common to fundamental team
topologies, 104–109
enabling teams, 86–90
enabling teams vs. communities of practice, 90
four, 178–179
key, xx
Ops team, 80–81
platform composition, 96–99
platform teams, 92–96
platforms, 100–104
stream-aligned teams, 81–86
support team, 80–81
team silos, 99–100
team-first boundaries, 111–126
application monolith, 113
business domain bounded context, 115–116
case study, 121–125
change cadence, 117
coupled releases, 113
distributed monolith, 112
fracture planes, 115–123
hidden monoliths, 112–114
joined-at-the-database monolith, 113
in manufacturing, 123–125
monolithic model, 114
monolithic rebuilds, 113
monolithic releases, 113
monolithic thinking, 114
monolithic workplace, 114
open-plan office, 114
performance isolation, 119
rebuild everything, 113
regulatory compliance, 116–117
risk, 118–119
single view of the world, 114
software boundaries, 115–123
standardization, 114
team location, 118
technology, 120
user personas, 120–121
team-first mindset, 37–38
team-first software architecture, 35
team-first thinking, 31–57, 179–180
benched bay approach, 51
boundaries, 39–47
case study, 50–52, 53–55
cognitive load, 39–47
diversity, 38
domain identification, 43
domain limitations, 42–45
Dunbar’s number, 32
ecosystem tuning, 46
environment design, 50
extraneous, 40
“Eyes On, Hands Off,” 46
germane, 40
guilds, 49
heuristics for domain assignment, 43
high-trust organizations, 32
horizons, 36
intrinsic, 40
onion concept, 34
relative domain complexity, 41–42
responsibility restriction, 39–41
small long-lived teams, 32–39
software boundary size, 45–47
software ownership, 36–37
squads, 50
team APIs, 47–56
team definition, 32
team interactions, 49
team lifespans, 35–36
team size, 32
team-first mindset, 37–38
team-first software architecture, 35
trust and team size, 33–35
Tuckman Teal Performance Model, 36
types of, 39–40
virtual environment design, 53–56
workspace design, 50–56
teams, small long-lived, 32–39
diversity, 38
Dunbar’s number, 32
high-trust organizations, 32
horizons, 36
onion concept, 34
software ownership, 36–37
team definition, 32
team lifespans, 35–36
team size, 32
team-first mindset, 37–38
team-first software architecture, 35
trust and team size, 33–35
Tuckman Teal Performance Model, 36
team-scoped flow, 21–23
technical and cultural maturity, 72–73
“Technical Consulting Teams,” 86
technology for team-first boundaries, 120
Thinking Environments, 4
thinnest viable platform (TVP), 101, 184, 188
Thoughtworks, 10
tool choice and communication, 27
tooling teams, 106–107
topology choice considerations, 72–75
dependencies, 74–75
engineering maturity, 73–74
organization size, 73–74
silo reduction, 74
software scale, 73–74
splitting responsibilities, 74
technical and cultural maturity, 72–73
topology choice considerations, 72–75
wait time between teams, 74–75
Toyota, 134
TransUnion, 76–77, 157–159
Treynor, Ben, 70
tribes, 75
trust and team size, 33–35
Tuckman, Bruce, xx
Tuckman Teal Performance Model, 36
Tune, Nick, 116

user personas, 120–121
uSwitch, 155

version compatibility, 22
virtual environment design, 53–56
Vogels, Werner, 83

wait time between teams, 74–75
Watson, Ian, 76–77
Weston, Robin, 88–90
Whyte, Dave, 53–55, 97–99
Wiener, Norbert, xix
Wiley, Evan, 48–49, 149–150
Windows, 101
Witthoft, Scott, 53
workspace design, 50–56

X
X-as-a-Service mode, 9, 133, 137–140
and collaboration mode, 149, 153–154
defined, 188
and delivery predictability, 160–161
and evolution of team topologies, 159–161
team behaviors for, 143

uncertainty, reducing, 148–151
unnecessary communication, 24–26
Writing a book is a collaborative task involving many people without whom the finished book wouldn’t be impossible. We’d like to thank our peer reviewers for taking the time to give detailed feedback on the book: Charles T. Betz, Jeremy Brown, Joanne Molesky, Nick Tune, and Ruth Malan. We also want to thank the authors and originators of our case studies and industry examples: Albert Bertilsson, Anders Ivarsson, Andy Humphrey, Andy Rubio, Damien Daly, Dave Hotchkiss, Dave Whyte, Eric Minick, Fernando Cornago, Gustaf Nilsson Kotte, Henrik Kniberg, Ian Watson, Markus Rautert, Michael Lambert, Michael Maibaum, Miguel Antunes, Paul Ingles, Pulak Agrawal, Robin Weston, Stephanie Sheehan, and Wolfgang John.

We’d like to thank everyone who contributed to the original DevOps Topologies patterns, especially James Betteley, Jamie Buchanan, John Clapham, Kevin Hinde, and Matt Franz. A special thanks goes to John Cutler for a passionate outsider’s view of the Team Topologies approach, and to Gareth Rushgrove for helping to expand the audience for the original DevOps Topologies patterns. We also want to thank our colleague Jovile Bartkeviciute at Conflux for her tireless research.

The team at IT Revolution Press has been amazing, especially Anna Noak, Lean Brown, and the other editors and designers—we’ve really valued their advice, support, and infectious enthusiasm. We’re grateful to Gene Kim for inviting us to speak at DevOps Enterprise Summit in London in 2017, which helped us to realize the value of the emerging Team Topologies ideas.

Finally, we’d like to thank the people whose ideas, talks, and writing inspired us to become interested in the fascinating relationship between teams and software in the first place, and helped to make this book a reality: Allan Kelly, Andy Longshaw, Charles T. Betz, Donella Meadows, James Lewis, Gene Kim, Mel Conway, Mirco Hering, Rachel Laycock, Ruth Malan, and Randy Shoup.
Matthew Skelton has been building, deploying, and operating commercial software systems since 1998, and has worked for organizations including London Stock Exchange, GlaxoSmithKline, FT.com, LexisNexis, and the UK government. Head of Consulting at Conflux (confluxdigital.net), Matthew is the co-author of the books Continuous Delivery with Windows and .NET (2016) and Team Guide to Software Operability (2016). Matthew holds a BSc in computer science and cybernetics from the University of Reading, an MSc in neuroscience from the University of Oxford, and an MA in music from the Open University; he is a chartered engineer (CEng) in the UK. In his free time, Matthew plays the trumpet, sings in choirs, writes music, and enjoys trail running.

Manuel Pais is an independent DevOps and Continuous Delivery Consultant focused on team design, practices, and flow. He helps organizations define and adopt DevOps and Continuous Delivery (both from technical and human perspectives) via strategic assessments, practical workshops, and coaching. Manuel is the co-author of Team Guide to Software Releasability (2018).

Matthew and Manuel have worked together on organization design for modern software systems with many clients around the world. Their training sessions on organization design for modern software systems have helped numerous organizations to rethink their approach to team intercommunication and software architecture, improving flow and the effectiveness of software delivery.