Organizing Toward Agility

ADVANCE READING COPY

UNCORRECTED PROOF

NOT FOR SALE

Design, Grow, and Sustain

Self-Organizing Structure at Scale

Jeff Anderson

Organizing Toward Agility

Organizing Toward Agility

Design, Grow, and Sustain Self Organizing Structure at Scale

Jeff Anderson

Agile By Design
Publications

Published by Agile By Design Publications, Toronto, ON
agilebydesign.com

GIRL FRIDAY PRODUCTIONS®
Edited and designed by Girl Friday Productions
www.girlfridayproductions.com

Cover design: Kathleen Lynch
Project management: Kristin Duran
Project editorial: Bethany Davis
Illustrations: Tom Holmes

ISBN (paperback): 978-1-7780930-0-5
ISBN (e-book): 978-1-7780930-1-2

First edition

Contents

Introduction

How we tend to think about organizations is wrong. Completely wrong.

The way we traditionally design organizations, the way we go about managing our people in those organizations, and the structures within which we operate—all are based on concepts that are more than a hundred years old. The traditional organization—one made up of departments, steered by management, working toward long-term plans, and operating based on a fixed playbook—is built to meet the needs of the industrial age, not the digital age. The traditional organization is a legacy organization, whose goal was to drive down the price of production and achieve economies of scale through the principles of division, standardization, and control.

But the world looks very different today. We are in a hypercompetitive global economy. Markets are both crowded and quick. Users are fickle, and customers are differentiated and exacting in their needs. Trying to solve the problems of today while being organized for the problems of yesterday is a recipe for frustration and even misery. New problems require new ways of getting organized, a new approach to thinking about how we collaborate.

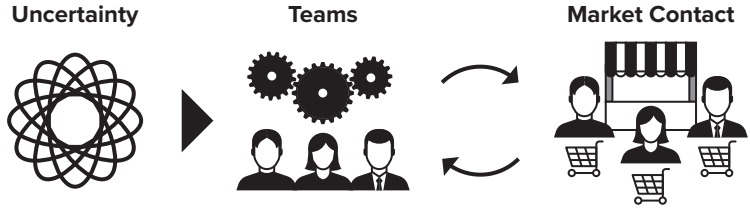
Most people struggle and strain against their organizing structure in order to deliver value. Many organizations are saddled with bureaucracy that inhibits value while doing little to reduce risk. But it doesn't have to be this way. We can organize ourselves in ways that increase our ability to deliver value. We can harness a new management system based on organizing principles that emphasize our humanity rather than belittle it.

The goal of this book is to offer a new way to create value at enterprise scale, one inspired by Agile thinking. My aim is to provide methods and examples that show how ideas like self-managing teams, direct market access, and continuous feedback can be expanded upon to function in an enterprise setting. This book will give you practical tools to help you set up and operate new organizing structures, whether you're looking to make a larger organizational change or to move your organization forward incrementally. It will help you organize toward a state of greater agility, and organize away from the era of industrial thinking.

Throughout this book, I'll use the term *Agile* as a proper noun. People in the industry often use it to refer to the collection of practices, principles, and concepts that were inspired by the Agile Manifesto, a body of knowledge that lays out a team-centric, iterative approach to delivering value using customer feedback and continuous experimentation. Agile comes in many flavors and methodologies—for instance, scrum, Kanban, SAFe, and Extreme Programming—and while they each differ in their specifics, they are all based on a common set of principles.

When I use this term, I will be putting this definition aside for a bit. It's helpful to understand Agile in terms of its history and its body of knowledge, but I don't think it's essential. For the most part when I use the term, I'll be referring to a simpler definition:

Agile is a strategy to help organizations that face increasing market uncertainty by forming self-managing teams that have the autonomy to decide how value is created for their customers, enabling those teams to achieve their outcomes through intimate, direct, and frequent market contact.



I will also often refer to Agile-powered teams, or simply Agile teams, in this book. I'll define *Agile teams* as teams that are pursuing their path toward greater self-management through the use of Agile principles, practices, and methods, something I'll cover in greater detail later on.

A good deal of content has already been written on the subject of how to organize around Agile teams. Having spent more than a decade working in this area, I have had to pull from a lot of different sources to help educate others. My team[1] and I have integrated and extended a lot of this material into something that is cohesive and practical. With this book, I want to take all of the different ideas and experiences that my team has used over the past decade and put them under one roof, presenting them in one consolidated package.

Unlike some other writings on this subject, this book will not present a magic, out-of-the-box solution that you can use to suddenly create agility within your organization. This book won't try to sell you on the idea that scaling comes from bloating your processes or from applying a more complicated methodology. Instead, this book will present a set of practical thinking and facilitation tools to allow you to take concepts that work at an intimate scale and expand them to support organizing at multiple scales. In a nutshell, this book will provide concrete tangible advice, without falling into the trap of being prescriptive in terms of method or dogmatic in terms of process.

The book is written so that the reader is taken through simpler concepts that apply to smaller scales at first. These concepts are then elaborated and expanded upon as the book progresses. A number of real-world examples are used to articulate various concepts and practices.

The first part of the book focuses on the quintessential building block of organizing toward agility: the team. These chapters explain why teams are important and how Agile helps create more effective teams. I will introduce you to the idea of teaming—the act of engaging in outcome-focused, high-collaboration activity in a way that builds lasting social cohesion over time.

Next, we will look at the organizational ecosystem in which teams

1. https://agilebydesign.com/about_us.

flourish (or perish). Despite what some of the literature will tell you, no team exists as an island. There will always be some sort of larger structure of which each team is a part. Ecosystems are one approach, which act as a thin container to demarcate groups of teams that share a common mission or overarching outcomes. This part of the book talks about how to define, operate, and improve your ecosystems, and how to take Agile concepts and adapt them to larger concepts.

Finally, we'll look at the whole enterprise. As many an Agile practitioner will tell you, you can only go so far in terms of increased agility at the team—or even team-of-team—level. This part of the book goes over various design practices and thinking tools you can use when rethinking how to lay out your organization in a way that promotes self-organization at scale. It provides a different path for organizational designers than the typical building blocks traditionally used.

Sources of Inspiration

Before we get into the book, I want to acknowledge the reality that I am standing on the shoulders of giants with this work. Very little of this book was invented by me; I have borrowed a lot. Below is a list of some of the sources that have encouraged this work.

- David J. Anderson's Kanban method,[2] specifically using Kanban to perform long-term planning with an eye toward managing the dynamic formation of feature teams and allocating specialist pools.
- The works of Donald Reinertsen,[3] whose principles of flow shape many of my thoughts, but especially the idea that process and organizational structure can be defined based on reusable patterns, like the classic GoF (Gang-of-Four) patterns for programming.
- Jurgen Appelo's management books,[4] especially his thinking on flatter organizations, wide job functions with narrow roles, and informal leadership.
- The excellent book *Organize for Complexity* by Niels Pflaeging.[5] The book is an awesome read for a number of reasons. This text has helped me shift my conceptual model of how to describe various organizational structures.

2. https://kanban.university/kanban-guide/.
3. http://reinertsenassociates.com/books.
4. https://jurgenappelo.com.
5. https://www.nielspflaeging.com/books.

- The book *Corporate Rebels*, by Joost Minnaar and Pim de Morree[6], which is a fantastic treatise on the common principles that progressive organizations use to achieve self-management. *Corporate Rebels* is a true inspiration for people looking for real-world examples of how to move organizations into the modern era.
- The O'Reilly books *Microservice Architecture*[7] and *Building Microservices,*[8] specifically where they intersect with concepts from Eric Evans's *Domain-Driven Designs*[9] (e.g., bounded contexts, domain aggregates, and context integration patterns to help teams align to solution architecture).
- *Team Topologies*.[10] I actually was almost done with my book when I came across this excellent text. This book has many complementary, adjacent concepts, and even coincidentally identical terminology to what I present here. Great minds think alike. :) *Team Topologies* has helped me articulate some of my thinking around domain-oriented organizations.

The concepts from all these sources work amazingly well together, and this book attempts to tie them together. The book represents almost two decades of experience of our team working with clients, trying these concepts, and learning from them.

Hopefully the ideas you are about to learn will accelerate your learning journey and offer an approachable path to getting a handle on some of the amazing concepts from the above content. I think these concepts can be learned and even mastered (if mastery in this domain can really exist) in a relatively short period of time, and I want this work to be helpful.

---

6. https://www.corporaterebelsbook.com.
7. https://www.oreilly.com/library/view/microservice-architecture/9781491956328.
8. http://shop.oreilly.com/product/0636920033158.do.
9. http://domainlanguage.com.
10. https://teamtopologies.com/?gclid=CjwKCAjw87SHBhBiEiwAukSeUS-KefW8SEWBsP- p11vjcd18JNS2MlLtcmFDxMEV31wZ9sA924QjTBoCg7AQAvD_BwE.

Part I

The Team

1

From Industry to Uncertainty

Why do so many organizations seem so dysfunctional? It starts with how businesses view their own people.

The predominant thinking about how people ought to be managed is more than a hundred years old. The traditional organization is based on the idea that people can be treated like components in a machine. Despite the rise and fall of new management fads in recent years, the way we believe organizations ought to be structured has not fundamentally changed for over a century.

THE INDUSTRIAL ERA

Before the industrial era, when goods could suddenly be mass-produced on a scale never before dreamt of, *people* played the primary role in creating goods for others. Whether with the local tinsmith or the neighboring cooper, placing an order for your unique needs was met with familiar, even intimate service. The production of these custom goods, however, was expensive and therefore unavailable to the masses.

The resounding challenge of this preindustrial era was to distribute a limited number of goods to as much of the population as possible. As

decades passed, factors like decreasing death rates, immigration, and advancements in technology encouraged people to increase consumption. A new and growing population wanted to attain a better quality of life. We needed to get goods out to the masses, and we needed to do it for the lowest possible price.

This increase of economic demand gave rise to an industrial management model that imposed a top-down hierarchy. Organizations scaled based on the principles of standardization, division, and control. Henry Ford's Model Ts launched off the assembly line by way of these principles. Eventually, this top-down model became the dominant mode for almost all organizations.

The industrial management model is geared toward attaining high efficiency. Workers were divided according to highly specialized tasks. People with separate skill sets were placed in separate parts of the organization. Repetitive tasks were documented into detailed instructions that were carried out, to the letter, by workers. Managers were responsible for defining these tasks and held accountable for their orderly execution.

The increased production of the industrial era significantly raised the standards of living for us all. Cars, radios, ovens, refrigeration, vacuum cleaners—you name it. A nice side effect of mass production was that we needed to employ this population of consumers; someone had to make these goods, after all. A balance became evident: Everyone got a job and the money they made was fed back into the same industrial machine that employed them. We had a phenomenon of give and take, a novel and progressive economic life cycle.

Indeed, standardization, division, and control were fit for purpose for organizations that wanted to create wealth on a widespread scale. More wealth was generated in those hundred years than in the previous thousand. This wealth was enjoyed not only by the familiar few—the colloquial fat cats at the top of the socioeconomic pecking order (although, yes, they did do fantastically well)—but also by a much broader audience of previously poor and uneducated workers. A new middle class was born.

THE ERA OF CHANGE

The world of today looks very different from the world at the turn of the past century. Change is the name of the day: constant change, rapid change, exponential change, changing change. The only constant seems to be that the rate of change keeps accelerating.

It would not be a stretch to say that the industrial era management model has become toxic in our current socioeconomic environment. In many ways, industrial era thinking is eroding our ability to effectively deliver value. As the industrial organization gained prominence, the *machine* became the engine that delivered value. Humans were merely cogs in this machine. Automatons. Parts that could be replaced with cheaper humans, or automated away with technology, the *real* machines.

While this was great for that initial burst of wealth, it wasn't great for our humanity. Thinking about humans as machines is also a lousy way to take advantage of people's innate ability to solve novel problems.

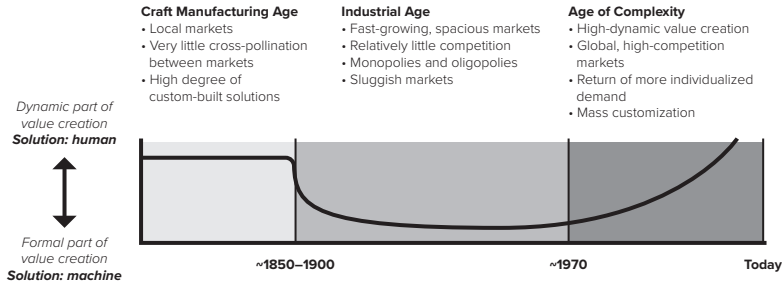


Illustration originally from *Organize for Complexity* by Niels Pflaeging

Humans are in charge again (for now)

Thanks to software, the vast majority of work is now knowledge work. So we have now gone full circle: from human, to machine, and now back to human.[11]

At their simplest, organizations exist to help people collectively create more value than they could by themselves. Yet the industrial

11. Niels Pflaeging, *Organize for Complexity: How to Get Life Back into Work to Build the High-Performance Organization*, p. 6.

principles of standardization, division, and control have increasingly created the opposite effect. How many of us feel that our organizations make us collectively dumber and slower? How many of us bemoan the negative impact our organizations have on the world or the loss of humanity we feel from working for them?

It turns out that the industrial organization is a victim of its own success. When customers become wealthier, they also become much more discerning about what they want to buy. Wealth brings with it a lot more competitors, all of whom vie for the customer's dollar. As more and more suppliers compete for that spending, product life cycles become shorter. And shorter. And shorter again. Planned obsolescence, especially in the tech space, can come in months, weeks, and even days. If you were an early adopter of Apple's iPhone, I would guess that within the past ten years you have likely purchased three to five phones. Compare that with your home landline phone you may have grown up with; my household had the same kitchen phone for more than fifteen years.

The examples of our new world are endless. From theaters to streaming, from file cabinets to the cloud, the sharing economy, the gig economy, digital everything—the world continues to go through waves of unprecedented change. And people are not looking back.

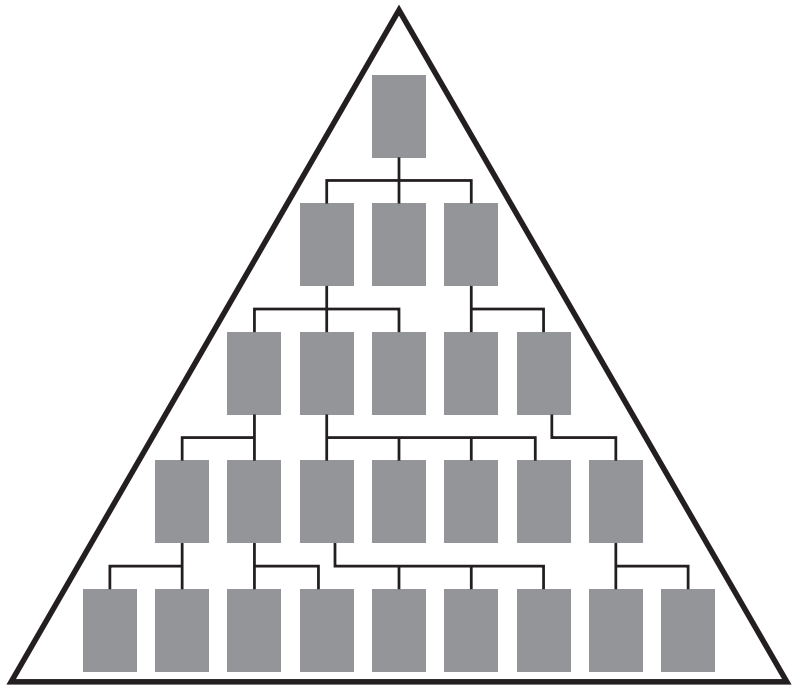
All of this innovation is making it feasible for customers to ask for products personalized to their exact needs, wants, and tastes. So that design-it-once-and-repeat-for-everyone idea is pretty much out the window. We are long past the point where markets are vast with spacious amounts of room to easily grow simply by doing more of the same. Markets are now highly segmented.

This all adds up to an era typified by volatility, uncertainty, complexity, and ambiguity—by VUCA, a term first introduced by the US Army War College in response to the "multilateral world perceived as resulting from the end of the Cold War." VUCA has now been repurposed toward strategic leadership theories focused on discussing the great changes organizations face. The challenge of today's era, thus far, is to use market learning to provide differentiated solutions that create customer delight and loyalty to an exact and exacting audience.

INDUSTRIAL ERA ORGANIZATIONS VERSUS MODERN ORGANIZATIONS

So how, specifically, can today's organizations set themselves up for success in today's changing world?

Organization in the industrial era, when looked at from a distance, looks like a pyramid, with direction flowing from the top and information flowing from the bottom. The emphasis is on command and control: Directives come from the top of the organization and are executed by lower levels in the hierarchy. Information resulting from activity within the lower levels of the organization are then fed back to the top levels of the organization. In turn, the top of the organization then issues its next set of directives.



We are of the triangle

There are some fundamental issues with the industrial era organization in the face of market uncertainty. As uncertainty increases:

- The top of the hierarchy doesn't have the context required to make the right decisions. The **wrong decisions** are made.
- It takes too long for the top of the hierarchy to process information from lower levels of the organization in order to make decisions in a timely fashion. It takes **too long to react** to market change.
- Accountability is taken away from the lower levels of the organization. The organization is **robbed of a wealth of intelligence and creativity**.
- Siloes form under distinct leaders at the top of the hierarchy. Internal, **departmental objectives take more focus** than market outcomes do.

Organizing for the modern world requires us to decentralize authority into teams responsible for doing the work. We group these people into cross-functional teams that have all the skills and permission they need to operate with the autonomy required to own their market outcomes. Instead of relying on one-size-fits-all standards, teams continually iterate and improve based on frequent market feedback. The operating metaphor is no longer the corporate hierarchy. Rather, we think of people and teams as being part of a dynamic value network that flexes based on the changes observed in the market.

For this to succeed, we need a strong belief that most people want to work and that, under the right conditions, they enjoy doing so. We also need a strong belief that when teams with diverse skills are exposed to market feedback, they will make better decisions than their bosses, who possess limited context in the face of growing complexity.

Of course, shifting toward a more people-centric organization cannot be done lightly. It requires work. Work from your positional leaders and work from every person across your entire organization. New practices and processes can illuminate parts of the path, but this transition really starts with a shift in mindset; this takes time.

The good news is we have plenty of examples out there to draw inspiration from. Your path will be unique to you, but that doesn't mean you can't learn from others. The other good news is that this new way of thinking and working already exists in your organization. If it

didn't, you wouldn't already be providing value in a complex world.

The problem is that your value network is (or has been) at odds with the traditional organizational structure and management system. Your value network is working *in spite of* your management structure, not *because of* it. It is running against the "official" part of your organization.

Many pundits believe that communism in Russia has lasted so long because of a vibrant black market that kept goods flowing across market participants. In the same vein, your organization is functioning because you already have a value network of self-organizing knowledge workers; you just need to uncover it, make it official, and then remove the structural obstacles.

THE WAY WE ORGANIZE IS CHANGING . . . *SLOWLY*

You would think, with our current environment and the many resulting social and technological innovations, that the industrial era organizational model would already be a thing of the past. You would think that most of us would be working in a very different type of organization by now. You would think that many of us would have an insatiable need to move beyond industrial era organizations. You would think that all levels of our organizations would be yearning for a change.

You would think.

However, most organizations are still operating—dare I say, lugging along—like industrial era machines. The majority of leaders and their teams are still constantly fighting their management systems and organizational structures to deliver value. The mindset, skills, and the will to move to a new way of organizing seem to be in shorter supply than they need to be.

But change is happening in many organizations. Zappos, Netflix, Airbnb, and other modern companies are all examples of organizations that have established a different way of working. Some organizations have gone further than your typical tech-product company has. Companies like Monsanto, W. L. Gore, Buurtzorg, Handelsbanken, and Patagonia have taken even more significant steps toward empowering people in the value creation process.

Shifting our organizations from a machine-centric to a people-centric model makes good business sense from the perspective of better capturing customer value, encouraging growth, and generating profit. This is all true, but it misses a very important point. Organizing around people allows us to consider how to create a truly *human* organization. People-centric institutions create meaningful work for their employees, and few things are more meaningful than achieving a higher purpose. As organizational leaders explore ways to organize around people's strengths, passions, and creativity, they begin to understand that higher purpose is a better motivator than profit, market success, or other factors that merely lead to corporate success. There is plenty of research that backs this up, with a recent paper in the *Journal of Business Ethics* showing that the motivation of employees improved between 17 and 33 percent when profit was not the primary objective.[12]

Growth, market share, competition, to name a few driving facets, are all necessary to run a large organization. However, none of these is sufficient alone for an organization and the people in it to excel. So, when embarking on the journey toward a more Agile organization—an organization that responds quickly to the changing needs of customers, and even society at large, through motivated and autonomous employees—consider the human reasons: *Why* are you doing so? When you connect the change to a higher calling—to concepts like stakeholder capitalism, closed-loop systems, sustainable development goals—then your organization will perform better. Conversely, if you keep the focus solely on profit and growth, it will not. When people organize around a strong purpose, then motivation, autonomy, and self-organization all become a matter of course.

IN SUMMARY

- Industrial organizations grew in response to the need to

12. Tu Yidong and Lu Xinxin, "How Ethical Leadership Influence Employees' Innovative Work Behavior: A Perspective of Intrinsic Motivation," *Journal of Business Ethics* 116, no. 2 (2013), https://www.researchgate.net/publication/257541870_how_ethical_leadership_influence_employees'_innovative_work_behavior_a_perspective_of_intrinsic_motivation.

generate wealth for a largely poor, uneducated population in markets that were spacious, sluggish, had little competition, and had huge potential for growth. The strategy of industrial management was cost containment through principles of division, standardization, and control to achieve these outcomes.

- Today's organizations are still largely based on industrial era thinking that is turning toxic in our current era of accelerated change. Discerning customers, segmented markets, crowded competition, shrinking product life cycles, and rapid technological innovation mean we once again need to rely on people and their problem-solving capabilities rather than on mindless machines.
- The modern organization is structured around decentralizing decision making to teams that have the autonomy to own outcomes through frequent market feedback. Moving toward a more people-centric organization requires that we promote trust, give up control, grant ownership, and overcome our fear of failure.
- The change to modern organizational models is highly synergistic when growing into a *human organization* focused on using profit and growth to make the world a better place. Your value network of people already exists where you work; you just need to make it your official, core organizational operating system.

2

Agile Teams

A little over five years ago, I worked with Sean Deschamps, a fellow coach at my consulting firm Agile By Design, to help Scotiabank's Global Payments group adopt Agile organizational principles on one of their projects.

The larger Scotiabank organization, numbering over eighty-eight thousand employees, was deep in the throes of an overarching transformation. A brand-new digital organization had been created by a mix of both business and technology professionals from across the bank. Hipster-friendly dwellings, expensive consultants, executive training—the works—were made available to this new digital organization.

If Scotiabank was spending a buck and then some on reorganization, this investment, for better or for worse, had not yet made its way to the payments group. This was apparent the first day Sean and I arrived at their downtown office, ready to start. Here are Sean's impressions of our first day:

> *My previous experience to date was working in a more entrepreneurial setting, small companies where people were passionate about what they did. This experience likely colored my perspective of the dire state of affairs I witnessed when I first arrived at the payments group*

> *head office. As I gazed across the empty walls and old cubicles, I noted a deafening silence. I wondered how this could be a place where work was getting done, especially the complex kind of work that required constant collaboration. This place felt like it had been defeated a long time ago, like the people had lost a war.*

This first impression was only reinforced as Sean and I spent more time getting the lay of the land.

The project required the use of numerous distinct systems, ranging from modern platforms to legacy systems. We needed people who knew how to design and deliver within a range of technologies that included more-modern platforms as well as creaky old legacy applications, some of which had originally been built before humans ever landed on the moon.

But everyone was functionally divided across both the business and technology parts of the payments organization, in a manner typical of large enterprises. There were departments for analysts, for testers, one developer department for each system, and so on. Some of these functions actually fell into an entirely different part of the bank, outside of the payments group. Of course, everyone was seated on separate floors (or even in separate buildings), according to this division of functional and system expertise. They also had both tech and business project management offices that operated as entirely separate, siloed, and often competing entities. This separateness was evident in the way people worked as well as the language they used. It seemed that every group had different terms for the same concepts: their work, their systems, their products, and their users.

Sean noticed a trend as we traveled across the various physical locations to gather context:

> *Every place we visited was stacked with cabinet drawers tightly packed into every space available, and these were overflowing with binders and reams upon reams of paper. We could imagine the arcane processes that led to the creation of these tomes of lost and forgotten artifacts. The sense of bareness I noticed on my first visit to*

> *the payments group was pervasive across the other sites we visited. We couldn't find many instances of people actively collaborating. It seemed that, outside of official meetings, things were being done largely in isolation.*

Sean and I felt it was time to bring some recognizable signs of life back to people's work. If not to the entire payments group, then at least we could start with the traces and recalls project. We wanted to tackle this sense of separateness, this feeling of dilapidation. It was time to take all the living, breathing people who were asked to deliver on this project and bring them together in a way that allowed them to work in a living, breathing environment. It was time to put people together into a team.

AGILE HELPS PEOPLE TO TEAM

In this age of uncertainty, the team is the organizing construct that is best positioned to create value. This means teams that are empowered to work directly with market actors to achieve real market outcomes, teams that are capable and empowered to get the job done with minimal interference.

In contrast to the sterile, even dead office environment we often see in the industrial world, the team environment is a living, breathing space where hypercollaboration and learning can thrive.

Working in a team is more motivating than working in a functional group. When people work in teams, we see better communication and coordination, more learning and growth, and stronger commitment. Team members are better able to solve complex problems than members who work in separate departments or on temporary projects. They are better able to generate new ideas and synthesize feedback and learning. Teams can gain the autonomy to self-organize around value creation.

When teams are properly nurtured, leaders can grant them an increasing amount of autonomy. Leaders can step away from the illusion of centralized control. We can decentralize decision making to the people closest to the real work. We can dismantle mind-numbing

bureaucracy.

One of the most compelling aspects of Agile is the wealth of concepts dedicated to getting a team up and running. Agile has a wealth of good practices that can help us operationalize the modern team, and it gives us a frame to help people in the team react to feedback and continue to grow and learn.

In the rest of this chapter, we will examine the key traits of Agile teams.

RADICAL TRANSPARENCY

Many Agile methods have the side effect of contributing to an environment of *radical transparency*. Visual backlogs, visual workflow, visual maps, and visual models all contribute to this radical transparency. One of the first promises of Agile is that we can shine a bright light on what often is a very opaque working world.

When one is thinking about how to transition to a team model, it is often valuable to start by bringing radical transparency to the problem and solution space. Modern work is not factory work. We (mostly) aren't processing physical goods. Our inventory is invisible, the work is often not repeatable, and our progress is hard to quantify. We call this type of work knowledge work. When we do knowledge work, it is like working in a fog. It can become impossible to understand what others are doing, the value they are creating, or the problems they face. This lack of common understanding kills agility. It just takes too long to share a common understanding. When it comes time to make a pivot, we all change in different directions.

Agile practices are great at shining a bright light, taking complicated concepts and representing them in a way that is digestible, that encourages participation and co-creation. It isn't necessary to use any particular Agile method. Choose the method(s) that will help you to illuminate your goal as well as to illuminate how you will solve it, why it is important, and the value your users will receive.

Here's what Sean remembered about his experience in getting started with the traces team:

The recalls solution, while simple from a customer and business perspective, involved a myriad of systems, ranging from modern platforms to legacy systems. Creating a common agreement and understanding of the work between all the different groups required would be challenging.

We decided to go with story mapping, a practice invented by Jeff Patton, to bring some clarity to this complicated problem space. We adapted this approach so that we could go through every customer flow and map out not only the user's observable activity but also the flow of behavior between the various front- and back-office systems, understanding how high-level outcomes mapped to small and testable increments of system behavior.

We then moved to trying to understand how to visualize how we can work together, and started by simply visualizing a flow of delivery that the various functional and system experts could recognize as their own. Using the Kanban method, popularized by David J. Anderson, we represented each system team's work using discrete swim lanes, and the involvement of the different functional experts as columns within the swim lanes. The biggest innovation we introduced to this flow of work was the idea of stories and thin slices, small increments of value that are a subset of the overall initiative scope, smaller chunks that could be defined and delivered independently of each other. We wanted people to flow these thin slices through their Kanban, with the goal of testing frequently, thus reducing the opportunity for different functional and system experts to work in separate siloes.

When we first started our visualization exercises, we were often met with skepticism, if not outright defensiveness. By focusing on bringing clarity to the work as they understood it, we were able to shift people's opinions.

Radical transparency is a great first step toward better behavior. It is harder to avoid uncomfortable truths, to be political, and to work on things that don't add value. Radical transparency makes the consequences of people's actions more obvious, both good and bad. It is important to note that radical transparency can make the need for a more cross-functional teaming approach more obvious, which is useful when buy-in for a cross-functional team is not particularly strong.

CROSS-FUNCTIONALITY

Most areas of enterprise, and especially IT, are rife with overspecialization. Separating testers from analysts, software designers from developers, back-end developers from front-end developers, or user experience (UX) from everything has more to do with following historical precedence and protecting people's egos than with doing what works.

Cross-functional teams are a shift away from traditional attitudes toward specialization. The idea is to equip each team with all the skills they need to achieve their outcome. Getting the right people on the team to enable autonomy can be deceptively difficult in some situations. We want people to be able to pinch-hit for each other, and to learn how to take on numerous roles. This is a balancing act; we still want deep, specialized expertise in many of our people, but we balance this with the need for people to gain adjacent skills.

Here's what Sean remembers about how the traces team evolved from a project to a more complete team:

> *We had a decent understanding of what we had to accomplish and how we wanted to accomplish it. But our team wasn't yet assembled; we needed a way to find the rest of the team. Our story map was particularly helpful in allowing us to tap into our core subject matter experts to source the rest of the team. We had color-coded every story according to a particular system skill set. This allowed us to estimate the work and pull in the required number of people for each system. An*

> *iterative process followed, using the story map to understand who to pull into the team and then showing the story map to the new team members. These new members then added their insight, which caused us to pull in additional team members. After a few iterations, we had (most of) our team!*

When we put teams together with diversity in mind, we bring distinct skill sets, ranges of experience, and a mixture of perspectives. Instead of using management to move work across departments, we bring the right skills to the team. Radical transparency helps us along the journey. Teams are able to operate with more autonomy and independence and with minimal impacts on other teams. This can make setting up a cross-functional team harder than setting up a functional group, but once the team is up and running, it will be able to run with a minimum of intervention.

FREQUENT MARKET FEEDBACK

One of the best ways to help a market-oriented team to be successful is to increase the frequency of market feedback. *Frequent market feedback* is the corrective mechanism teams use to stay on track. It is an unambiguous way to understand whether what is being built is leading to the right outcomes.

Teams can start down the path of increasing market feedback by using Agile techniques to thinly slice work so it can be delivered in smaller increments of value to the market. Agile also encourages teams to only work on a small number of those increments of value at a time. When a team works on a small number of small things, we get an earlier opportunity to validate that value of what we are working on with our market, increasing the frequency of feedback.

It can be powerful to think of market feedback from a perspective of safety. Positional leaders fearing that teams might self-organize toward market failure can review the market feedback being collected from the team. This allows leaders to shift from giving market direction to teams, to checking whether teams are reacting to the market

feedback they are collecting. Market feedback can be a powerful antidote to command and control.

At Scotiabank, Sean and I didn't want to get too far along in the process before getting feedback from internal stakeholders, so we used a thin-slicing approach to test our solutions. By accident, we quickly discovered that some of our original ideas didn't really add much value to the end process and were able to remove a lot of the scope from the overall initiative.

TEAM SPACE

The team concept cannot work if individuals do not learn to act as a team, a hopefully straightforward concept. Yet many underestimate the investment required for individuals to operate with a sense of real teamwork, as well as how important this cohesion is to team performance.

A team space dramatically accelerates people's ability to, well, team. Historically this presence was created through dedicated physical space. In our current climate of mandated working from home, this sense of presence needs to be created virtually, which is trickier but still possible to pull off with the right tools and focus.

As mentioned previously, it was not apparent that Scotiabank's payments group had a history of collaborating well across the many other groups and departments in the bank. This limited collaboration was likely amplified by a working environment where people were functionally separated and physically distant from each other. Here's Sean talking about how traces set up a team space:

> *We felt it was important for us to find a space where this team could truly be together. Trying to find a team space across the sprawling footprint of Scotiabank was surprisingly difficult. In the spirit of true self-organization, our product owner, Dan Hawkins, went rogue and annexed his boss's spacious downtown corner office. We immediately put all of our team's information radiators [big visuals that contain key team*

information in a simple, informal way], in this case a story map and Kanban, up on the windows of the office. A facilities manager arrived shortly thereafter, demanding we take the artifacts down. Agile artifacts on the windows looked ugly from the outside of the building, and this was not proper use of an executive office. Dan, to his credit, held firm. To this day, we are still unsure how Dan managed to accomplish this.

OPERATING NORMS

Effective teams will often establish a set of operating norms to help align and interact with each other as a team. Agile provides a specific set of operating norms for planning, review, improvements, and daily coordination. Teams frequently set these operating norms up according to set cadences. For instance, a short iteration is started with a planning session and ended with a review and retrospective. Short stand-up meetings occur daily. Teams often learn to decouple these sessions into separate cadences and to vary the type and frequency of sessions according to their needs. Teams that frequently establish, retire, and invent new norms inevitably become better at developing their shared identity.

At Scotiabank, Sean and I instituted biweekly planning meetings and retrospective sessions, as well as daily stand-ups. The visibility of these meetings paid unexpected dividends, generating interest and enthusiasm throughout the group.

STABILITY AND DEDICATION

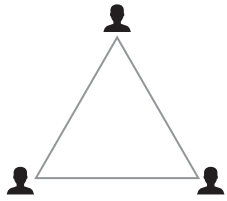
In the traditional working world, team members often do not have a chance to gel into anything cohesive, simply because team membership is thought of as a temporary and/or part-time commitment. The team roster is constantly changing. No one is truly a member of any team. Teams cannot establish meaningful norms, and team members never really learn how to work with each other.

The team never actually does any *teaming*. (*Teaming* is a verb, the title of the next chapter.) An Agile team is made up of *full-time members*, and that membership is made up of a *stable roster*.

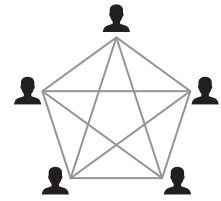
Stability and dedication are both necessary to maintain a *stable social boundary*. That's what we found on the Scotiabank project too. Very quickly, the culture changed and the team became much more efficient.

SMALL AND SOCIALLY DENSE

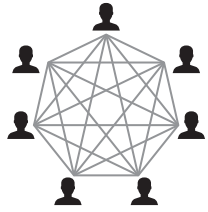
We also want to keep teams small, somewhere between five and twelve people. Metcalfe's Law, which states that the effectiveness of a network is proportional to the square of the number of connected devices, can be interpreted to show how many communication pathways exist based on the size of a social grouping. So, for a team increasing from three to four members, the number of efficient communication lines suddenly doubles from three to six. In other words, smaller teams are more effective teams.



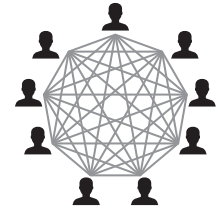
3 Members = 3 Connections



5 Members = 10 Connections



7 Members = 21 Connections



9 Members = 36 Connections

At Scotiabank, we found that keeping teams small proved challenging. As Sean noted, the traces team needed to redefine itself

repeatedly over time:

> *As our team hunted for, and successfully found, more work, an interesting thing happened. New demand kept uncovering the need for new skill sets. Expertise based on different systems, different customer groups, internal users, product experts, and channel owners were all required for various features that the team was attempting to deliver. We needed to add more people to the team. Before long, the size of the team became unwieldy, sessions became unbearably long, and the dedicated space became overcrowded. We moved the team to a much larger room, but that too soon became overcrowded as more and more people joined the team. It felt like the entire organization was trying to become part of our one team!*

We will talk more about how this team evolved in part 2. For now, it is important to note that Agile concepts make it easier for people in teams to adapt to change. And change is the one thing we can be certain of in today's world.

PATTERN

The Team Worker

This book makes plentiful use of both patterns and antipatterns. Patterns are a very useful way to discuss various solutions to problems we encounter when faced with complexity. When our world is uncertain, we can't turn to fixed, out-of-the-box solutions. Instead, we turn to a library of patterns, each pattern describing a different aspect of a problem, along with a corresponding solution. Care is taken to talk about how the patterns relate to each other.

A key aspect of this book is *team collaboration patterns*. Collaboration patterns address different ways a person can engage with teams to create value. They are especially helpful for people who provide support functions and want to understand their place in an increasingly Agile world.

Just like children can use a finite set of LEGO blocks to build a much wider, almost infinite variety of larger creations, collaboration patterns can be connected together as required to define larger organizing solutions. Patterns include not only stable, cross-functional team membership, the most obvious pattern we see in the Agile world, but also lots of other ways of forming together to collaborate.

With all this chapter's talk of teams, it is time to introduce our first and most obvious team collaboration pattern, the *team worker*.

The team worker spends the majority of their working time in a team. They maximize the amount of time they spend teaming with others, delivering value through *teamwork*, not lone work. A team worker is part of a cross-functional, stable, market-facing team.

A mix of skills and capabilities is required to successfully deliver that value. The need for this particular set of skills is significant, and it is stable over time. A high degree of collaboration is required between the team worker and the rest of the team. The work is complex and emergent and requires continuous learning.

When you choose to be a team worker, you are committing to the fact that you will be spending the majority of your working day with a team in order to achieve team goals using team metrics. Wherever possible, we want most of our people in our organization to choose to be team workers in market-facing teams, taking as much responsibility for end-to-end value creation as they can. This won't always be possible depending on context, but it is a goal to strive for.

The more key team attributes mentioned in this chapter you adopt, the better; it is the *combination* of these attributes that creates a complementary set of corrective mechanisms that enable self-organization.

These attributes can be applied in a way that is agnostic of any one Agile framework, method, or methodology. There are organizations whose teams exhibit most or all of these attributes without regard to a single official Agile practice, or without even really being aware of Agile at all.

That being said, there is a wealth of Agile material out there that provides some very compelling approaches; teams could do a lot worse than reading up and experimenting with them!

ANTI-PATTERN

The Department Worker

The opposite of someone self-organizing in a team is someone who is doing work at the behest of their functional manager. Being part of a functional organization is and will for quite some time be a reality for many of us. This in and of itself is not what we mean by a *department worker*. A department worker is tasked to help with inward-facing organizational work. They do the work because their boss wants it done. There is no other team asking for the work, at least not directly. As a result, it can be very hard to trace the work back to market value.

At Scotiabank, Sean and I struggled with trying to grow a team in an organization of department workers. As he recalled:

> *Several months into the payments group's adoption of Agile, the inevitable happened. A combination of*

individuals working hurriedly and in isolation, a lack of alignment across the team, and continued pressure to meet deadlines all culminated in work that had poor quality. They were passing very few tests. An obvious and significant bottleneck had formed in testing. One of the initial issues identified was that testing was often a separate and siloed activity.

In order to make real progress on the issue, Greg Lahn, the executive in charge of the payments group, took the ownership of testing their solutions away from the centralized quality assurance [QA] group. He then immediately gave that ownership directly to the teams, empowering them to evolve to a combined business/quality analyst role.

When we use functional department managers to deliver value, we often create internal functional goals that interfere with market goals. When teams are up and running, we can see conflicting direction given to them. When teams do not exist, we see single contributors who coordinate their work with other single contributors from other functional departments. Collaboration needs to cross departments; it happens outside of your organizing structure and management system.

As organizations further move toward a team-based model, they must strive to reduce the amount of work that flows into departments through department managers, and to increase the amount of work that flows from the market into the dedicated members of a team.

ORGANIZING CONSTRAINT

Only Teams Deliver Value to the Market

Throughout the book, I'll share a set of *organizing constraints* we can use when thinking about our organizational design. These constraints are pretty specific on *what* we need to do if we want to increase organizational agility but are deliberately ambiguous on the *how*. That is because there is no correct way to introduce these constraints. How you go about introducing these constraints will depend on your context.

Our first organizing constraint is that we want to limit market access to teams serving that market. We want teams, and only teams, to be market facing. This implies that the majority of people in our organization spend the majority of their time working in teams.

This orientation has a profound impact on our organizational structure, with an emphasis on teams that can achieve market outcomes. Teams are connected directly to the market they serve, and are empowered to interact with the market actors.

So, stated fully:

Deliver market value through small, stable, cross-functional, self-organizing teams that take advantage of radical transparency, full-time membership, focus, and frequent feedback.

ORGANIZING CONSTRAINT

Functional Departments Only Grow Capability and Capacity

As stated previously, the functional department approach results in the need for a lot of coordination; this makes us reliant on bureaucracy and management to get things done.

A much better model is to rely on functional departments and functional management to define and grow functional capabilities. Hire people who can possess a functional expertise to provide training and learning and who can grow their careers to excel in a particular set of functional capabilities.

Delivery of value, on the other hand, is taken away from functional leads and becomes the responsibility of cross-functional teams. These teams can have leaders, but these leaders should not be confused with functional management.

So, stated fully:

Use functional departments to grow capability in a functional skill set; avoid managing value creation through functional managers.

IN SUMMARY

- Teams are the organizing construct that bring the motivation, collaboration, and feedback required to deliver value in the face of constant change.
- Agile can provide many of the foundations to set up and operationalize a self-organizing team.
- Starting with radical transparency will increase shared understanding across all participants.
- Grow team cross-functionality and increase autonomy over time.
- Adapt to market feedback frequently, iterating toward success and reducing the need for command and control.

- An Agile team is made up of a small and stable roster, with an established team space, so that they can develop the social bonds required to be effective.
- Establish operating norms for your team, such as planning, stand-ups, and reviews, and hold them at a steady cadence.
- Teams deliver value, while functional departments grow expertise. Leading value creation and leading the development of expertise are two very different leadership activities, and the two should not be confused with one other.

3

Teaming Is a Verb

So, now we have gone through why Agile teams are a robust organizing construct for delivering value in the face of constant change. With all this talk of how great teams are, you might think that setting up Agile teams will solve all problems in all situations.

In truth, most of you will find it incredibly difficult to get all aspects of teams working in an Agile way in an enterprise setting. It's a good thing that there are other structures we can lean on to meet some of these challenges. We will go over these challenges and the organizing structures that can help in the next two sections of the book. For now, it's important to note that all of the organizing structures and techniques that we'll cover have a common purpose: to enable a group of people to engage in the act of *teaming*.

Before getting to what I mean by "the act of teaming," let's take a second to think about the classic definition of a team. Many people would say that a team is "a bounded and stable set of individuals interdependent for a common purpose." Heidi Helfand, in her excellent book *Dynamic Reteaming*, challenges the "stable" part of the definition. She asks readers: If a team is more highly changeable, in the case of a hypergrowth start-up, is it still a team? Helfand posits that dynamic teams are still teams. When two people get together to collaborate on a common outcome, they become a team.

It is this notion of teams having an element of dynamism that leads us to the idea that teams are not merely some boundary that we use to group people together. Teams are the set of active, living interactions between individuals. We can't have a team if people aren't teaming. Teaming is a verb!

So *teaming* is the act of people coming together to deliver value. It is the act of people organizing so they can engage in hypercollaboration. Establishing trust, understanding each other's strengths and weaknesses, positive conflict, working toward shared outcomes, and holding each other to account while having each other's backs are all part of teaming. Teaming allows us to get organized in a way that respects our humanity by placing people front and center. When we engage in teaming, we deliberately place our trust in others to accomplish a shared outcome. We collaborate to generate value.

Agile teams are really a means to introducing the behavior of teaming, and only one means, albeit a foundational one. In this chapter, I'll discuss how the Agile-powered team enables the act of teaming. Teaming is the goal, not team.

TEAMING IS HOW HUMANS GENERATE VALUE

Traditional approaches to getting organized rely on creating fixed structures in response to a plan, a road map, or a strategy. We place people into functional hierarchies or project teams. Managers assign work to their subordinates, which often gets pushed down to the exact person and down to the exact task. That is an approach meant for machines, not people.

Of course, the way we actually get work done in the knowledge economy does not work this way. At least, the valuable work does not work this way. The reality is that valuable work gets done when people closest to the work learn to collaborate, when they engage in teaming in a way that ignores and often directly contradicts the official management structure. People in large organizations get high-quality work done when they learn which hoops to jump through when they need to and who will help them avoid the various traps lying around their organization.

The hierarchy supplemented by underground teaming is not a great solution. It subjects people to a completely ad-hoc and unpredictable workplace. Everyone is at the whim of everyone else. New rules appear out of nowhere. People leave massive holes when they exit the organization.

When we organize ourselves based on the idea of teams, we provide a safe container for people to form into groups and collaborate on the delivery of value. There are other containers, which we will discuss later, but teaching people to collaborate well inside a team is a critical first step. Again, establishing this collaboration between a small number of people is the art of *teaming*, to team up with one or a few more individuals for a short period of time.

In fact, a key tenet of early forms of Agile is that an individual task should always be worked on by a *pair* of people, not an individual. We want to maximize the number of tasks that are accomplished through the collaboration of a group of people. We want to minimize the amount of time an individual works on something by themselves. When we use teams as a container to encourage teaming, we get the best of both our formal structure (the team we are in) and our dynamic structure (the teaming we do). It's an approach that places our humanity front and center.

And now a tongue twister . . .

Team members team with other team members to effectively team!

This is so because of the following:

- We can adjust how we are organized based on unforeseen changes.
- We grow our skill sets and increase learning and exposure across different roles and skills across the team.
- We reduce the impact of anyone leaving the team.
- Our quality goes up as a diverse set of perspectives are simultaneously applied to the work.
- We reduce the impact of bottlenecks as we pinch-hit as required to complete unfinished work.

The perceived pitfall of constantly teaming on work is the illusion that it is inefficient, that we will get less done, and that we will waste time. The reality is any complex endeavor requires an immense amount of learning and knowledge transfer. Furthermore, siloing and handoffs cause an immense amount of waste. When teams are working on dramatically different things at the same time, we tend to see a huge increase in defects and rework. Just-in-time feedback is way cheaper than after-the-fact feedback. Bottom line, when multiple people team on the same work, it almost always ends up being cheaper than the alternative. It won't always be: solo work is suitable for trivial work, monotonous work, and commoditized work, the kind of work that should be automated and abstracted away at some point, anyhow.

Agile, regardless of which method, helps us set up a space where teaming can occur. In the rest of this chapter, we will look at how Agile, both in general as well as specific Agile methods, can bring about teaming within your organization.

ORGANIZING CONSTRAINT

Use Agile Team Spaces for Hypercollaboration

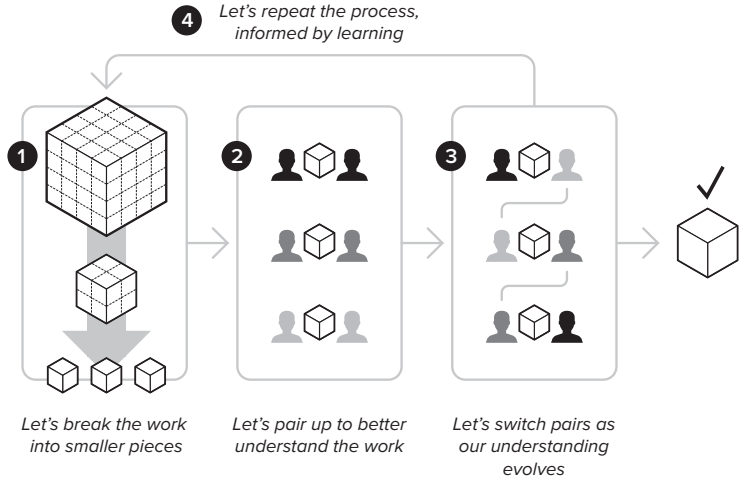
The Agile team space is a container where co-creation and shared work can take place. Within this space, it is typical to see people reconfigure dynamically into twos, threes, and fours, as necessary to accomplish the team's overall goals.

People in a team coordinate with stakeholders to shape the work, decomposing it into smaller pieces, defining acceptance criteria, fleshing out and refining the details, and so forth. They collaborate to prioritize the next set of work to accomplish, thus creating a backlog of tasks. The team will often divide into smaller groups to "pair off" against individual items taken from the backlog.

People in a team space also meet frequently, perhaps daily or even several times a day, to agree on how to do the work and

on who does what. Pairs are often switched as needed based on availability, capability, knowledge sharing, or pinch-hitting to help each other out.

At the end of the cycle, people collectively review what was accomplished, often with a set of active stakeholders. They will take time to reflect on how they could have planned and delivered differently in an attempt to improve the way they work. Groups may form to implement discrete improvements.



Agile is a constant act of organizing

So, stated fully:

Use teams and other safe, trusted social spaces to enable the majority of work to be done in co-creative or hypercollaborative ways, such as in pairs, mobs, swarms, or cells. This is known as teaming.

TEAMING WITH KANBAN

Kanban, originally from lean manufacturing, has been adapted to be a formal method that is suitable for knowledge work, thanks in large part to the efforts of David J. Anderson (no relation).

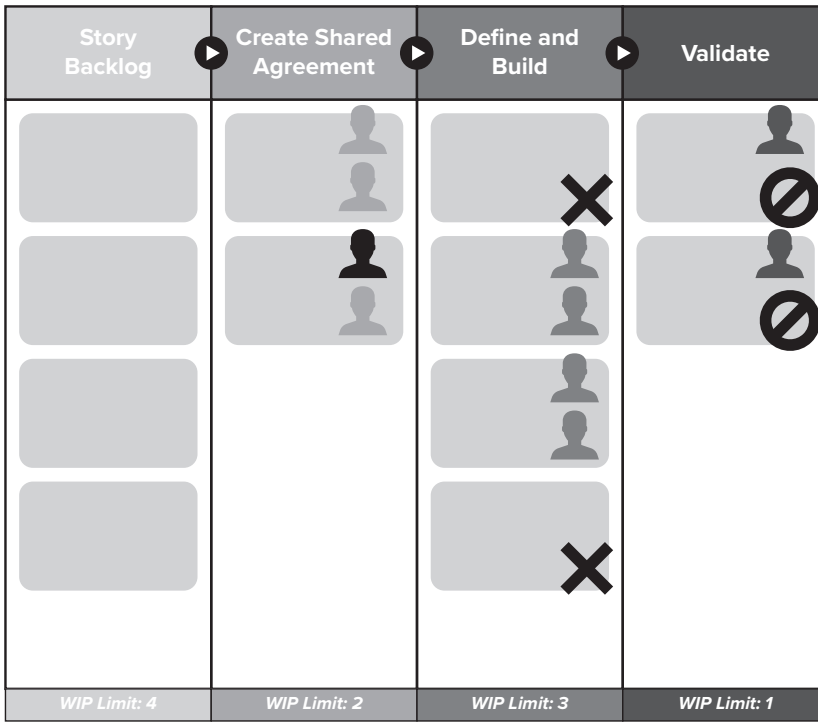
Kanban, as taught to me by David, has a special place in my

heart as it is a particularly good method to enable the art of teaming. Kanban has a special place in my heart, as I have witnessed some very compelling success stories in the context of helping people better collaborate in traditional enterprise settings. Because of my experience with the method, I'll continue to revisit how Kanban can be used to enable people to team in a variety of different ways, both within and across teams. I'll illustrate some teaming patterns later in this chapter through a Kanban lens. Of course, Kanban isn't a prerequisite for teaming to take place, but if you are open to trying it out, Kanban can be a solid enabler for teaming. Before getting into how Kanban helps with teaming, I'll do my best at providing a brief description of the method itself.

Of the many methods and ideas out there in the Agile-verse, Kanban seems to inhabit a unique space. Unlike many of the other approaches, Kanban is really a tool kit for people to design and operate their own system of work. It is a set of concepts you place on top of whatever method or methodology you want to use, be that another popular Agile framework, like scrum or SAFe, or a "roll-your-own" methodology. Kanban exists on top of whatever choice you make. The method introduces a set of principled behaviors that help you increase agility, transparency, and resiliency in terms of your team's ability to process change. Kanban asks you to enhance the way you work, whatever that way of work is, through the following fundamentals:

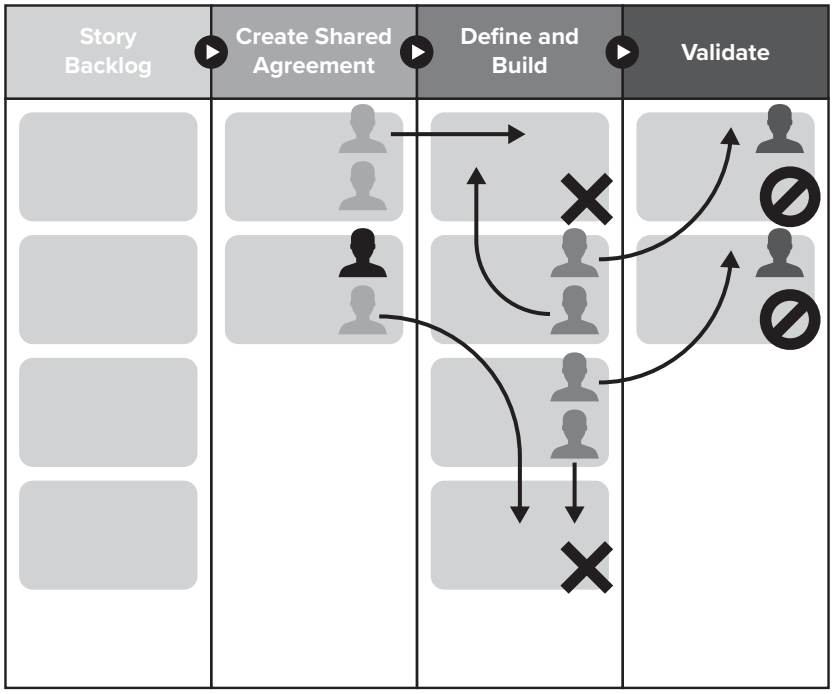
- Visually define and manage your workflow and all of your work that is in that workflow.
- Collaboratively refine the agreements that define how work flows from one state of completion to the next.
- Limit the amount of unfinished work in the system at any one time.
- Establish cadences that fit the needs of your team, your customers, and other teams you work with.
- Improve the performance of flow by constantly identifying bottlenecks, blockers, defects, and other impediments, and introduce improvement experiments that are verified using quantifiable metrics such as lead time and throughput.

A team using Kanban starts by agreeing on a *reasonable* workflow to sequence their work. The team then collaborates to write out a set of agreements (*policies*, in Kanban speak) to create a shared understanding of how the work can flow. These agreements document the work to be done at each stage in the workflow. Note that it is expected that this flow and the accompanying agreements are expected to change and to change constantly! When new work comes in that does not fit the existing flow, then people are expected to change their flow. The idea of Kanban is that we are not setting rules in stone. Rather, we are writing them in pencil, a pencil with a big eraser. Kanban gives the team a lightweight way to continually update how they can collaborate to flow their work to completion.

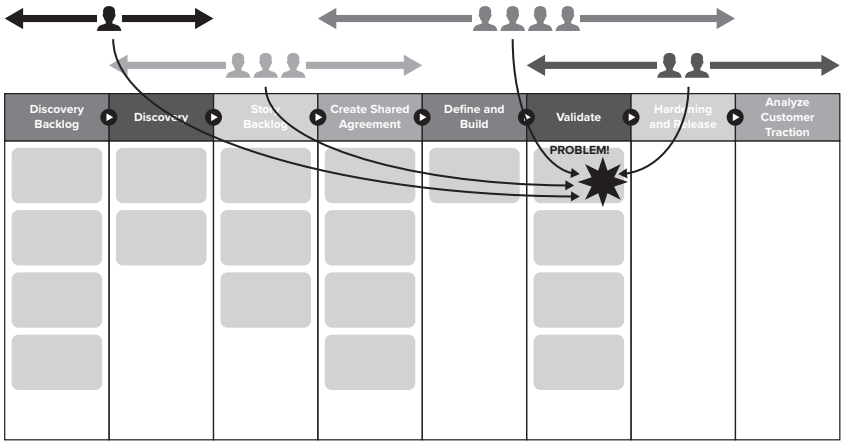


All of these activities are acts of teaming. When a group of people collaborates to set up this form of visualization, they make it easier for themselves to collaborate around the work. They get to agree on how to facilitate the flow of value. When a bottleneck forms in one area of the workflow, the team is provided a signal to stop what they are doing and offer assistance. For instance, a bottleneck in testing may indicate a problem in defining acceptance criteria attached to the team's

stories. This signals the developer and analyst, who originally collaborated on the story, to now turn their attention to working with a tester to resolve the problem being illustrated by the testing bottleneck.



This trivial example of Kanban in action is really an example of teaming, of individuals stepping away from their siloed view of the work and collaborating together to accomplish the delivery of value. Kanban works especially well when people are able to move past any official job or role they have to come together and solve a problem that is impacting the flow of value.



When team members, their stakeholders, and other participants collaborate by attending this style of visual management, they often end up *teaming* outside of their official organizational structure, whether that structure be team or functionally oriented. We end up seeing teaming across official organizational boundaries and toward common outcomes. We see tighter engagement across a wider horizon of activities.

We see team members expand the teaming concept to other knowledge workers who may typically work upstream or downstream of them as part of the flow of value creation. Kanban can scale the teaming *horizontally* across a value stream.

Tyler Motherwell, another fellow coach at Agile By Design, provides an example:

> *A Canadian extension of a large US-based FSI, focused primarily on the credit card business, was in the midst of a company-wide Agile transformation. [I'll call the company Canadian Credit Card, or CCC for short.] Product, operations, tech, you name it, everything was moving over to this new pod-oriented model. As part of this transition, I was asked to coach the line of business responsible for partnerships with retailers and help the pods within utilize some of the tools in the Agile toolbox.*

One pod, focused on marketing for new products, had some very real challenges. Their goal was to acquire customers through physical and digital marketing campaigns. The team had felt for a while that getting campaigns out the door in a timely fashion required an inordinate amount of heroics and was looking for a way to streamline. Marketing managers in the product/marketing pod relied on creative professionals in a creative pod to develop the campaign assets, as well as production professionals in an execution studio pod to finalize the assets and deploy them to various online and offline channels. All three pods made some use of Agile techniques, but work did not flow well across the three teams, communication was spotty, and rush jobs were the norm.

The team had traditionally used calendar-based plans to synchronize across teams. However, plans were constantly having to be adjusted and readjusted, making scheduling work across teams feel like a futile endeavor. The marketing team felt like their expectations were never being met; the creative and execution studio teams felt like requests appeared out of the blue. Despite all teams feeling like their own pods were working well, all involved teams were actively looking for a way to improve.

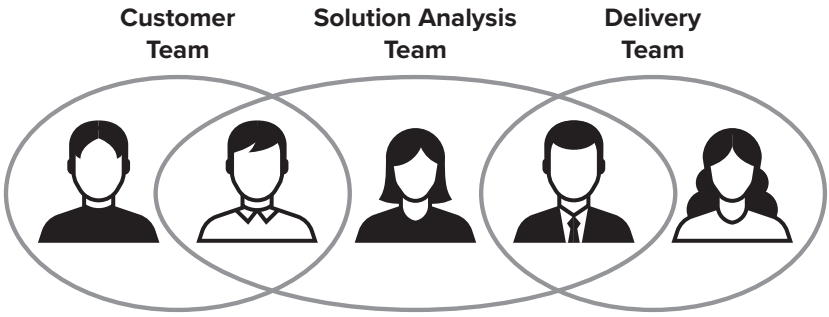
The first thing we did was work with members from all three pods to build a high-level value stream map. We got agreement on all the steps, states, and outputs required to bring a market campaign from a glimmer in someone's eye to live in market. We highlighted where members from all three pods currently needed to come together, for instance during planning and review sessions, and where they wanted to work in a more loosely coupled manner. I helped the teams agree on a simple model to estimate the complexity of campaigns and the individual creative assets that made up the campaigns.

We took this agreed-upon process and created a Kanban system dedicated to managing the flow of campaigns across the three teams. At first, we planned to run this system of work in parallel with the schedule-based one the teams were currently using in order to minimize change resistance and to not be perceived as "taking something away." After a few sessions of drawing analogues between what was being discussed with the calendar and how it was represented in the Kanban system, the team decided that they could get everything they needed with the Kanban and sunset the calendar system. Personally, I was blown away with what a huge difference a bit of flow-oriented visual management made to help people collaborate and get organized outside of their home-base pods.

THE TEAMING VALUE NETWORK

Some perceive Kanban as a tool to define work that is performed by isolated roles and individuals. I think of Kanban as a good way to identify a person's *starting position* within the flow of the work. More importantly, we can use Kanban to agree on how and when people can move into new positions, upstream or downstream, to collaborate with others. Value networks can be thought of not as handoffs across teams, but as concentric circles of teams where people overlap to ensure work flows smoothly.[13]

13. Jurgen Appelo, *Management 3.0*, Indiana: Pearson Education, Inc., 2011.



A simple view of the concentric circles in our value network

We can think of connecting our people across teams as linking; *linkers* are people who collaborate across team boundaries. And linking is something that is pretty easy to visually manage using a Kanban system. This is an idea I first came across when reading Jurgen Appelo's now-classic book on agile management, *Management 3.0*.

ANTI-PATTERN

The Cross-Functional Team of Siloed Team Members

The ways that people can form together and team in different configurations is likely endless, but let's share some popular patterns, starting with an anti-pattern, the *cross-functional team of siloed team members*.

It is the unfortunate reality of many teams proudly proclaiming how agile they are that they appear to be cross-functional from the outside but that, when we take further notice and look at the makeup of the team, we see very specialized knowledge workers with few overlapping roles or skills. We see a cross-functional team of siloed team members.

Many teams start off like this, and this is perfectly natural given that most Agile teams are born out of industrial-managed organizations. The good news is that the individuals in this type of team have all kinds of opportunities to expand

their horizons and increase the amount of teaming they do.

Signs that you have a team of siloed team members include:

- It seems like team members only communicate across functional siloes (the analysts, the developers, the testers, etc.) during official team cadences like stand-ups, retros, planning, and so on. It feels like sessions take way longer than they should and that team members are constantly being surprised by new information. Likely, they are hearing about things for the first time.
- Everyone has a different way to refer to things. Taken to the extreme, I have seen different functions have their own list of stories that got translated multiple times across the team. UX stories! BA stories! Dev stories! Ughhhh . . .
- We have a lot of handoffs, often occurring in big batches, accompanied by a slew of blockers, defects, and other team impediments. (For the most part, this wouldn't occur if the members of the team worked with each other.)
- Finger-pointing. *It's your fault! No, yours!* Something I (un) pleasantly refer to as blamestorming. Always an amusing experience.

THE PAIR

I first came across the concept of pairing about two decades ago when Extreme Programming became popular.[14] I originally hated the idea.

I loved the idea of test-driven development. I loved continuous integration. I loved stories, backlogs, and stand-ups. But I specifically loathed the idea of pair programming, as too did most of my peers. It felt unnatural and weird. It meant I had to take my quiet, isolated coding bliss and make it a social experience. Groan.

14. "Pair Programming," Extreme Programming, http://www.extremeprogramming.org/rules/pair.html.

It took a couple of years, but I came around. A friend and trusted colleague convinced me to try it, so I took the plunge.

We took our team of six and formed into pairs. Each of us grabbed our piece of the backlog and went for it. One of the paired members would have his hands on the keyboard. He was the *driver*. The other member played a navigator role. The navigator would sketch out ideas, take note of future refactorings we needed to do, help the driver stay focused when their eyes glazed over, and overall try to help sustain a nice, continuous flow of work. The driver, in turn of course, typed code. We followed a test-driven approach and would pass the keyboard to each other every fifteen minutes or after each step in the process, whichever was sooner.

It took some getting used to. At one point we asked the pairs to engage in a highly scientific experiment. I asked them to ring a bell every time they felt their pair was able to save time or improve quality because of the fact that they had two brains working on one problem. The bell was rung many times. Other people in the office got very angry with us, but it was worth it. We had our irrefutable proof: pair programming was indeed worth doing.

In truth, pairing is perhaps one of the most foundational and core concepts to try if you are serious about increasing agility. Pairing is an excellent way to level people up and improve their skills. Stepping a bit away from the keyboard-swapping notion of pairing and thinking about pairing as the act of two people working closely on one thing, we can see that pairing provides a lot of value when the members of the pair belong to different functional disciplines. Example pairings include:

- Marketing and production
- Operations and everyone else
- Call center people and everyone else
- UX and developers
- Developers and analysts

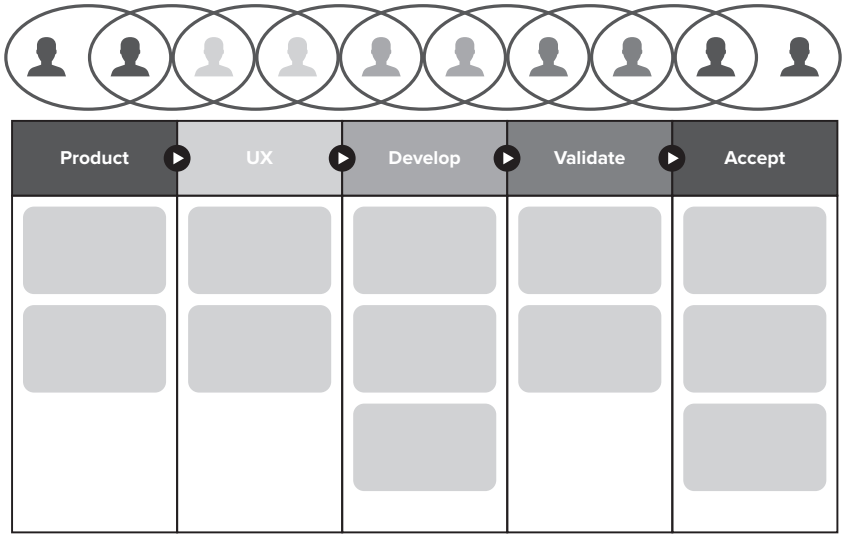
In fact, every handoff you see across multiple functions and teams can be better handled through pairing. Using this approach, work is never handed off between people; rather, someone in the pair switches

out. For instance, in a stereotypical web development flow, we might see product owner ⟷ UX, UX ⟷ developer, developer ⟷ QA, QA ⟷ product owner. Team members can expand their use of pairing, from the more traditional pair programming practice to pairing across disciplines and functions, avoiding handoffs as a result.

Mina Tawadrous, a senior product manager at Pivotal, described the importance of teaching their clients the practice of effective pairing:

> *At Pivotal, our goal is not only to help our clients modernize their business applications to take advantage of the latest software platforms, but also to help them modernize how their people work together to build those applications. Whenever we help a client deliver a new solution, we do so by maximizing the pairing of our people with the client's people. We pair our developers with theirs, our team leads with theirs, our product managers, etc. We try to keep our mix to an even 1:1 ratio and incrementally grow the team over time, say 1 to 2 new developers per team, per month. When pairing on code, developers were encouraged to frequently switch pair assignments, often as soon as a pair completed an individual acceptance criteria on a story.*
>
> *We use this approach because we want client developers to get a really solid foundation in modern engineering concepts like test-driven development, software craftmanship, refactoring, and of course pair programming itself. There have been times when we have attempted to work a smaller ratio of pivots, or brought client developers on board faster, at the client's behest. While this may have allowed us to get an application out the door by a certain date, it prevented them from growing the foundational engineering capabilities to sustain fast delivery over the long term. Pairing has to be about the human element; it can't be scaled prematurely or rushed. When you approach pairing and growing the team thoughtfully, you end up with teams*

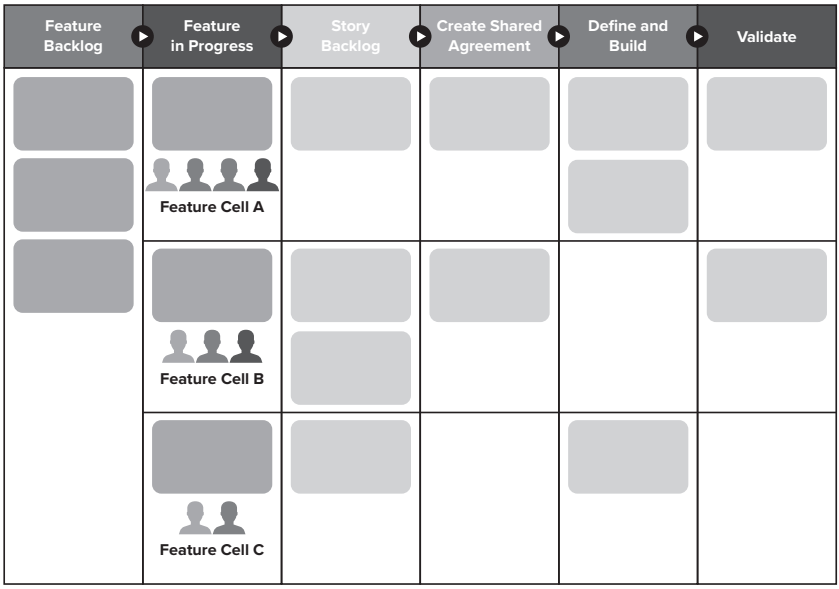
where everyone knows the entire domain, they understand the codebase, and they operate with an equivalent skill set.



THE FEATURE CELL

The idea of a cell is that it often makes sense to explore and even deliver a piece of work with a tight grouping of people. A group that is smaller than the team but larger than a single individual or single pair on the team. Often larger teams—say, more than five or six people—struggle to effectively collaborate across the entire body of work that the team is delivering. Splitting the team may be impractical or undesirable for a variety of factors.

One approach that teams can take is to ask people in the team to form into a cell for a short period of time. The cell agrees to take on a collection of work that is tightly connected. For example, a feature containing several stories. Once the work is completed, the cell can either take on the next piece of work or dissolve so that the people can be available to take part in a new and different feature cell.



In a software setting, it is helpful that the cell follows a *three amigos approach*. The term, first coined by George Dinwiddie,[15] expresses the idea that people representing the three perspectives of business, development, and testing should collaborate closely together to deliver on a small increment of value. This does not mean a feature cell should contain three people strictly, just that we need a feature cell to include enough people to cover these concerns.

Stephanie Dimovski and Ruth Nielsen, both product owners working for the IT organization of a large Canadian pharmacy, were having challenges effectively collaborating both within and across their teams. The teams were tasked with modernizing their point-of-sales systems being used by in-store pharmacists, a job which required a range of domain and systems experience.

Stephanie described the complexity being faced as she worked with the rest of the product owners:

Our team was expected to work cross-functionally,

15. George Dinwiddie, "The Three Amigos: All for One and One for All," StickyMinds, November 29, 2011, https://www.stickyminds.com/better-software-magazine/three-amigos.

delivering an end-to-end slice of functionality that demonstrated a complete journey. Things seemed really complicated. We had a good deal of churn. There was a lot of complexity that our largish teams (approx. ten each) had to handle; the technology and architecture of the various front- and back-end pieces of the solution were so different. It was really challenging to make progress on an end-to-end flow. Like the team was processing more concepts than they could handle. But we didn't want to shrink the team sizes as we felt that would create a lot of dependencies. But the team-level ceremonies took a long time, and we could never get through everything we wanted to as a team.

Ruth, Stephanie, and the rest of the leads across the teams decided to try out feature cells. Ruth explained the approach they took:

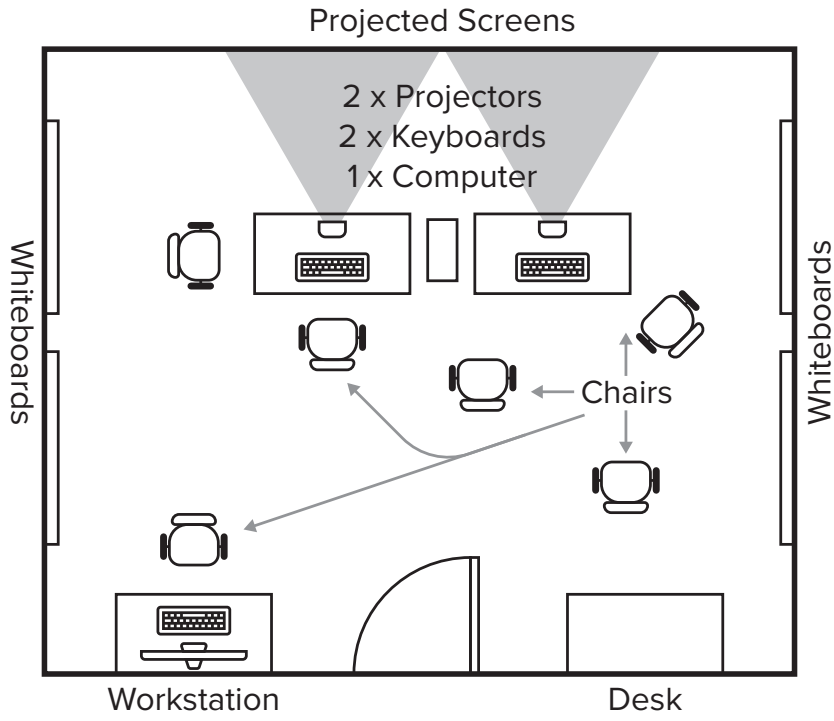
Feature cells would form as part of feature shaping, and then potentially disband once a feature or small group of features was done. Each cell had a combination of front-end and back-end specialists, an Agile engineering expert, and a product analyst who provided support by way of facilitating the gathering and testing of functional acceptance criteria. The approach took a couple of weeks to get going, but the results were remarkable. Our throughput increased significantly. More importantly, we empowered the team to choose the cells they were a part of. We balanced ownership with the ability to work with a tight-knit team and focus. The cells evolved as they needed to, which allowed us to avoid dependencies that spanned cells or teams.

THE MOB

A mob is like a pair but taken to the next level. In a software development setting, mob programming is a practice where only one person

in the team is allowed to program at a time.

Woody Zuill pioneered the first instance of mob programming during his time at Hunter Industries.[16] The keyboard is rotated frequently across each member of the team, sometimes at fifteen-minute intervals, sometimes less. The rest of the team supports the flow. One person may play the navigator role we mentioned in pair programming. Others may be exploring new technology, looking up information in databases, modeling stuff on whiteboards, you name it. Anything but hands-on programming.



The team at Hunter Industries set up the entire room to support the mob programming approach. Computers, projectors, and desks were all arranged to create focus on the one piece of work being completed by the (only) active pair. All interactions were done as a team. Emails were sent as a team, phone calls were made on the team

16. Woody Zuill, "Mob Programming: A Whole Team Approach," Agile Alliance, July 2014, https://www.agilealliance.org/resources/experience-reports/mob-programming-agile2014.

speakerphone, and so on.

Some of the benefits? Less thrashing, higher focus, freedom from "overhead practices" like estimating and scheduling, better communication, lower technical debt (the amount of technical work that requires cleanup in the future), less team politics, and fewer meetings. Incredibly, the team at Hunter and others who have tried mob programming have actually reported higher productivity. That can be a hard number to measure, but in many cases teams who mob feel they are getting more, not less, done.

According to Zuill, the ideal size of a mob is not unchangeable: anywhere from six to twelve people. The best heuristic? If team members feel like they are contributing, learning, and growing, then the mob is likely the right size.

THE SWARM

A swarm is any group of people, more than a pair, that assembles dynamically to accomplish an outcome. This group swarms often by coming together to solve a problem that is preventing work from being completed. They swarm to collaborate on improvement experiments or to resolve a persistent customer complaint. Swarms come together to bring the diverse perspectives across functions, roles, and even teams.

I first encountered the swarm concept around 2007, as I became deeply immersed in the Kanban community. I am unsure who coined the term, but I remember falling in love with the concept as soon as I heard about it.

We know that Kanban has inherent capabilities in pointing out impediments, bottlenecks, and other forms of dysfunction. Given this, the idea of swarming is especially prevalent. A common scenario is for a team—or representatives across several teams—to attend a common Kanban system. When a bottleneck forms in the flow of work, often made apparent by an excess of work in progress (WIP) in a particular column in the Kanban, then people agree to drop what they are doing and swarm around the bottleneck until flow is restored, as indicated by a drop in WIP. Swarming also takes place when other particularly

thorny or persistent problems are encountered. Some examples include reoccurring blockers, high defect rates, and overall long lead times.

IN SUMMARY

- Teaming is a humanistic approach to organizing people that asks members to closely collaborate in groups to deliver value.
- The Kanban method makes it easier for knowledge workers both in and across teams to understand when they need to closely collaborate both within and across more static organizing structures.
- Pairing is when two people work on a single piece of work, typified by only one person on the keyboard at a time.
- A feature cell is a subgroup of the team that comes together to deliver a coarser piece of work on the team's backlog.
- Mobbing takes pairing to the next level, an approach where the entire team closely collaborates on a single piece of work, again with only one person on the keyboard at any one time.
- A swarm comes together across roles, teams, and functions to solve a sticky problem that transcends official organizational structure. Kanban is designed to facilitate the swarming required to eliminate bottlenecks and maximize the flow of value.

4

Roles and Jobs for Team Members

The team and the act of teaming are concepts that allow people to organize in a way that delivers results while still being fundamentally human. Teaming is a way to create space where people not only get things done, but also learn from one another—where people can grow beyond the confines of a narrowly defined set of activities.

Traditional organizations contain a lot of machinery that gets in the way of effective teaming. One big and pointless obstacle is the definition of the jobs we get hired for and the roles we take on. The standard, tried-and-true approach for most organizations is to give everyone a highly defined job description. The idea is that if we define what everyone is signing up for up front, then it becomes much easier to work together as we scale up. Everyone has clear and specific instructions, and everyone knows what to expect of everyone else.

Sounds great, doesn't it? Not if we want living, thinking beings to thrive through teamwork.

JOB AUTOMATONS

In an environment of constant change, such new and novel work is always happening. But your people are not thinking about how to

organize around this new and novel work. People are focused on doing their job, according to its exact definition. Or else they are focused on doing what you tell them to. No less, but no more either. In other words, people are behaving as *job automatons*!

Employment in any organization comes with a job. A job that has a very specific definition. This job definition is used to describe:

- The role you play (typically one role)
- The accountability you have
- The (very specific) boundaries that demarcate your responsibility
- How you progress to the next level of your career
- The level and number of people that report to you

This documentation will be largely static and changed infrequently. Let's take an example:

Business System Analyst I

- *A "business system analyst I" works under general supervision to identify gaps between business objectives and systems scope.*
- *The analyst is responsible for working with both business and technology to make sure requirements are clear, accurate, and well understood.*
- *The analyst will facilitate various requirements collection workshops and is accountable for eliciting business needs for all stakeholders.*
- *The analyst documents these requirements and reviews them with developers and testers. The analyst reports to a "business system analyst II."*

This type of job description puts the focus on individual effort rather than on the interactions across individuals. It starts and ends with an individual's functional duties. Remember, when our people have an unduly narrow specialization, we end up with lots of handoffs

across people. Work tends to be performed in a highly serialized manner, with individuals working in mini (or maxi) siloes. These handoffs create waste, cause rework, and impede learning. We face constant bottlenecks.

A job automaton does their work according to the letter of their job description. The problem is that most job descriptions are out of date the moment they are written, so job automatons consistently fail to deliver on outcomes that require adapting to change. Things get stuck in the cracks between all those individual instructions we have defined. When this happens, things get missed. Organizational failure happens and, without fail, all the people involved in the mishap do their best to avoid taking any personal responsibility or try to pinpoint the guilty party.

Creative, motivated people don't like the box that overly prescriptive roles put them in. Often when I ask such people what they think about their job, their role, or their position in their organization, I hear a common refrain: rigidity, constraint, lack of empowerment, even fear of failure.

In an age when we need to attract and nurture passion and creativity, we must do everything we can to eliminate the trappings of a bureaucracy that has long outlived its usefulness. If we want to create an environment where people focus on constantly learning and continually adapting to create value, we need to discard antiquated notions of jobs and roles. We need to avoid adversarial-style discussions of accountability and expectation. We need to refrain from approaches that start with us getting told what we can and cannot do.

SIX-YEAR-OLDS, TABLETOP PLAYERS, AND PROFESSIONALS: A HOCKEY METAPHOR FOR THINKING ABOUT JOBS

So how do we avoid creating an army of job automatons? First, a metaphor.

Who out there has watched their kids play hockey at a young age? It is certainly a sight to behold, and it is a lot of fun. Here are some common traits of a hockey team of six-year-olds:

- No one knows how/when to participate.
- No one is taking accountability, outside of coaches and referees.
- Everyone is interfering with each other.
- Teams do score (eventually . . .)!

How many of us have felt like we were working in the organizational equivalent of a hockey team of six-year-olds? Some commonalities between the two include:

- Everyone shows up to every meeting.
- Everyone is consulted on every decision.
- People come and go at random.
- Lots of overlapping and conflicting effort.
- No ownership of outcomes.

Where a team of six-year-olds has too little structure, a tabletop hockey team (you remember those, right?) has too much. In tabletop hockey, you have:

- Individuals with very specific job functions accountable to a very specific task.
- A much tighter organizational hierarchy, with prescribed spans of control at every level.
- Jobs and roles that are hard-coded; no one changes their role.
- The puck is constantly getting stuck in the corner, metaphorically landing in the gaps between people's job descriptions, where overburdened leaders have to pick it up.

How many of us have felt like we were working in an organization where we felt like a *tabletop piece*? Who out there has worked for an organization where management got serious about putting some rigor back into the organization but tried to fit everyone into heavily prescribed boxes? Likely you ended up with this kind of situation:

- Team members are told not to do something because it is

someone else's job.
- No one takes the initiative to improve things because it is the responsibility of some other department.
- Everyone needs approvals from higher up the hierarchy to get anything done.
- People get things done by flying under the radar and begging for forgiveness rather than asking for permission.
- There is a lot of departmental protectionism and blaming others for failure.

Now, compare that to a professional hockey team. Watching an NHL team, or even a team of committed amateurs, is an entirely different experience.

- Team members have positions, but positions change based on the play.
- The team outcomes trump any concept of role or position.
- Interactions between team members trump individual accountability.

Most of us actually have solid experience working in this type of organization, even if it's an organization that is outside the workspace. Some evidence you are on a true team includes:

- You don't check whether something is in your job description before helping.
- You proactively seek advice and help from people more knowledgeable than you before trying something new.
- You focus on getting to the outcome, not creating deliverables.
- Specialists and specialization exist, but specialists train and mentor others on the team and level them up.
- Leaders step in to help when required and then get out of the way.

LET THE TEAM DECIDE WHO DOES WHAT

In the modern age, we want to encourage people to play more than one role. Building on the hockey metaphor, we want them to play to score rather than just playing to a position. We want to remove roadblocks that interfere with people stepping outside of their job function. We want to increase the opportunities to provide value, as well as increase the chances to learn how to add value. The easiest way to encourage a new mode of thinking is to simply empower the team to determine what jobs and roles mean to them. Let them determine how to share responsibilities. Encourage the team to make roles and interactions a part of their discussions on how to improve as a team.

Consider this example of role churn at Scotiabank's billing lab, which was tasked with creating a consolidated billing system for its diverse suite of cash management products. Margaret Walczyk, the tech lead in the billing lab at the time, recounted her experience this way:

> *Being in the billing lab felt really chaotic in those early days; people were getting their backs up. I felt like I had to play peacemaker many times a day. And I was suffering a similar issue with the other "leaders" in the lab. There I was, the technical lead, with a project owner and a couple of seasoned architects. We had a small cast of product owners and a few scrum masters [the Agile word for servant-oriented team leads], all trying to "manage" a team of thirty-plus folks. It felt like we were overriding each other frequently, which negatively impacted others in the lab. It was a bit of a mess.*

At first, some of upper management (VP level) wanted to resolve the situation at Scotiabank by handing down roles, responsibilities, and accountabilities that were distinct for each role. The team suggested a different idea. They took a stab at coming up with their own answer. They started with the leadership group and facilitated an exercise in which each person listed the skills that each team member brought to their role. They found that there were a few unique things that really belonged to only one role.

But there was a lot more overlap in terms of what everyone wanted to bring to the table. They made significant progress by categorizing this overlap:

- Core skills: things that we identified as foundational to the one role
- Adjacent skills: things that we felt were core to our role but also core to other roles
- Peripheral skills: things that we could do in a pinch, and maybe even wanted to learn how to do

As Margaret later recounted, the team moved forward with the approach:

> *We did this in waves, waves with the entire team; everyone got a chance to illustrate what they felt were their core, adjacent, and peripheral skills and behaviors. This approach made clearer the things the team could rely on others to complete, but more importantly it became clear where the overlap was, where the team would need to collaborate to get something done, and where they would need to agree on who would do a piece of work in a more dynamic fashion.*

Perhaps most importantly, according to Margaret, outside of a few mandated responsibilities, it was the team who managed this approach. The result was higher trust. Not only did team members trust each other more, but the executives trusted the people in the lab.

As organizations grow larger—say, many teams, many teams of teams, hundreds if not thousands of employees—it might make sense to experiment with a bit more formalism than the "every team figures it out for themselves" approach. There are advantages to getting clarity on some foundational behaviors we want to see when someone is performing a role as a sales lead, a developer, a financial analyst, a call center agent, a product manager, or whatever. The trick is to avoid ending up with:

- The one job spec to rule them all! Everyone is bound to the definition of this job.
- Lots of narrowly defined jobs, a job for every function you can imagine!
- Jobs that are focused on defining individual contributions, often to eye-glazing levels of detail.

With conscious effort we can model our jobs in a way that is more conducive to a living, breathing workplace:

- Fewer job definitions
- Jobs are a loose container for a collection of roles
- Roles that overlap across jobs
- Team-oriented behaviors
- Breadth and depth competency for all team members, a concept known as T-skilling

GETTING TO ROLE CLARITY AT ALBERTA TREASURY BANK

A short while ago, the digital guild at Alberta Treasury Bank (ATB) wanted to increase the clarity of roles with their organization. Leaders within the digital group wanted to put some structures in place to make it easier for team members to appreciate the skills and behaviors that would help them progress their careers within ATB's digital group.

Michelle Kimoff, who at the time was the director of digital strategy at ATB Financial, was hearing several concerns from people across many of the squads. People felt that ATB could do more when it came to bringing clarity to people's work and roles in the organization:

> *The digital group had been taking advantage of Agile concepts for several years. The pace of tech change became quite high and was frequent. This brought about great benefits, but also some challenges. Some people felt like they lost what was expected of them; many felt*

it was unclear in terms of how to grow their career at ATB. Am I stuck doing back-end work, or can I enable others? We wanted to provide something to employees that would help them understand how they could move their careers forward. We also wanted to provide some of our more delivery-focused people with a career path outside of moving into people management. There were other paths. So we came up with a kind of treasure map that employees could use as a guide for people to move and grow. For example, a developer may progress by becoming more of a full-stack developer [a developer that can work on both front- and back-end parts of a software solution], a tech practices coach, or an architect or move into product or delivery roles. As a side benefit, we wanted a way to show the more traditional parts of the organization outside of digital how skills-rich our delivery professionals needed to be in order to justify the kind of pay these types of roles demanded in the market.

Will Davis, a consultant at Agile By Design, was tasked to work with Michelle and the rest of the digital guild to put together a "role clarity" framework to help people navigate some of these issues. He says:

When I was first asked to help Alberta Treasury Bank on the role clarity framework, I remember being both a little excited and a little trepidatious. I've seen organizations do this kind of role definition work before, and frankly the output tends to not be that great. I have seen some role work that is embarrassingly bad in some cases, and downright toxic and counterproductive at worst. There are a lot of enterprise leaders out there that seem to think that we can define roles and responsibilities using the same rigid mindset and approach that we did in the old waterfall days, but replace it with Agile jargon. If we do that, we don't really change anything.

As Will started to engage with various members of squads within ATB, he encountered some very typical issues. Some team members were complaining that they felt overtly tied to the role they were playing. Others felt that the pathway to progressing in their career was not as clear as it could be. Many felt it was hard to change to a new path, if one so desired. Will was also troubled by the amount of old-school thinking about jobs and roles he encountered. It appeared that some did not want to take on things that fell out of the scope of their job. Others were wary of stepping on other people's toes. Most troubling was the number of people that would indicate they didn't know what their role actually was.

ORGANIZING CONSTRAINT

Define Coarse-Grained Jobs in Terms of Many Overlapping Roles

One thing we can do to avoid the pitfalls of strictly defined job descriptions is to simply have fewer job definitions than we ordinarily see in an organization—a lot fewer. Since we have fewer of them, we can make those jobs expansive, with lots of options. Job definitions can provide for lots of different kinds of work. Take this principle to its extreme and you end up with organizations that only have one job definition: employee.

Valve, a video gaming company in the US, is the maker of widely successful game franchises such as Half-Life, Portal, and the social-distribution network Steam. While they are a privately held company, they claim their profitability per employee is higher than Google, Amazon, and Microsoft. With over 360 employees, there are no official jobs defined. Or really, they have one job defined, which is a person who has a job at Valve. You are hired because they trust you to add value based on your skills and experience, and they trust you to figure out how to work with others to do so.

Organizations getting started with this approach may take a slightly gentler approach than Valve—perhaps three, four, or maybe five job descriptions. Each job would have a very wide scope of responsibility, within an overarching functional category.

So, stated fully:

Design job specifications so that people can play many different roles.

As Will got started with his task, he remembered the chapter he had read in the book *Management 3.0* by Jurgen Appelo on growing structure. In his book, Jurgen talks about how adaptive organizational structure can be grown by focusing on developing people with T-shaped skills, and using a minimum number of job titles that are widened in their applicability to encourage a balance of generalization over specialization.

Using these principles, it was pretty easy figuring out the jobs for ATB's digital group, according to Will:

> *ATB Digital [was] a delivery organization. Their mandate was to build and improve on Alberta Treasury Bank's digital channel so that customers could bank online with the best possible experience. So when it came right down to it, we started with just jobs:*
>
> - *Engineers: the people who performed the creation and operations of our technology solutions.*
> - *Production: the people who focused on achieving market success with our technology solutions, including supporting customers.*
> - *Delivery: the people who were there to support the other jobs. They would do whatever it took to keep things moving, including facilitating, communicating, and visualizing the delivery of working software.*

> *And that was it. Everyone could fit into those three jobs. Every possible permutation of what people could do to create value could be slotted into one of these three jobs. I wanted to empower people to flesh out the details in terms of level, behaviors, roles, etc. But I wanted official job titles to be limited to those three.*

Jobs are really just containers that house a lot of the different types of work that one can do as part of that job. One option is to define all the roles that could be considered within the scope of the job in question. It is a good idea to keep these roles fairly small and detailed. Figuring out the roles was where the real work was at ATB.

Michelle talks about the need to move from jobs to roles:

> *Your job doesn't define the work you do. On paper you might be a developer, but back end versus front end versus lead. We wanted something that allowed you to pull on whatever skills you had to perform the role that you were able to play. More-progressive organizations have work boards, where people choose their role every day. Maybe we couldn't get all the way there in our more hierarchical organization, but we wanted to get at least partway there.*
>
> *So our approach was a kind of Snakes and Ladders for roles, only with no bad moves. People needed to look at all the roles and skills the organization needed and say, I know what I am currently doing. I have looked at my skills and see the needs in my squad, in my tribe, both vertically, in terms of people leadership, professional skills, etc. You own it rather than relying on a manager to tell you what it is going to take to get to the next level. You can map out your career your way.*

We can add clarity to these roles by fleshing out what a role does in the context of team dynamics and interactions with others. Feel free to describe roles in terms of behaviors that overlap with those found in other roles. Finally, we need to empower teams so that they define and refine any standard definition of jobs and roles needed to meet the

context of the team. Dynamism must trump consistency!

Michelle noted how empowering volunteers from squads to shape the various roles was a key part of the approach:

> *Our success at ATB Digital came from the fact that we brought in a lot of members from our Agile squads to contribute to the work. At ATB we tend to talk about co-creation in the abstract, but I feel we really lived that principle when coming up with our role clarity work. The teams really focused on the level of definition that would be helpful. This might have taken longer than if management and consultants cooked something up in a room, but the result was that people felt OK with it and not just slapped in the face with something.*
>
> *Back in my old Deloitte consulting days, I did a lot of competency modeling and job description writing, squirreled away in a back room at a client's head office. While the work was helpful for some, it didn't put any power to leaders and the employees to make meaningful decisions. The work had become out of date by the time we built it. Some could argue that roles like a branch manager or customer service rep may not change a lot. But in digital, things change much faster, and traditional models just don't keep up.*

Will remarked on those early acts of co-creation too:

> *The first thing we did was make it as easy as possible for anyone in the digital group to tear down our work in an act of creative destruction. We created a wall of sticky notes with fundamental behaviors. We had behaviors for every role we could imagine. We deliberately left the role mapping on the wall. We then asked team members from across the digital group to self-identify with behaviors, group behaviors into roles, and discuss where roles overlapped and how one could play a different role.*

Roles and their behaviors for a team are best identified by the people who are part of that team. Required behavior can and will change based on the emerging needs of the team. It is critical that individual teams be able to tweak, and even end up changing, the model as necessary to come up with the set of jobs, roles, and behaviors that fit their context. Again, teams need to self-organize to determine which team member plays which role. None of this is a one-and-done discussion, but something that is often revisited as part of team-level planning and review sessions.

We increase agility when individuals in the team pair with each other to play different roles across the life cycle

Meet Mary...

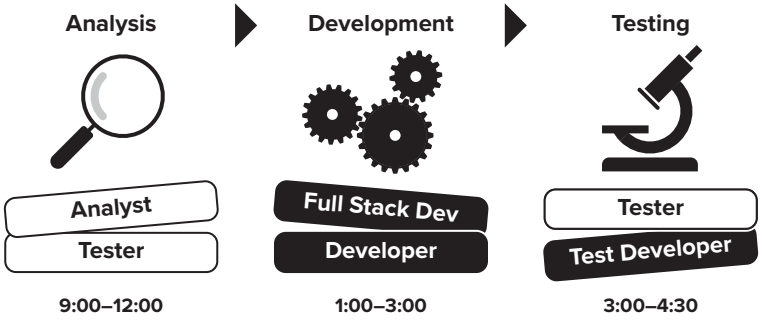
Mary Swanson
JOB TITLE:
Developer
SENIORITY:
Intermediate
TEAM:
Microservices

Over the course of the day, the story moved through 3 phases of the build – Analysis, Development & Testing

Mary played 5 different roles in 1 day!

Yesterday, Mary collaborated with a squad member to complete a story from the backlog . . .

9:00–10:30	Reviewed a story with an analyst, broke it up into two stories with a team member
10:30–12:00	Documented the new specs and test cases for one of the stories
1:00–4:30	Developed the new service and tested it manually
4:30–5:00	Wrote test script to automate for the feature



I can do it, I really can . . .

The behaviors, roles, and jobs that came from consolidating the output of all these workshops wasn't necessarily groundbreaking or novel, but the approach solidified buy-in. It provided an opportunity for people to be educated on both the consulting and the ATB side.

FROM JOB AUTOMATONS TO HUMAN BEINGS

So, let's take another look at the previous job automaton example we defined at the beginning of this chapter. Remember that analyst job description that just made you want to cry? Let's see if we can describe the job a little bit differently.

First of all, we will change the job title to something a lot broader. Let's call the job *business specialist*.

Business Specialist

- *The business specialist brings ownership, structured thinking, facilitation, and communication, with a focus on aligning the business objectives of the team.*
- *The business specialist collaborates with business stakeholders to validate that the work of the team is accomplishing business objectives.*
- *The business specialist works closely with other team members to clarify and validate that the work of the team meets business needs.*
- *The business specialist facilitates workshops with the rest of the team and business stakeholders.*
- *The business specialist often engages closely with real users to gather insight and feedback.*

The business specialist can play any of the following roles as befits their experience and knowledge:

- *Analyst (documenting required functionality)*
- *Tester (validating functionality)*

- *Domain subject matter expert (providing expertise on functionality)*
- *Product owner (prioritizing functionality and determining business fit)*

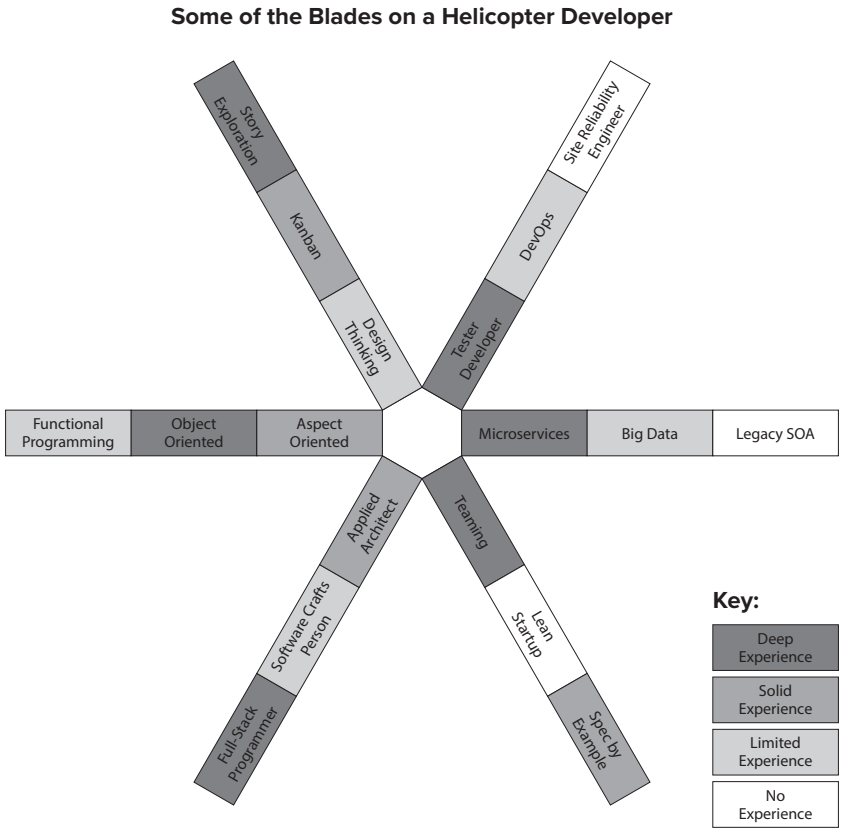
Depending on their aptitude and interest, a business specialist may start to focus on user experience (UX) or facilitating customer insight as a design thinker. The entire team is expected to collaborate and collectively identify the roles that the business specialist can take in order to serve the team's goals.

See the difference here? Again, this job description is broad, it provides options, and it is described from the perspective of interactions and engagements with others.

GROW HELICOPTER-SHAPED PEOPLE

Previously we talked about how we want more than just cross-functional teams; in other words, we want a team of diverse individuals. We want cross-functional *individuals* inside of our cross-functional teams. We want people learning, growing, and expanding their skills. As people become more senior, we want them to expand their expertise at a reasonable level across a number of other knowledge areas. Yes, we want them to gain deep knowledge in one or two disciplines, areas of deep expertise, but this should not come at the expense of broadening their skill base.

In other words, we want to encourage the development of *T-shaped people*. Your knowledge literally expands deep and wide, like a T. Perhaps a better description is *helicopter-shaped people*. We want people to go wide in a number of directions. Your depth of expertise still matters, but its real value comes from educating others on how to apply it. Mastery is really about pairing, mentoring, and teaching others to apply your domain. The narrow roles and wide job approach lend themselves to the development of T-shaped people, but it is really just one blade of the helicopter. We want to encourage people to expand across all the blades.



Some of the blades on a helicopter developer

Michelle gave a great example of helicopter skilling at ATB:

> *It was important for ATB Digital to move to this idea, but it was tricky putting this thinking on paper. We talk about roles, but what about the stack? What about the business domain? It was important to understand where depth was needed versus which areas needed more breadth. As things evolve we have found we needed more people to have breadth for things like making changes to our core banking platform versus everyone being super deep in one area.*

After Michelle and the team felt they had a good taxonomy for jobs and potential adjacent roles people could move into, they asked squad members to come together and co-create the skills each job would need. This filled out a set of job-specific helicopter blades, in effect defining a helicopter for each job. A self-assessment tool was built where people in digital could define how they wanted to expand their role, skill sets, and other capabilities. Managers were trained on the approach and coached to advocate for their people in their learning and growth objectives. This is allowing people to define personalized career paths to grow into the areas that drive their personal interest.

The feedback received was very positive. Some people even indicated that the dialogue and attention to people's growth had motivated them to be a part of ATB like never before. But there were also challenges, according to Michelle:

> *Self-assessments can be terrifying for people unless they know how they are going to be used. Unless you have trust, people can literally freak out. We had to spend some time getting people's trust that this was for people's betterment and not something to be used against them. We started by being really transparent about the outcomes. We found a group of eager cohorts and dived in, taking the plunge to use it for positive things. It proved to be a simple way to get a baseline, and a simple way to keep it. In the end it led to more work that people wanted to do, to more training.*

IN SUMMARY

- Jobs in the traditional organization are overspecialized and specified and are a poor fit for the collaboration and dynamism required to succeed in an uncertain world.
- Think of jobs like a professional sports team: we need both expertise and pinch-hitting in order to succeed!
- Design jobs so that we can encourage people to play many different roles, defined around behaviors that focus

on collaborating with other roles to achieve valuable outcomes.

- Moving to a more dynamic job model can be done incrementally.

5

Truths, Myths, and Lies About Teams

As I hope to have convinced you by now, designing your organization around cross-functional, self-organizing, market-facing teams trumps designing your organization around functional departments and hierarchies. It is how we must organize if we want to take advantage of our humanity to create value, to organize in a way that asks us to challenge each other, and to learn and grow through exposure to diversity.

But in a larger enterprise setting, placing people in Agile teams alone is often not enough. While it is only the first step of many, it is an incredibly powerful first step. We will need other organizing structures as well.

This will be obvious to some, but I still see too many organizational leaders (very senior leaders at that), guided by big name consultants, proceed to stand-up teams, boot-camp them on some basic Agile teaming practices, maybe (rarely) pair with people on some technical practices, then sit back and wait for the magic to happen.

And wait. And wait.

Many organizations manage their Agile teams with a traditional management layer, a hierarchical organization. They struggle to make Agile teams truly work for them in an enterprise setting. They struggle to grow teams that exhibit independence, to build teams that are responsible for end-to-end value creation, and to foster teams that can

operate with true autonomy. The enterprise world as it exists today resists such simple solutions as the Agile team, as much as we wish it were otherwise.

When moving to Agile teams in an enterprise setting, organizations often encounter some very real, almost intractable problems:

- Fragmented software architecture leads to a spaghetti of team dependencies, no matter the team structure we use. So much for independence!
- Crazy complicated initiatives that require a revolving ad hoc assortment of part-time specialists. So much for dedicated team members!
- Outcomes that require significant contribution from many people: Thirty people! Eighty people! Two hundred people! So much for small teams!
- A diverse set of stakeholders, and none of them seem to agree on anything. Good luck picking a dedicated product owner!
- My teams are all going in different directions on every little thing!
- My teams can't deliver unless we get approval from legal/compliance/architecture/Agile coaches/quality management/release management/you-name-it-and-they-can-veto-us. So much for self-organization!

To address these challenges, we need to tackle what have become sacred cows in the Agile space. We need to explore some uncomfortable ideas.

THERE ARE NO INDEPENDENT TEAMS

OK, time for our first sacred cow, the idea of the independent team in an enterprise.

There are no independent teams, but there can be autonomous teams. Sound confusing?

Take a second to think about the value your team is trying to

create. Perhaps we are talking about a software application delivery team. The team has accountability for everything from prioritization, through delivery, all the way to rollout and operations. Sounds like a truly independent team, right?

Now think about all the things that team needs to deliver on any one increment of value. Is there another team in our organization that does market or competitive research? Do we rely on this research to make decisions? Should we? Is there a common business model the team is leveraging? Do other teams have input into it? Do we have input into it? Some other questions we could ask:

- Is there an overarching strategy being used to guide our work? Do we help build it and validate or invalidate it?
- Are there common cultural artifacts that we share?
- Do teams share a common physical workspace?
- Do multiple teams want to or need to share a common software platform, even if that sharing is in a completely voluntary or autonomous way?
- Do other teams help set that up?
- Does the team share any common data with other teams?
- Do teams share common customers?

Tired of these questions? Hopefully I have made my point. In an organization of any scale, the answer to at least some of these questions will be yes.

STABLE TEAMS ERODE AGILITY

Now that we have taken a shot at team independence, let's take a crack at team stability. I'm sure this one will get some people's backs up.

Stable teams erode agility. There, I said it.

This is a strong statement. I admit to trolling a bit to get your attention, and while I have it, I want to ask you to think about what Agile is trying to address.

Uncertainty. Complexity. Change.

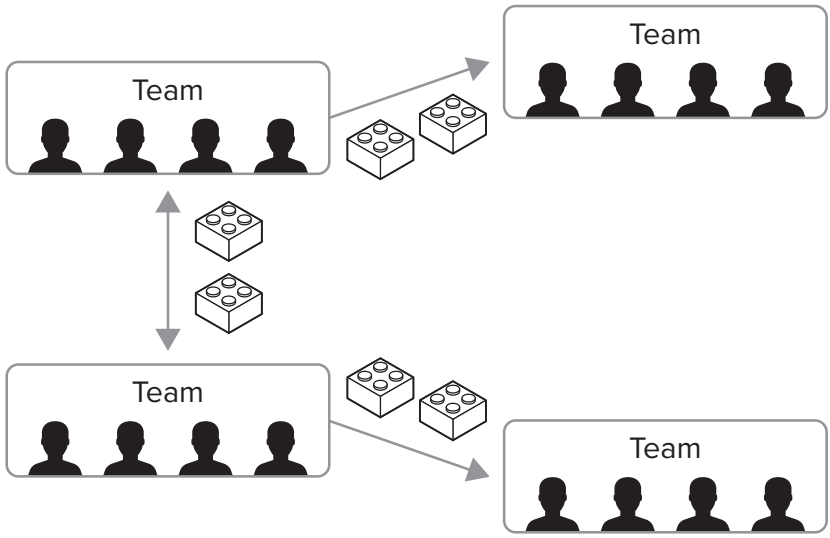
Perhaps some other team has those missing skills we need to

deliver value. *Couldn't we send that portion of the work over to the other team?* We could, but now two teams need to coordinate to deliver value, which *interferes with each team's ability to independently self-organize*.

Maybe a piece of work involves more effort than we thought. *Can't we keep the team the same size and take longer to deliver?* We could, but we may *delay feedback and learning*, and depending on how important that feedback is to the team, we could end up delivering the wrong thing to the market.

Perhaps we have discovered a new opportunity, unlike anything we have done before as an organization, and delivering it would require collaboration that spans across every one of our teams. *Couldn't we just coordinate the work across the teams?* We can, and in the short term we likely should, but in a way that takes advantage of the various teaming and team collaboration patterns mentioned so far in the book.

But as time passes, and our work increasingly defies our existing team structure, the mismatch creates a dissonance that is increasingly hard to navigate. Managing the difference between our dynamic teaming structure and official team structure can start to take on a life of its own. We start entertaining ideas like coordination requiring classical, old-school management—one that is now informed by Agile backlogs, stories, visual flow, and so on, but is still old-school management. The more the teams start depending on each other, the more we start to rely on an old-school kind of management, and teams lose their ability to self-organize.



All these dependencies look nice in a diagram; the reality sucks

No matter what, your team structure will always be wrong. That's because we are working in a world where change is constantly accelerating. Our team structure, no matter how well designed, is always going to be wrong.

Sometimes it will be a little bit wrong. Sometimes it will be very wrong. It will never be perfect, but that's OK. Just keep changing according to the needs of your team and your ecosystem.

AGILITY DOES NOT COME FROM AGILE TEAMS

How can we say agility does not come from Agile teams? Didn't I say previously that Agile teams and Agile *teaming* are core building blocks for our new organization?

Hear me out. A lot of organizations focus on the team to the exclusion of all else. This is too myopic a view. How is the team velocity? What are the team's practices? Can the team meet the team's sprint (a short-time iteration of delivery for a team) commitments? How did the

team retrospective go? Is the team blocked? What are team-level impediments? What is on the team backlog? How do we protect the team from everyone else in the enterprise? How do we make sure the team is successful, even if the organization isn't?

There is nothing wrong with a strong team focus. We *want* a strong team focus. But we want to avoid a team focus becoming a *team-only* focus. Then our teams would just become another set of siloes working in isolation. When that happens, we might as well go back to our original machinelike functional department–style organization.

Unfortunately, Agile has a track record of successfully optimizing the health of individual teams but having no effect on organizational value creation. In some cases, team-level optimization has even caused harm to organizational value creation.

Let's take an example: *Team software* is a software delivery team that works with *team training*, a team that delivers end-user training to support users based on that software. Team software is rocking it! They deliver and they deliver quickly. They demo with some real support users every month. They can and do deploy on a dime.

But team training just can't keep up. They can't get support users trained fast enough to keep pace with all the changes to the software that team software is making. Worse, team training starts to compromise training quality to maintain pace. This leads to poorly trained support users. These support users make mistakes using the software, which causes real problems with actual paying customers of the business. As a result, customers stop relying on our business, and demand for team software's product goes down. Even worse, market share for the business is lost and profitability goes down. None of this is apparent from team practices, team metrics, or a single team focus.

BUT (AGILE) TEAMS *DO* WORK IN THE ENTERPRISE

Before we lose all hope, let me pause and say the concepts of Agile teams and teaming *can and do scale* to meet these and other challenges we see in the enterprise.

We can use team and teaming concepts to evolve our organization toward greater agility. Confused yet? Stay with me, and hopefully

things will become clear.

Agile provides us a means to deal gracefully with a world that is complex and constantly changing, and it does so through a mindset and through methods based on using feedback to respond effectively to this constant change. Inside an Agile team, I have talked about how people constantly reorganize to team in a way that allows them to respond to this change.

I have seen a lot of organizations do a reasonable job of figuring out their Agile team structure yet eventually end up with a mess of dependencies down the road. Hypothetically, let's say you have analyzed the demand for your organization, mapped it to capability, formed an ideal set of teams around those capabilities, and staffed those teams with people that possessed those capabilities. And you actually got it all right! Despite your enterprise setting with its dizzying array of business, product, and customer stakeholders, the shantytown of supporting systems, and arcane maze of regulatory and compliance expertise needed, you managed to set things up so that there are but few dependencies between teams. The teams all have whom and what they need to be successful.

We don't need or want *cumbersome Agile scaling frameworks* to do this. Many of these scaling frameworks attempt to address the need for more people to collaborate, for work to scale, by adding a layer of the quote "Agile" processes to the mix. Some of the ideas coming from these frameworks are pretty decent, and some are frankly quite ridiculous (*ahem: normalizing story points across teams*). We also don't need to scale Agile with waterfall thinking, methods, or tools.

Instead, scaling Agile teams requires that we *take the time to understand the reasons behind why Agile works and why Agile teams work*. We need to understand what aspects are inherently scalable. More importantly, we need a strong appreciation for which Agile concepts are proven to be dogmatic and need to be taken with a grain of salt, whether we are talking about scaling or not. We are then in a position to extend the underlying principles of Agile teams that work and apply them at larger and larger scales.

When we talk about scaling Agile, what exactly do we mean? What is the "thing" we are trying to scale exactly? In my experience, *we are scaling the ability for our people to form into hypercollaborative groups*

that leverage rich (market) feedback to co-create value. In the coming chapters, I'll go over some design constraints that can help you scale what is great about Agile teams while inhabiting an enterprise that inherently resists such concepts.

IN SUMMARY

- Many organizations struggle to scale Agile teams. Adding a layer of traditional or ad hoc management structure does little to help.
- In an enterprise, it is rare for teams to be truly independent from any other team. Expect dependencies to happen.
- In the face of complexity, your team structure will always be a little bit wrong. Static team structure will result in a loss of agility over time.
- Your organizational agility won't come from improving the agility of your teams in isolation.
- Agile teams do work in the enterprise; group people into social-bounded contexts, manage dependencies through teaming, and deliberately evolve team structure to avoid handoffs.

Part II

The Ecosystem

6

The Ecosystem of Agile Teams

Picking up where we left off with the payments group at Scotiabank, the team was going through a phase of rapid growth and expansion and beginning to become a victim of its own success. Official sessions were long and started to feel bureaucratic. Interestingly, the real cadences happened through informal sessions, taking place at intervals.

Even with visual workflow and other information radiators, things started to feel chaotic as the team size grew. The Kanban became hard to navigate, and some people were multitasking way too much. Sometimes work would be temporarily lost in the shuffle. Product owners and scrum masters were especially stretched. In short, things were becoming unwieldy. The team went about discussing how to put some additional structure in place.

I'll use the next section of this chapter to discuss how to grow structure to meet growing demands on the team, using the team at Scotiabank as an example.

SAFE, EFFECTIVE HYPERCOLLABORATION AT LARGER SCALE

OK, so your team needs to scale. Let's break out the scaling framework,

right?

Wrong.

What the "more Agile process is required to scale" camp seems to miss is that Agile teams work because they create an environment where people work well with each other. If you recall in my chapter on teams and teaming, the whole point of Agile teams is to create a space where people can form into hypercollaborative groups to co-create value.

Agile takes advantage of both method and structure to foster an environment of teamwork, collaboration, and having each other's back. Agile done well results in an environment of high *social density*.

Market contact, transparency, and feedback all contribute to this concept of social density. This idea of creating spaces for hypercollaboration is an Agile concept that can and does scale.

Social density is a term I first learned about while reading Niels Pflaeging's work on the dynamics of organizational physics, much of which he discusses in his book *Organize for Complexity*.[17]

Social density is a concept that scales.

Instead of scaling by adding controls, think of scaling as an act of ensuring the right collaboration is happening beyond the scale of a single team. Look for ways to scale social density to the larger organization. Identify opportunities to apply the team and teaming concepts to larger contexts to make it easier for people to collaborate across team boundaries.

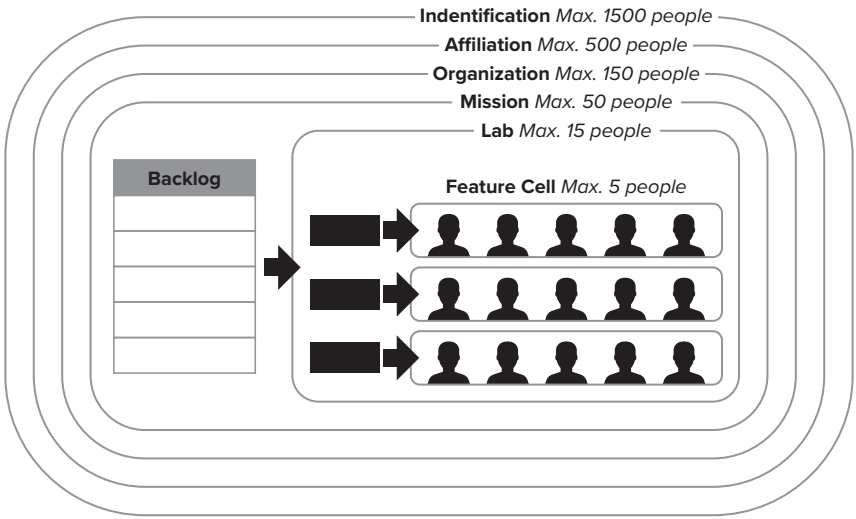
SOCIAL DENSITY THROUGH CONTEXT BOUNDARIES

Some relevant research to draw upon when thinking about these larger contexts is Dunbar's numbers, named for scientist Robin Dunbar, who studied social interaction. Dunbar's numbers suggest that there is a cognitive limit to the number of people one can maintain stable social relationships with. It suggests that close-knit groups involve approximately 5 people. A person's socially active number, one where they maintain a fair amount of social contact, is about 15 people. A

17. https://www.nielspflaeging.com/books.

larger network, one where people can still interact and engage with each other at some reasonable frequency, caps at about 50 people. The total size of a network with which one can share a common mission or active identity should not go much past 150 people. Past that point you are talking about broader and more cultural forms of association; this would be akin to something like living in a small town, or being part of an association, which spans 500 to 1,500 people. When you get to these numbers, it is unrealistic to expect any form of meaningful collaboration.

We can use Dunbar's numbers as a guideline to set up organizational *context boundaries*. A context boundary is a boundary we can place around a group of people that shares a particular context, a common outcome, a shared platform, or the same problem or solution space. Larger context boundaries contain smaller context boundaries: in other words, contexts can and do nest.



Agility comes in all sizes

For instance, a *feature cell* is an example of a context boundary that should not contain more than 3 to 5 people. A *team* is a context boundary that should reasonably max out at around 7 to 9 people, and it gets pretty unwieldy when we get close to 15. Multiple teams can often be linked by a common *mission* or shared outcome. Referred to

as a *tribe* in Spotify lingo, this group can often have between 30 and 50 people. I call this association of teams an *ecosystem of Agile teams*. Past those numbers, we start thinking in terms of (sub)organizations, groups of 150 people or so dedicated to a common business or higher-level value proposition. These organizations are sometimes part of a larger enterprise, whose numbers span to the hundreds and thousands of people. As per my previous comments, it's best to consider that these numbers are general guidelines only: context matters and culture makes a huge difference.

Back to our intrepid team over at Scotiabank: members went about discussing how to put some additional structure in place to manage what was a rapidly growing group. We noticed that the larger team was organically separating into smaller groups to deliver different features. The grouping wasn't always clean and we saw all kinds of churn and confusion, but we also saw the kernel of a good idea.

The team, for the most part, felt they all had a common identity that they wanted to keep. There were many synergies across the work: common platforms, the same customers, shared stakeholders, and a common purpose.

The answer was to divide the team into three separate but closely associated pods. Each pod would be responsible for the detailed analysis, development, and testing of small numbers of features over time. Members from each pod would rotate to determine who would pair with the senior product owner and architect to participate in a common idea shaping and intake flow. Once delivery started, team members would work primarily in their pods, except when dependencies across pods were identified, in which case people would pair across pods. A common planning and demo cadence was held at the ecosystem level, as well as a weekly stand-up.

We used the team's Kanban and visual backlog as our design canvas. We created swim lanes for the separate pods, identifying the parts of the flow that would be worked on by the separate pods (e.g., feature delivery) and what parts of the flow would require tighter integration across pods (e.g., idea shaping and demo and release). Eventually, pod members chose to have their own Kanbans to track story-level work, as well as a common Kanban to track overall features for the entire payments group. With a minimum of fanfare, the *payments ecosystem*

was born!

When rethinking your organization, there are two levels of context boundaries you will want to pay particular attention to early on. The first, obviously, is the team. The second context boundary is sometimes less obvious: the *team-of-teams* structure. This is what I am going to call an *ecosystem of Agile teams*, or just an *Agile ecosystem* for short.

An Agile ecosystem is an ecosystem made up of a small number of market-facing teams, along with dedicated teams (or people) allocated to support those teams. We can also think of the external stakeholders—and even market actors that the Agile teams interact with—as being part of an Agile ecosystem.

The Agile ecosystem is defined by a common context boundary that serves to establish a naturally cohesive grouping of teams, establishing a shared identifying element based on one or more of the following:

- Shared mission or outcome
- Shared platforms or systems
- Same business units
- Common customers
- Common customer life cycle events (e.g., onboarding)
- Common business portfolios
- Common business models

Agile teams will often coalesce into an ecosystem structure over time, either intentionally, organically, or accidentally. By being intentional in identifying our ecosystems, we try to contain cross-team dependencies to the teams that are within an ecosystem. In an enterprise setting, containing dependencies to a smaller social circle (e.g., the ecosystem) is often more tenable than getting to truly independent teams.

Using Dunbar's numbers, we still want to keep the ecosystem contained to a reasonable number of people, with well under fifty people being as good a guideline as any. Once our ecosystem starts to grow past five to eight teams, maintaining a shared identity becomes problematic and alignment can start to become unwieldy.

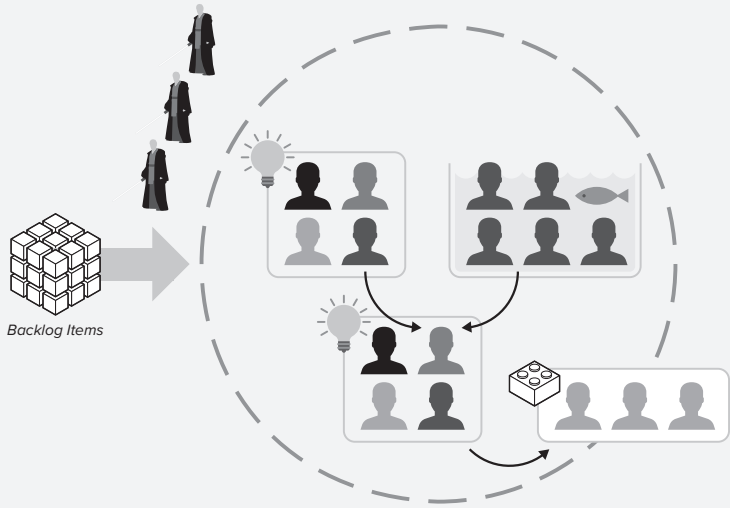
Ecosystem team members are responsible for facilitating alignment across teams. Organizations new to Agile methodology often assign this task to people with positional seniority on the teams. However, part of the journey involves inviting all team members to participate, perhaps on a rotating basis, through open spaces, or through grassroots participation in tending to ecosystem-level management.

Information radiators within an ecosystem, such as a common backlog, visual management, and Agile-style cadences, can help teams stay aligned with each other. The perspectives guiding these practices are coarser than when using these practices to guide a team. For instance, instead of stories, the ecosystem backlog and Kanban system will typically contain epics (larger buckets of scope containing many stories) and releases, or perhaps individual features and thin slices. The lead times of work discussed may span months and even quarters, especially for organizations not yet accustomed to rapid delivery.

PATTERN

The Agile Ecosystem Member

The *Agile ecosystem member* is exactly what they sound like, a member of a team that is part of an Agile ecosystem. They may be part of a market-facing team or someone responsible for providing support to one or more team members. Either way, the majority of the work they do is dedicated to achieving the outcomes of the ecosystem they belong to.



They always try to deliver value to the market. However, there is often more work to be done than could be accomplished by one single team. As the Agile ecosystem member takes on the extra work required to scale, they spin up additional teams.

Invariably, teams run into challenges as they start duplicating each other's work, misaligning on outcomes, and in some cases even causing conflict with each other. The solution is to group teams that are focused on a common overarching outcome into a single context boundary, namely for the outcome trying to be achieved.

Thanks to its size, the Agile ecosystem can be given larger responsibility than a single team often can. The Agile ecosystem can truly own an outcome, including operational activities, customer interactions, marketing, and support. We want to do everything we can to minimize dependencies across ecosystems. For this reason, we want an Agile ecosystem to have dedicated support embedded whenever we can. We want ecosystems to have the right level of leadership to make real

market decisions.

This will not always be feasible, but it is a design principle worth keeping front and center. Here are some potential options for organizing your Agile ecosystems to take full advantage of the strengths of the Agile ecosystem members.

The Microenterprise

Microenterprises can be thought of as miniature organizations within the larger enterprise. They are the idea of independent teams taken to their fullest extent.

I can't remember the first time I heard the term *microenterprise*, but I am very fond of the overview and description provided by Joost Minnaar and Pim de Morree in their excellent book *Corporate Rebels*.[18]

"Microenterprises are expected to engage their users throughout the entire business process: from R&D to product design to production and delivery. Users can offer feedback at any time in the process. In this way, Haier (a company that organizes into microenterprises) wants to turn the one-time customer into a lifetime user."

According to the authors of *Corporate Rebels*, one of the keys to the success of microenterprises is setting them up as open platforms. In this way, teams can connect employees and users directly to each other via the internet and digital tools.

The microenterprise is the pinnacle of business agility. Successful microenterprises grow within the overall enterprise, potentially spawning new microenterprises and allowing unsuccessful ones to die.

18. https://www.corporaterebelsbook.com.

The Outcome Team

Another North Star in terms of agility is the idea that an ecosystem is no bigger than a single team. Each team has an independent outcome and is effectively operating as an entire ecosystem, completely independent from other teams.

Sadly, I have rarely encountered this in my experience of traditional enterprises making the Agile journey. Departmental protectionism, dysfunctional back offices, arcane technology, shambling architecture, changes too big for one team to absorb, fragmented business models, along with a maze of bureaucracy and dependencies all act as barriers. But this is an ideal worth striving for, and there are numerous examples of organizations who have teams that have achieved it. Why can't you?

The Program/Portfolio

For organizations that are following a more conservative path to agility, we can take the more traditional concepts of a large-scale program or a portfolio of initiatives and supplement them with an Agile mindset. Many enterprise programs and portfolios have more control over how they deliver. They are staffed by a diverse mix across the organization and are outside of the traditional organizational structure. They have a common funding model that is based on achieving larger outcomes than individual projects. There is a healthy backlog of work and a real challenge in coordinating it.

For these reasons, a program or portfolio can serve as a prime candidate for organizations starting to look for places to stand up a cohesive ecosystem of Agile teams, supporting enablers, and traveling team workers (see next chapter for a definition).

MAPPING OUT AN AGILE ECOSYSTEM

You will likely find yourself in a situation where there is a desire to move to a more Agile, team-based structure in a less organic way than the one taken by our payments team over at Scotiabank. Someone has made the decision to accomplish an outcome or set of outcomes through Agile teams. Both positional leaders and on-the-ground employees are ready to try something different; there is an opportunity to make a bigger, more significant step toward working in an Agile team model.

In this case you may want to take some shortcuts. Growing an organizational structure using an evolutionary approach, as the team gains a deeper and clearer understanding of the work, can take a long time. Where there is a desire to make a more profound shift, we can consider how to do this without knowing the throughput of the system or having a true understanding of where failures to collaborate are creating bottlenecks.

In the spirit of moving forward with imperfect information, we can work with volunteers to map outcomes of both work products and capabilities. We can then collaborate to identify people and their team structure that will bring these capabilities to bear.

We talked a bit about the Agile journey CCC has been making in chapter 3, "Teaming Is a Verb." Let's take a step back and look at CCC's larger transformation. Brent Reynolds, the senior executive responsible for CCC's branded cards and digital studio groups laid out the reasons his organization had both the desire and the need to move to a different working model:

> *"For the past several years the need to remediate some critical compliance-related issues had caused leaders to exercise very tight control in the organization. We had addressed the regulatory findings primarily through process rigor and quality controls. This was needed to address our operational risk, but we were really at the bottom of Maslow's hierarchy."*
>
> *Jay Archera, an executive in CCC's partnerships card business, discusses the resulting environment:*

"We were quite prescriptive: every artifact, every process, was documented in a lot of detail. Command and control did work in that we were able to put out the various organizational fires that we needed to, but it was getting in the way of organizing for a future that was fast coming. This all made sense at a point in time, but as we matured, it got in the way of being more nimble—a very expensive set of training wheels, so to speak. The good news was that leadership decided to put aside what was going on day to day at CCC and ask, How is the world changing? How is software disrupting banking? What do we need to do to become a digital organization?"

Brent agreed that the sentiment for a change was there. "Our parent organization had been asking these kinds of questions for quite some time and had been making substantial changes as a result; here in Canada we were only tiptoeing in the Agile waters. Yes, we had been experimenting with Agile and had proved that could help us on both complex and critical projects. But once the project was done, people would go back to working the way they always had. All of this meant there was a lot of pent-up demand for people to work differently. The idea of working in Agile in pods had a lot of appeal for our people. Atlas had shrugged. We were ready to organize our people into Agile-style pods."

So, we have the enthusiasm for a larger change, something more systematic and structural. Thinking through how this could look could end up being a very complex endeavor. How do we want to get started? How can we best facilitate the discussion? Jay Archera approached Tyler Motherwell, a fellow Agile coach, and me to help answer these questions for the people in the partnerships card business group in CCC.

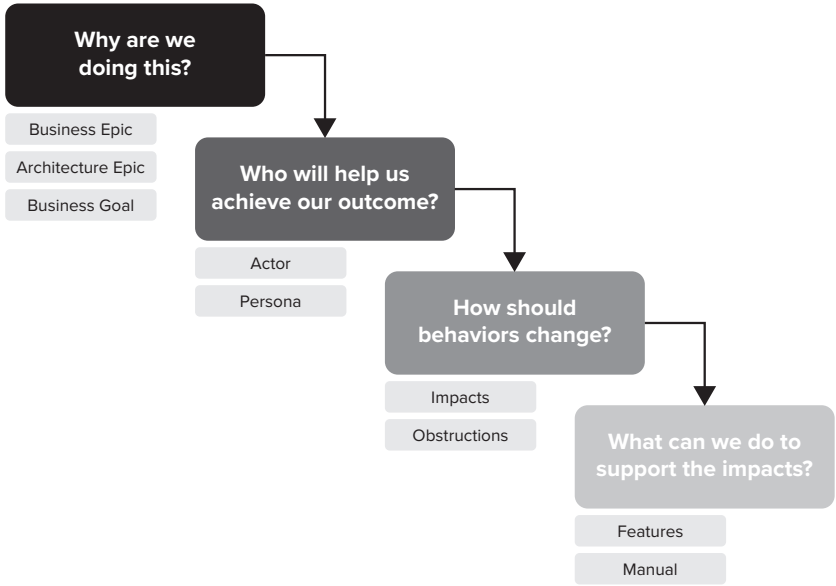
We felt that a modified version of impact mapping,[19] an Agile prac-

19. Gojko Adzic, *Impact Mapping: Making a Big Impact with Software Products and*

tice invented by Gojko Adzic,[20] would be a great way to start this work. Throughout the rest of this chapter, I'll use the story of how Tyler and I worked with Jay and other leaders from CCC's partnership business to help them reorganize the people in their group into multiple ecosystems of Agile teams.

Impact mapping is a technique in which you map the behavioral changes (impacts) of your users in response to organizational goals. Once you group impacts by actors, personas, or user categories, you then add deliverables that could support those behavior changes. Related items are connected visually through a mind map.

Tyler and I agreed that we could use impact mapping to support a design approach in which we focused on CCC's market actors and how to organize around the impacts we wanted to have on them, an *outside-in* approach to organizational design.

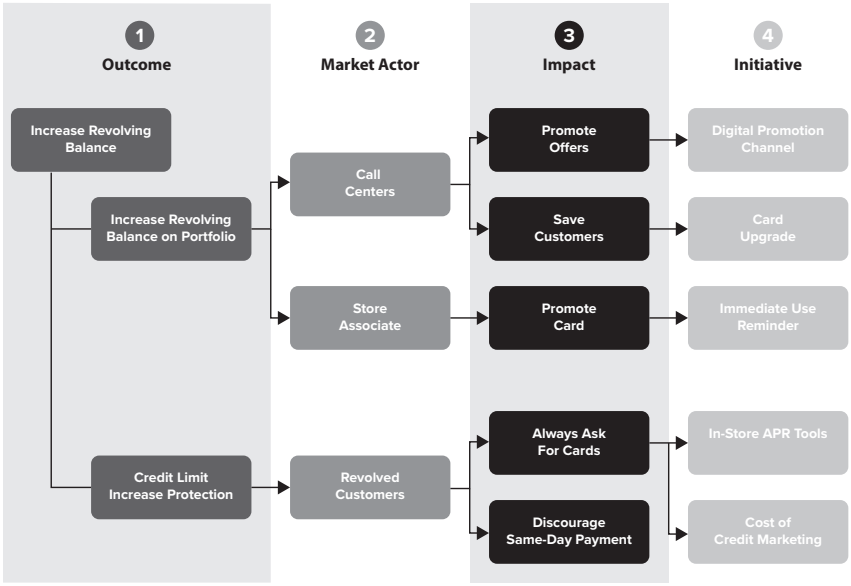


Impact mapping allows us to look at the relationship between a large collection of work, outcomes, and market actors, providing

Projects, Provoking Thoughts, 2012, https://www.amazon.com/Impact-Mapping-software-products-projects-ebook/dp/B009KWDKVA.

20. https://gojko.net.

information in a way that makes it easier to make decisions about structure and about how best to organize around the market outcomes you want to achieve.



DEFINING YOUR TEAM MODEL IN FOUR EASY STEPS

We used impact mapping within a four-step process:

1. Domain clustering the medium-term (three to six months) goals for organization; identifying common themes and synergy.
2. Impact mapping each domain in order to get a better understanding of initiatives and the behaviors that we are looking to change in our stakeholders.
3. Drawing boundaries on the impact map to define team mapping of specific impacts; analyzing the skills required to deliver potential initiatives.
4. Clustering related initiatives, unexplored domains, and operational processes; creating team definitions.

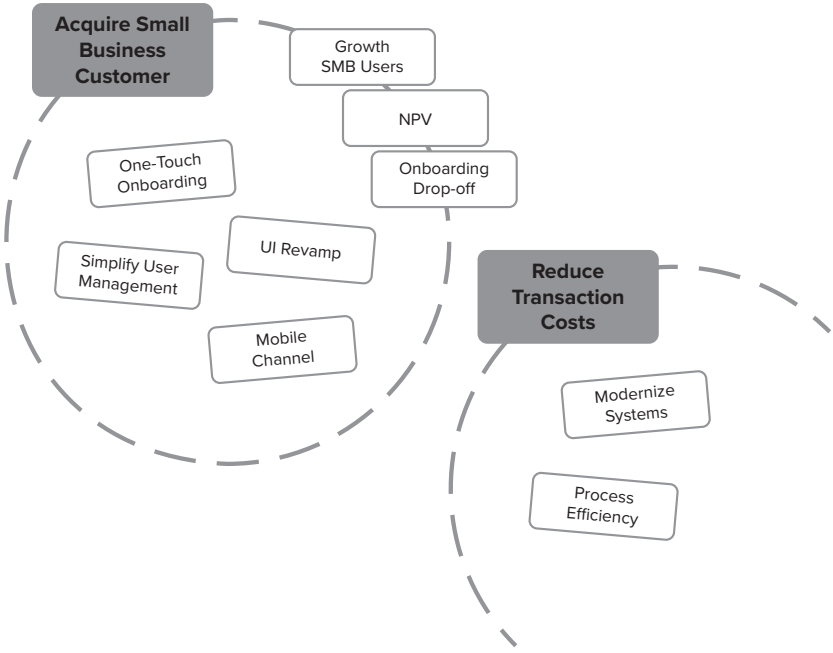
Tyler and I started with a highly collaborative barn raising–style session, where we had an initial kickoff with leaders in the organization,

followed by weekly sessions where we refined our design. We then ran several all-hands sessions to provide everyone with a chance to iterate on what was put together by the leaders, and then proceeded to stand-up team-level backlogs and Kanban systems.

The power in taking this approach is that we were able to pull people from what they were used to in terms of existing organizational structure. When you really want to initiate change, it can be useful to start with tools that support blue-sky thinking, like impact mapping.

Step 1: Domain Context Clustering

Before we started our impact map, we wanted to get some initial alignment with the various teams on how important business concepts could be grouped together. We wanted to group outcomes and outputs according to how similar the involved work was. It was a quick and dirty way to initially identify various context boundaries of work, boundaries that could act as candidates to become teams, ecosystems, or larger organizational constructs.



Some examples of the domain clusters we came up with included:

- Customer acquisition
- Store associate advocacy
- Operations cost reduction
- Credit risk
- Digital engagement
- Service improvement

Each domain cluster contained a collection of outcomes or work that was most closely associated with the cluster. For instance, the customer acquisition cluster had the following outcomes:

- Simplify and enhance value proposition
- Simplify experience
- Acquire fewer (but better) customers

We also annotated each domain cluster with some of the key ways

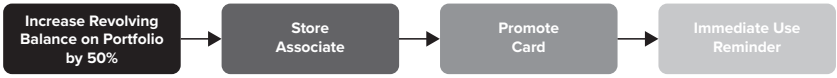
to measure the impact of working on a domain. For example, the customer acquisition cluster had the following metrics:

- One-and-done rate
- NPV
- Application volume

Domain clustering is a great way to get a look at the larger universe. It was as low-fidelity as possible, to allow participants to swarm on a number of context boundaries, with an eye to performing more-detailed impact mapping. We found that we could work with a fairly large group (about thirty-plus people) to imagine all the possible things that could fit in the overall universe of the business they were in. The result was a rich set of subjects and master subjects that we could start to associate with potential teams.

Step 2: Impact Mapping

OK, now we are ready to do some impact mapping!



To recap, each cluster in the domain was impact mapped to identify clear, measurable goals for the business to achieve. Market actors were identified who could help achieve these goals.

In the case of our client, the team took a divide-and-conquer approach, focusing on a collection of two or three domain clusters each and refining the domain concepts using impact mapping. The group then brainstormed the various personas and behaviors that would help that team achieve a particular goal or set of goals in that cluster. The deliverables were then estimated at a very high level and placed into an overall backlog.

Defining outcomes for the organization and mapping them to market actors is a good first step. Decompose these goals into finer-grained

subgoals until you have a set of tangible outcomes that you feel could be tackled by an ecosystem and/or a team. Knowing how far you need to decompose your goals will be more of a gut feeling, at least at first.

It's often good to get started by brainstorming all the external participants that can help achieve our organizational goals. Some examples include customers, buyers, partners, external marketing agencies, regulators, and so on. At a minimum, you want to post every market participant you can think of and start clustering them based on their needs and wants.

As stated previously, it is often helpful to map our impacts to discrete features/deliverables. As part of this exercise we can also refine these work products into the beginnings of an Agile-style backlog. In effect, you can start identifying some potential epics, features, minimal viable products (the smallest increment of value we can deliver to our customer that can increase market learning, known as MVPs), and so on that teams could deliver. Depending on how well your participants know your organizational goals and organizational demand, going a level deeper may facilitate a better understanding of the work for future ecosystems and their teams.

Oftentimes, it will make sense to go beyond the impact map. An impact map is a great and succinct way to look at your universe, but that means it will lack detail. You may need to elaborate using another artifact/facilitation type if a lack of detail is causing churn.

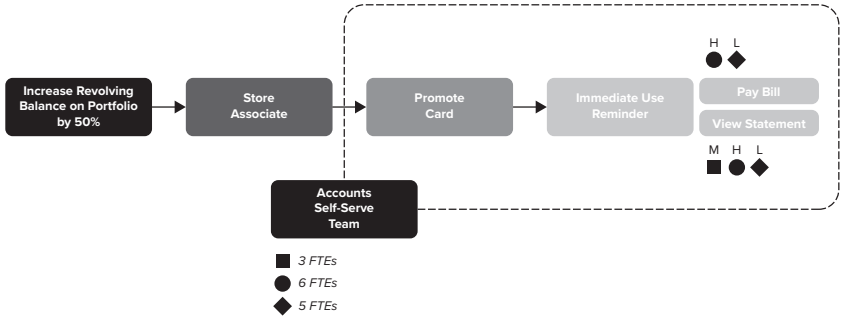
If you are facilitating an organizational design session and it appears that organizational priorities are unclear or conflicting, or there is a lack of proper understanding of the work, then take a step back and start building a backlog that contains discrete increments of value that you can start allocating to various ecosystems and teams. I call this practice *Agile long-term planning*, a practice I'll go into much more detail in in chapter 9, "Reteaming in an Agile Ecosystem."

Step 3: Team Mapping

The next step is to take a look at the impact maps and start to define team boundaries. The idea here is to allocate a certain portion of the overall impact map to a potential team. The boundary could be drawn

around any level of the map, from higher-level goals to subgoals, behaviors, or deliverables.

As Tyler and I were thinking about both team and ecosystem boundaries at CCC, we analyzed the work for the kinds of skill sets and capabilities required to realize that area of the impact map. This was an iterative process, with boundaries being drawn and redrawn based on an increasingly better understanding of what skill sets could be combined into an effective and reasonably long-lived team that could achieve organizational goals.



Taking a page out of *Domain-Driven Designs*'[21] bounded context map, we specified whether the team would be best served if that capability was provided by a dedicated team member or whether a service-provider relationship would suffice. For example, work could be done by sending it from one team to another.

Team mapping is simply the act of mapping the skills and efforts required for your deliverables, impacts, or outcomes. Many will point out that mapping capabilities to outcomes is nothing new; organizational designers have been doing this for decades. And they would be right. What makes this exercise different is the low-fidelity approach, the co-creative sessions, the shared artifacts that a large number of people are able to engage with. Traditional capability mapping exercises are often a slog, run by your favorite external consultancy, with way too much detail, and often used to build a hierarchical organization of specialized departments. In the case of team impact mapping, we are focused on how to bring people together to accomplish

21. http://shop.oreilly.com/product/0636920033158.do.

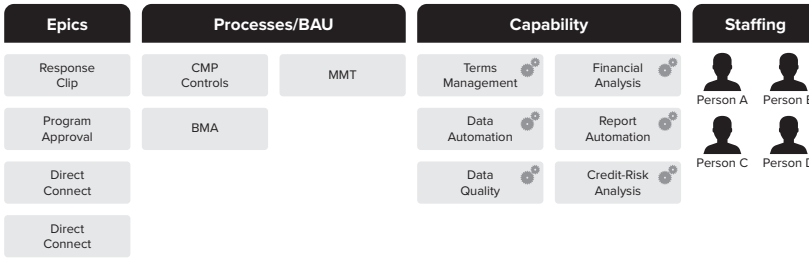
organizational outcomes—and on doing it quickly.

As we are designing teams, we may want to get a better understanding of the work that those teams will do. This will allow us to make more-informed decisions when trying to create context boundaries around teams and higher-level missions.

As pointed out earlier, Kanban is an excellent method that not only enables teams to operationalize their value creation activities with agility but is also a great mechanism to further refine your organizing structure. With Kanban, our effort to organize can be informed by cataloging demand into customer-oriented services. Each of these services can then be mapped to one or more discrete work types (capabilities), which in turn is defined by an explicit and visual flow of work. If you find a lack of clarity around how certain types of work are to be accomplished and by whom, then using Kanban to model, and even test out, a flow of work can be a good way to move forward. We can visualize the suggested flow of work and spot bottlenecks and other impediments. This provides a rich set of observations that empower teams to continually experiment with changes that further refine how we are organizing.

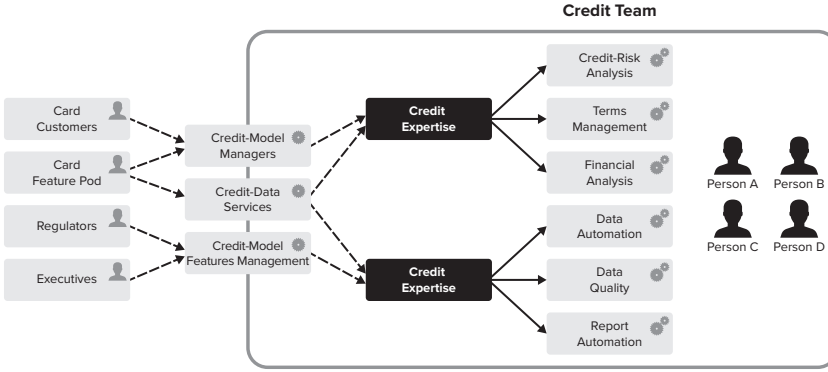
Step 4: Team Definition

The final step is to run a set of sessions focused on making the abstract team model a little more real. At CCC, we took each team identified in the impact map and started slotting any current work or upcoming work to it. We categorized this work into large "change the business"–style epics, process work, and "business as usual"–type activities. We then refined the capability-to-work mapping already done, and finally assigned real people into the various teams. This, of course, was also an iterative process that caused us to challenge some of our assumptions about mapping impacts to teams in the first place.



We added the metrics from our impact map onto the top of each team's definition. At this point, leaders were ready to "officially" socialize the various team structures across the organization. Of course, unofficial socialization had been happening throughout the process, given that the models were created and displayed in a highly accessible place and we welcomed anyone to come by and contribute at any time.

Team mapping provides a quick understanding of the relationship between demand and capabilities. We get agreement on who the customers are for each team, whether those customers be another team or an external customer. We also start getting an idea as to the reusable services that could be provided to those customers.



At this point the organization was more than eager to stand up teams and started adopting some Agile concepts. The group started by inviting all employees to a kickoff meeting where everyone was invited to update impact maps, redefine team structure, and challenge

outcomes. An all-day affair, the session provided an opportunity for everyone to give their input on the work that had been done to date.

Tyler and I got to work providing basic Agile training to any teams that wanted it, helping teams create backlogs, design Kanban systems at the team and ecosystem levels, and put into place both ecosystem- and team-level event cadences.

When facilitating this type of exercise, it's generally better to start from the outside and move inward. If you don't know your organization's outcomes, get those down. If market actors are unclear, define those next. Then move inward, establishing some context boundaries based on domains of knowledge. It's often better to start defining structure around outcomes and impacts that are market facing, and afterward move on to defining teams or ecosystems that exist primarily to fulfill support functions. We will go further into demarcating market-facing and support structure in the chapter on the new enterprise.

IN SUMMARY

- Agile works because of the social density that comes from a transparent team environment.
- You scale Agile by scaling social density, by applying the team and teaming concepts to larger contexts, not through extra methodology and extra controls.
- An Agile ecosystem is made up of a small number of market-facing teams along with dedicated support allocated to them, consisting of between twelve and fifty people.
- Typically, an ecosystem is dedicated to serving a subset of organizational stakeholders and market actors.
- We can define teams and ecosystems using team impact mapping: by mapping your users and their behavior to your organizational goals and adding the deliverables that support these behaviors, we can determine the capabilities required and group them into teams and larger structures.

7

Operating an Ecosystem of Agile Teams

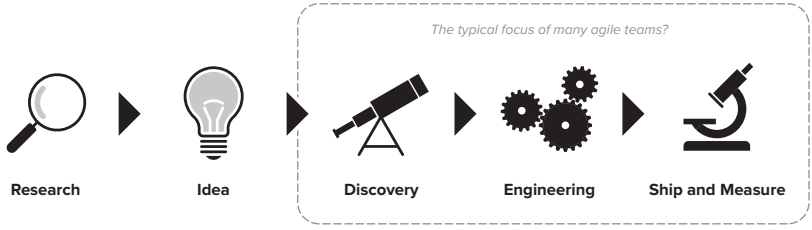
If we want to stay organized, we will likely need to put focus on the day-to-day operations of our ecosystem. How does work make its way to the teams in the ecosystem? How do teams align with each other when working through shared assets or other dependencies? How do we escalate issues and otherwise communicate to organizational stakeholders outside of the ecosystem? How do we gather and react to market feedback? How do we introduce changes to the way we are structured, including team composition and other changes to our management system?

Once we have defined an ecosystem and started to populate it with teams of people, we want to focus on putting a management system in place for our ecosystem. We will want to make sure the management system is both lightweight and fit for purpose. Getting this right is something many organizations struggle with. Many rely on a kind of Agile-waterfall hybrid, where development delivery teams use Agile to create software. But strategy, project formation, and market releases use traditional big-batch methods. Others turn to heavyweight off-the-shelf or bespoke Agile "scaling" frameworks. These typically take what are often good/great practices and clump them together in a way that does not suit the variety of contexts in which they are being

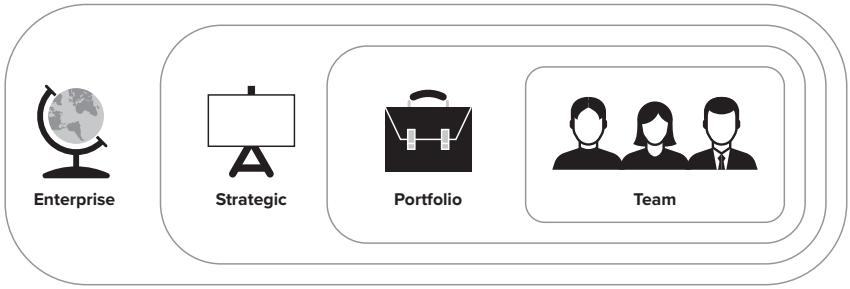
deployed. Finally, some try to manage the teams' perspective the way they always have, through tenacity, brute force, and sheer will.

Really, what we want to do is find a way to scale our Agile concepts with simplicity and grace. First, let's provide a description of what we mean by *scaling*, and how Agile ecosystems allow us to think about scaling.

When we connect teams into a common ecosystem, it becomes both easier and more necessary to scale our agility *horizontally*. When we scale horizontally, we connect concerns typically considered upstream or downstream from a core Agile team. We go beyond thinking about how to scale development and delivery to asking members of the ecosystem to operate as a larger part of the value stream with agility. We want to bring rapid feedback and self-organization to the entire value stream, from concept to customer feedback. Ecosystems are more likely to have concrete missions and measurable objectives than a single team. They are more likely to have accountability to concrete business outcomes.



Of course, we are also scaling Agile *vertically*, simply by the act of grouping Agile teams into a cohesive ecosystem. Scaling Agile horizontally is about coordinating work within larger and larger organizational boundaries. It is also about being able to manage larger and larger missions, or outcomes. When scaling horizontally, we care about both the strategic perspective and the tactics of managing interdependencies, common platforms, and enterprise concerns that come with them.



SCALING THROUGH THE PRACTICES WE ALREADY KNOW AND LOVE

A secret that many don't see at first is that Agile practices can and do scale; you just need to take the time to understand why they work and adapt them to the context of the larger scale.

Some of the simple behaviors we see in an Agile team that can and do scale include:

- Enhancing collaboration within teams using **visual story/task management** and **definitions of done**
- Synchronizing people using **sprints and ceremonies**
- Conducting team planning with **product and sprint backlogs**

We can scale Agile to operate an ecosystem of teams by abstracting these concepts into a collection of behaviors that can be applied at the ecosystem scale.

Agile teams make good use of team-level visual management, oriented toward story development and underlying tasks. At the ecosystem level, we can make better use of *visual flow management of end-to-end value creation*, making it clear how work flows across the ecosystem, both within and across teams. Illustrating impediments and bottlenecks makes it easier for people to understand when to team across teams and when to reteam.

We can also synchronize using *fit-for-purpose events* held at a steady cadence. Agile teams make good use of team-level cadences and

team-level feedback. Cadences aimed at multiple levels of feedback provide a means for members of the teams within an ecosystem to reteam in light of the latest learning.

Agile teams make good use of team backlogs and team planning. When an ecosystem uses *long-term planning and graduated and hierarchical backlogs*, it is easier for teams within the ecosystem and their stakeholders to reteam around the next set of overarching outcomes.

Let's go through some of these practices with an eye toward facilitating the operations of an ecosystem of teams.

LONG-TERM PLANNING

Long-term planning is the act of applying team-level sprint planning concepts to a larger scale. By planning at a larger scale while using visual and accessible concepts such as backlogs that are populated by increments of value, we can make it easier for all members of an ecosystem to understand what is coming toward them in the medium and even longer term.

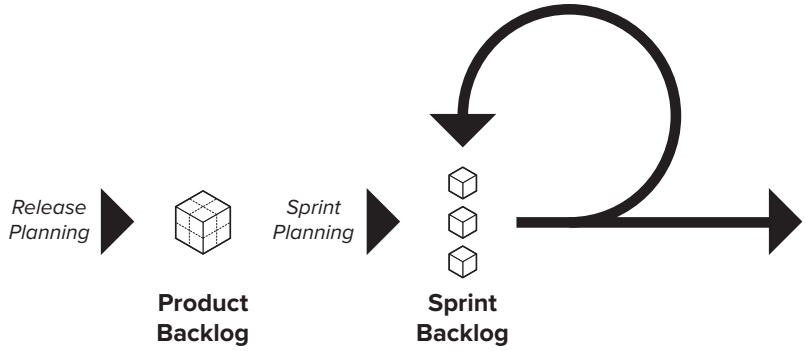
Isn't a long-term view by definition not Agile? Yes, a long-term view will never stay the same over time because, you know, change. But that is why we want to use Agile artifacts to represent the possible future. These artifacts are easy to build, easier to change, and much easier to tear down when they are wrong. The goal here is to take the long-term thinking that is often only described in heavy road maps, detailed business cases, and program charters and start representing it in a way that can serve as an information radiator to larger audiences, like the entire population of team workers in an ecosystem.

Remember, if the plan is really wrong, rip it up! I have done so many times. In fact, back in the day when we used stickies and tape for our artifacts, ripping up artifacts that have outlived their usefulness was a cathartic experience!

OK, back to scaling Agile team planning. When we think of team-level planning, we often think of teams that:

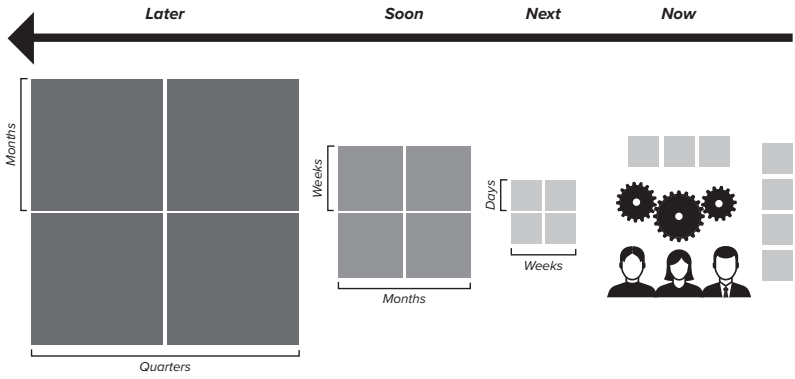
- Conduct release planning, where epics are placed into a product backlog

- Conduct sprint planning, where epics are decomposed into stories and placed onto a sprint backlog based on team velocity



Applying planning at a larger scale includes expanding our concept of a backlog to what I call a *graduated backlog*. A graduated backlog encompasses both a larger *time horizon* and coarser-grained *value increments*.

These horizons have orders of magnitude that are less precise the further away the work is from being *started*. When we don't expect to start work for a long period of time, we expect it to be poorly understood. As the team completes current work, items in the backlog are shifted to the right as the start date of the work becomes closer.



Teams can anticipate the reasonable start date of items in the backlog by estimating their *throughput*. Agile teams deliver value by breaking larger requests into smaller increments often known as *stories*. The more teams deliver, the better they can anticipate their average story throughput per month or per week. This number can be used to place increments of value in the graduated backlog and to position the work according to its likely starting week, month, or quarter, taking into account that the further out we go from today, the less precisely we want to define our start period.

As teams become more accustomed to delivering value using Agile methods, they often start to track how many stories they can deliver in a small period of time, typically a couple of weeks or a month in duration. Many Agilists love to call this period of time a *sprint*.

Once teams understand their story throughput (also known as sprint velocity), it becomes much easier to make informed decisions about when the next increment of value will make its way to their existing missions and teams. This gives them a signal when it might be time to revisit their organizational design.

But don't we need to estimate each story? Well, you can. Sometimes estimating each fine-grained unit of value brings about insightful dialogue, especially for new teams working on new systems.

But in many cases it can feel like needless overhead, an artificial act of illusionary precision. Turns out the biggest value in fine-grained estimates is to help teams ensure that *stories are small*.

If we can mentor teams to break down work so stories are small, *they will have a small amount of inherent variation*. So if teams are watching for items in the backlog that are not small, break them up so they are small. We don't need to waste time estimating if we don't want to.

ESTIMATE AND TRACK THROUGHPUT

Once our stories are small, we can make the *exact same observations* provided through story estimating by simply estimating and then counting the number of stories that flow through our system of work.

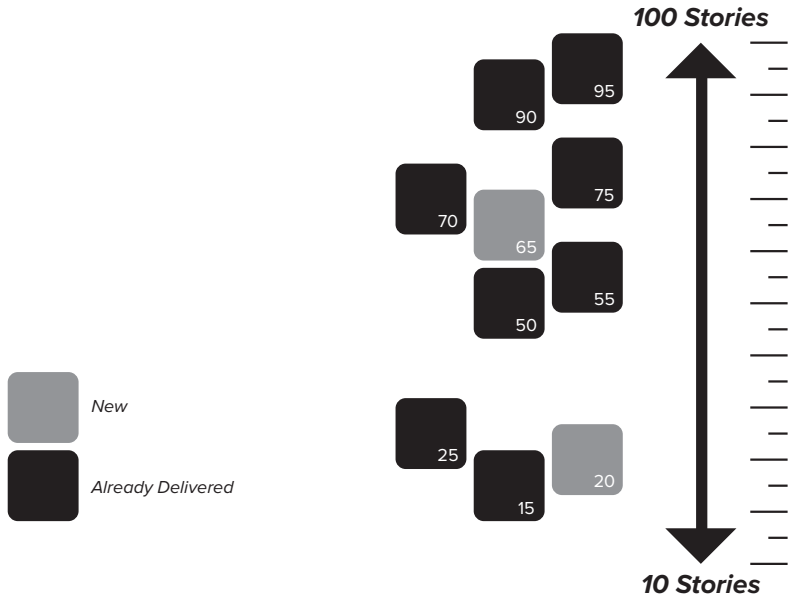
An emphasis on throughput can save a lot of time and effort from the team; they no longer have to debate small differences in complexity between each story, differences that do not matter if we look at the medium or long term.

Once we can use a team's story throughput, we can then estimate items in the backlog in terms of the *number of stories* in it. This is where estimating using relative sizing can really shine—not for stories, but for larger-scope items, which will by their nature have a high degree of variation. We can use relative sizing to quickly estimate new work by asking, How big is this piece of demand compared to something that was previously delivered? Is it bigger or smaller? By a lot or a little? Using this approach to estimating story counts for larger items becomes easier over time.

The first step to creating a backlog, one that provides a better understanding of value and impact, is to stack-rank backlog items against each other—or, better yet, against other units of value that your organization has previously delivered. This allows you to quickly estimate an increment of value relative to something else. It then becomes easy to assign an estimate to backlog items based on where they are in the

stack.

We can use this relative stack-ranking approach to estimate both the size/effort and the value/impact to the market we think it will have. This makes it easier to start prioritizing our demand, putting urgency on work that has higher relative value and lower relative complexity. Note this is not the end of estimating, but a quick way to do just enough discovery to define some reasonable mission and team structure.



I like to make relative size estimates visually by placing the work on a vertical axis and encouraging the team to place previous work initiatives onto the axis based on a complexity scale. Estimating is simply an act of placing new items in the right spot according to relative complexity.

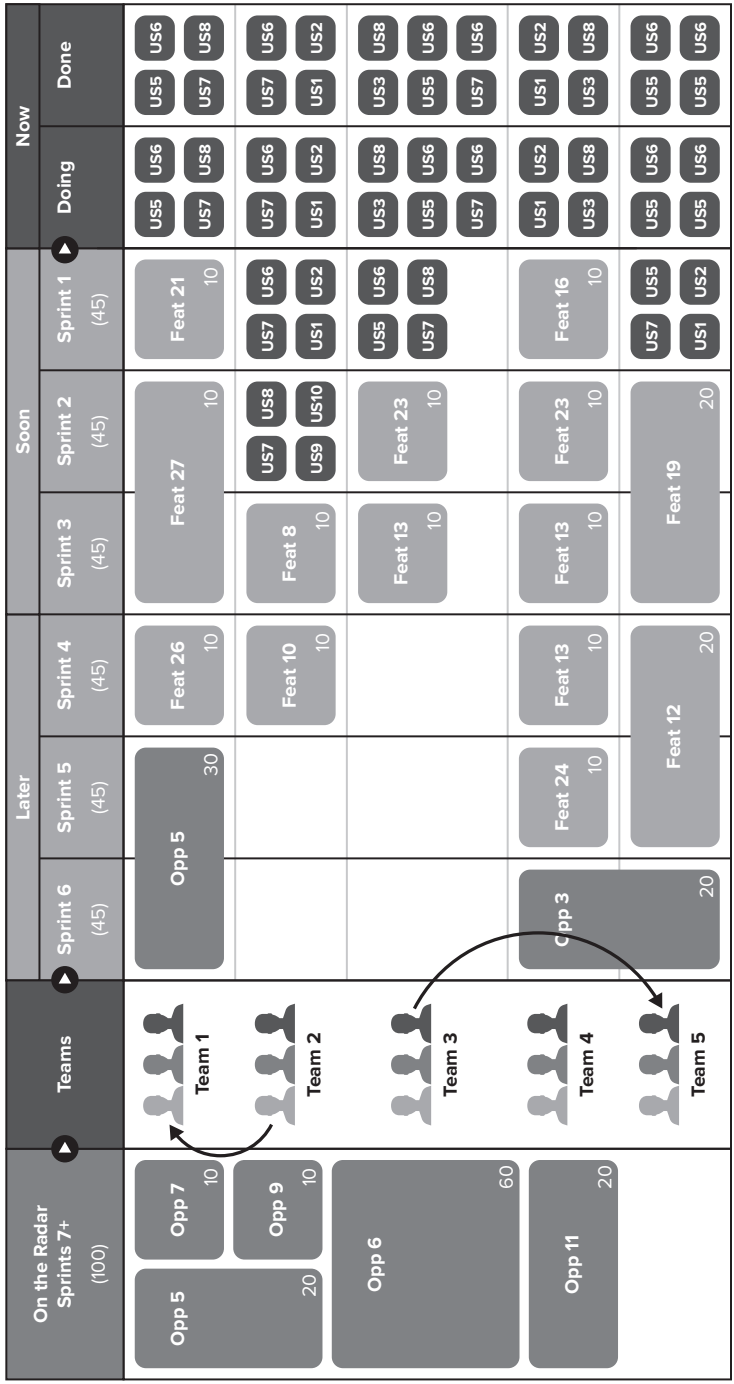
GETTING BETTER BACKLOG CLARITY THROUGH STORY MAPPING

If the goals are still not clear enough for you to guide the organizational design dialogue, consider refining one or more items on your backlog through a story mapping session.

Story maps allow you to map the flow of your solution in a hierarchical way. Story maps also allow you to define independently releasable thin slices of value, or MVPs. This helps you prioritize small units of value that you can release to your customer. If you are churning on trying to understand the what, the why, or the how of an increment of value, I highly recommend you try out story mapping.

Initially, estimating is just swag informed by experience! Where there are too many unknowns, teams can spend more time exploring the work using techniques like story mapping, design modeling, or even actual delivery. This act of discovery is from the vantage point of an explorer; go wide for the most part, and go deep only to uncover assumptions and unknowns. As the delivery start date approaches for the work, deeper dives can be used to get a more accurate understanding.

With a little bit of visualization, we can connect our backlog to all the teams in an Agile ecosystem. This allows us to see which teams are working on which outcomes, which allows us to anticipate problems in our team composition. We can clearly see if one team is overloaded or if there is less demand for another team. In this way long-term planning allows team members to understand which outcomes will be worked on and allows them to *move between teams based on that understanding.*


On the Radar Sprints 7+ (100)
Teams
Later
Sprint 6 (45)
Sprint 5 (45)
Sprint 4 (45)
Soon
Sprint 3 (45)
Sprint 2 (45)
Sprint 1 (45)
Now
Doing
Done
Opp 5 20
Opp 7 10
Opp 9 10
Opp 6 60
Opp 11 20
Team 1
Team 2
Team 3
Team 4
Team 5
Opp 5 30
Opp 3 20
Feat 26 10
Feat 10 10
Feat 24 10
Feat 13 10
Feat 12 20
Feat 27 10
Feat 8 10
Feat 13 10
Feat 13 10
Feat 19 20
Feat 23 10
Feat 23 10
Feat 21 10
Feat 16 10
US7 US8 US9 US10
US5 US6 US7 US8
US1 US2 US3 US5 US6 US7 US8

It is important that all members of the ecosystem have access to this view and can spend some time looking at it holistically and asking the questions:

- Do any teams need my help?
- How can I more effectively team with other teams?
- Should any of the teams consider reteaming?
- Should our team be working on a different outcome?

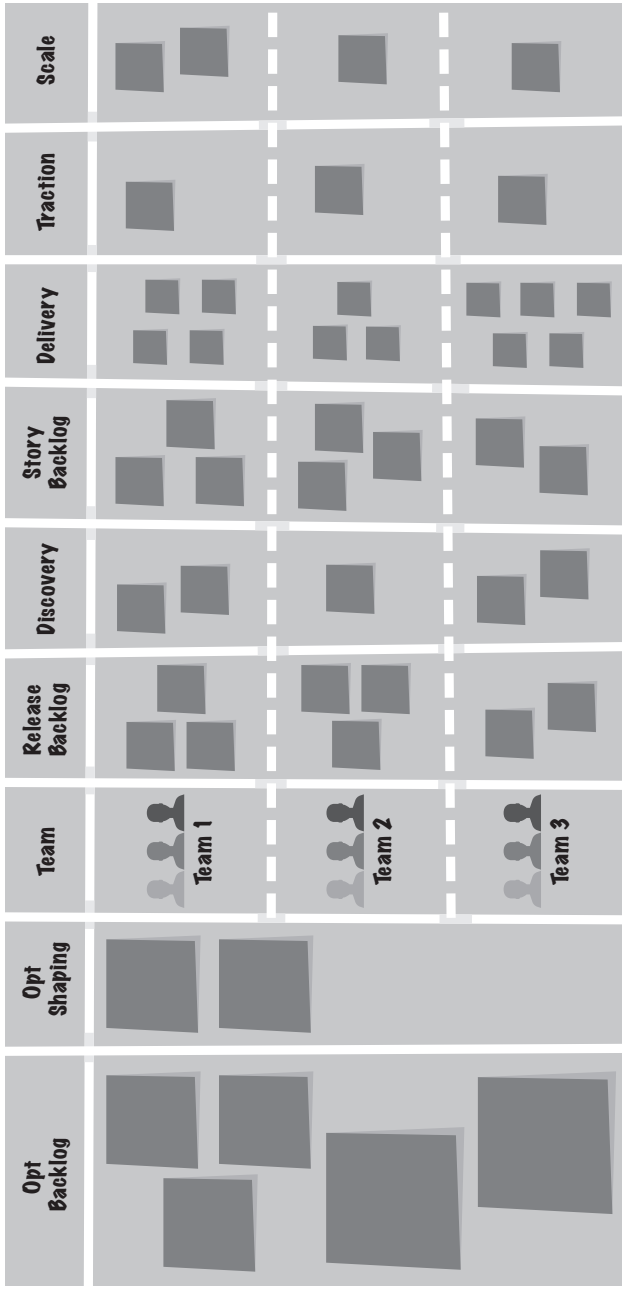
This snapshot was taken from a program-level backlog used to facilitate a large-scale regulatory program that consisted of more than eleven teams. At the beginning of the program, a high-level story map was used to understand what product and system experts were needed from across more than seven different organizational departments. Teams were quickly assembled by color-coding each system or product expertise required for the program, matching people to the colors, and then assigning that color to epics in the program-level backlog. Here we can see the people belonging to two different teams represented on the Kanban, right next to the epic that team is working on.

This planning exercise took little more than a couple of hours. While it is true that we spent weeks ironing out the kinks, it was an incredibly fast process!

During this program, the team composition and the number of teams flexed based on the work flowing through the system. Some team members were more or less fixed to a team, while others acted more as travelers and would shift across teams frequently, in some cases daily. The process of people moving across teams started as a manager-led process. But over time it became the responsibility of everyone working on the program. Planning, stand-up, and review.

ECOSYSTEM-LEVEL KANBAN

As the complexity of our ecosystem increases, we can use Kanban to enrich our team members' understanding of the work and scale team-level visual management to the entire ecosystem. To do so, we abstract the work each team is doing to a higher level, each ticket representing an outcome, or perhaps a feature or thin slice. Even though this level of abstraction is less accurate, it is often enough detail to facilitate understanding of where our organizing structures are failing us, and where failures in collaboration are impeding value creation.



Kanban scales across multiple teams

Much like when we pay attention to an ecosystem backlog, when team members and people in team support functions attend an ecosystem-level Kanban it becomes clear when it makes sense to engage in teamwork across the official organizational structure, as well as when to reteam, either making a small tweak or a complete reshuffling of people.

With the right attention to an ecosystem-level Kanban system, people can start to form muscle memory in terms of how to collaborate in different configurations, which helps with teaming across teams. People start to transition their thinking from static structure (team or otherwise) to the set of organic interactions that promote flow. This is a high-maturity state; in this form all teams are created through teaming. It can take time to see this type of behavior take shape. We will talk more about this more dynamic form of teaming in chapter 9, "Reteaming in an Agile Ecosystem."

USE VISUALIZATION TO ILLUSTRATE CAPABILITIES AND EXPERTISE IN MORE DETAIL

Simple flags and other annotations can be used to articulate which capabilities are needed to complete a piece of work, as well as which capabilities a team within an ecosystem is in possession of.

There are all kinds of creative ways people can choose to illustrate the capabilities required to complete a piece of work, in addition to the capabilities present in the current structure of teams. I have seen people use clever annotations, as well as team/individual "hockey cards" with simple icons to indicate various skill sets.


Opt Backlog
Opt Shaping
Team
Release Backlog
Discovery
Story Backlog
Delivery
Traction
Scale
Team 1
Team 2
Team 3
Enrich Onboarding with AML Capture
7 Legacy Sys. Stories
5 Service Stories
5 Ops Stories
UI
Service
Database
Legacy
Infrastructure
IT Ops

DEFINING ECOSYSTEM-LEVEL EVENTS

On an Agile team, events are an important part of maintaining an operational cadence. It is one way teams can maintain a sense of predictability in a sea of change. The work may change, the process may change, our customers may change, the solution may change, but our cadences remain steady. We use these steady events to give us a sense of consistency. Events also help teams maintain a sense of being as a team. They act as a forcing function that gets individuals out of their individual work and oriented toward the outcomes of the team. Examples of team events include planning, team stand-ups, reviews, customer demos, and team retrospectives. Events are scheduled either according to a set cadence—for instance, we may run a planning session once every two weeks—or according to certain WIP limits—for instance, we may run planning whenever we have three or more features that are in the "analysis done" state of our Kanban.

While teams are free to choose the operational cadences they want to use, the list of team cadences is relatively standardized, as are the attendees. Most teams will have events that include planning, reviews, retrospectives, and stand-ups, with some teams combining a few of these together. In almost all cases, the entire team will attend these sessions, with some events including participation with key stakeholders. Again, there are cases where teams have departed from the norm. Teams that appreciate the consequences of not attending to flow, and that understand how to effectively self-organize, can and will move beyond the need to strictly follow these events, and they should be encouraged to do so.

Using events to set operational cadences is something ecosystem members will want to do. Just as we want individuals to look at the perspective of the entire team on occasion, we also want team members to think about the ecosystem they are in. In an enterprise no one is an island. Ecosystem-level events provide a space to look at the larger picture and to consider opportunities to team across team boundaries.

In contrast to team-level events, I have found that an ecosystem's use of events will more heavily depend on context. When an ecosystem is being used to run a highly integrated program, it may benefit from mirroring the complete set of team-level events, even if they are

run less frequently. An ecosystem that is delivering various outcomes for a particular business area, with each team having a different set of stakeholders, may find they only need a weekly stand-up to align on overarching impediments. Attendees will also vary heavily depending on context. I have never seen a "team-of-teams" system adopt the exact same set of operational cadences. Despite what any scaling framework will tell you, you need to manage your ecosystem in a way that makes sense for your context.

With that in mind, it is better to share a set of *ecosystem event patterns*. The intent is to inspire you to come up with an event model that suits the needs of your ecosystem. This means that members of an ecosystem will need to determine the exact set of events that makes sense for them; there is no cookie-cutter list to borrow from. The exact set of attendees will need to be determined on a case-by-case basis.

ECOSYSTEM SYNCHRONIZATION EVENTS

An *ecosystem synchronization event* is where all teams in an ecosystem synchronize work by replicating one or more team-level events. Teams will nominate a team representative to attend each event. Ecosystem-level planning, stand-ups, and/or reviews are attended by delegates from each team.

Synchronization events make sense when teams in the ecosystem are dedicated to a program or to delivering value with numerous dependencies across teams. There may be a need to quickly escalate impediments and risks that impact the portfolio.

As an example, we worked with a client to visualize an ecosystem's workflow and design a set of cadences to allow for representatives from each team to escalate impediments, discuss dependencies, and align on their intake items. This proved to be a foundational process for directors and leadership to get a holistic view of the whole ecosystem on a biweekly basis.

SHARED STAKEHOLDER REPLENISHMENT EVENT

An ecosystem may need to service multiple stakeholders, who will not always agree on what the priority of work is. Rather than abstract the job of prioritization into the work of a single product owner, ecosystem members may decide to make the process of prioritization explicit by inviting stakeholders to attend a replenishment meeting held at a stable cadence. The goal of the event is to determine which work gets prioritized for the ecosystem, and then to place it into the immediate backlogs of each team within the ecosystem.

As an example, one of the ecosystems within the payments group at Scotiabank was made up of three or four application teams focused on high-value payments. The teams served the needs of almost every business line in the bank, due to the fact that wire payments are the de facto method for settling the bank's financial obligations. This included providing payment infrastructure for internal processes like employee payroll, payments for vendors and suppliers, and cross-bank settlements.

The wire payments processing engine was based on an ancient technology (Atlantis hadn't sunk when this system was first developed), and thanks to the incorporation of a constant stream of regulatory changes, it had accumulated an immense amount of technical complexity over time. Working on the system required an incredibly steep learning curve to be productive.

Given the diversity of requesters and their interests, all of whom viewed their requests as "highest priority," the idea of relative value was a nonstarter as each business line jockeyed for priority. The list of unfinished requests grew longer and longer as a result.

The teams in the high-value payments ecosystem remediated the situation by socializing a capacity-based model, where the ecosystem would only accept a fixed number of stories every couple of weeks based on their historical story throughput. Requesters were asked to pitch their highest-priority item to a committee of stakeholders representing each of the lines of business of the bank. Stakeholders would have the opportunity to ask questions of the requesters. Items were presented and then voted on by the committee in batches. Each batch was then prioritized by its number of votes and accepted into the

delivery backlog on a first-come, first-served basis until the backlog filled up to reach two weeks' worth of story capacity.

This approach forced requesters to simplify their value proposition and cut down on politicking and empire building. If an item was passed over, requesters had the opportunity to improve their pitch and submit it during the next session. This forced purging of requests, and removing them from the backlog helped requesters make the choice to either sharpen their value proposition or abandon their request.

This model has proven to be extraordinarily popular with the teams and their stakeholders. Despite the subsequent turnover of most of the stakeholders, the value- and capacity-driven approach is continuing to prove the simplest and most effective way to ensure meaningful prioritization for this critical enterprise capability.

LONG-TERM PLANNING EVENT

Agile teams delivering work with relatively stable throughput metrics are able to leverage relative sizing and throughput calculations to perform long-term planning. Putting a cadence around an event dedicated to aligning on this type of planning provides a place for cross-team coordination, to uncover any assumptions being made about another team or their work.

Scotiabank's payments teams eventually grew to cover more than five ecosystems and thirty-some teams. They relied on this technique as a way to have quarterly planning sessions with their stakeholders. Each ecosystem would do an estimate using relative sizing of all the epics and features in the backlog, determine how many stories the ecosystem could deliver in a quarter using throughput, and use quarterly buckets to prioritize and plan upcoming quarterly scope. This was a simple yet powerful way to align on priority and expected delivery timelines of a certain scope of work.

TEAM-TO-TEAM SYNCHRONIZATION EVENTS

In some cases two teams within an ecosystem may require closer

alignment across their entire body of work. The teams may decide to connect to each other by agreeing to share some or all of their team-level events. This helps the two teams synchronize work across their teams. Like regular team events, these synchronization events can follow a repeatable cadence or be triggered through a pull system (i.e., when a certain amount of work hits a certain state in a Kanban system).

One of our client teams over at ATB introduced a vendor squad to help increase throughput on what was proving to be a very large scope of work. Introducing an extra team, especially one external to an organization, always comes with its own challenges. Different approaches, social differences, and behavioral norms often get in the way. The teams elected to hold shared stand-ups every other day. This helped when teams needed to brainstorm for design solutioning and specific hiccups.

Sometimes teams need to be a little closer than normal, but not completely joined at the hip. Maybe two or more teams have dependencies on each other. Maybe all teams want to be aware of each other's impediments in order to spot shared problems. A nice lightweight way to keep teams informed of each other is to have one or two team members attend another team's stand-up session. Spread knowledge by rotating who attends.

One of the squads at ATB required the help from a legacy team to extend existing APIs that would enable the team to optimize the way they would retrieve their data. Changes would equally affect the front end and legacy system code. The squads agreed to elect members that would attend each other's stand-ups. For a period of time, the two teams had the feeling of almost being one team. This became an effective and succinct way for teams to ask each other questions and share updates.

COLLABORATION TIME (OPEN SPACE)

An ecosystem, or any larger social construct, needs space to work as a larger group on larger problems. Collaboration time is another solution I came across while helping CCC in their adoption of pods. The idea is that once a month all the members of an ecosystem would gather for

an open-ended facilitation of problems and solutions. Topics ranged from market opportunities to key risk and improvements, from career growth to learning, and so on. Topics to discuss were presented, voted on by participants, scheduled, and then facilitated in separate breakout sessions.

This approach allowed groups of interrelated teams to benefit from high collaboration in a way that is hard to predict. It allowed for people to cross their team boundaries and to innovate on higher-level mission objectives.

REPRESENTATION IN ECOSYSTEM CADENCES

Who shows up to represent the teams at ecosystem events varies depending on where the organization is in its journey toward more effective self-organization. We want to maximize decision power to teams and the individual team members; at the same time we need to concern ourselves with the churn introduced when we have too many people present at once.

The worst approach by far is when "team representatives" is shorthand for "positional leaders"—in other words, managers. When positional leads are the only group that meets to discuss team decisions, the ecosystem will suffer from all the problems we discussed that come with hierarchical management.

A nice evolution is to have different team members attend on a rotating basis, perhaps paired with positional leaders, perhaps not. If positional leaders retain a more permanent spot, they need to exercise care to make it easy for the rotating team member to contribute—perhaps by asking them to play the role of facilitator or supporting them to be the one that answers questions, brings up issues, and so on.

The most inclusive answer is to include everyone in the ecosystem in these kinds of events. Of course, this can be unfeasible when an ecosystem is made up of fifty-plus people. As more and more organizations move to a remote-first workforce, this becomes a bit easier, but strong facilitation is still required.

For more strategic events, one way to include everyone is through the open-space format. Open spaces provide a lightweight means for a

large group of people to co-create solutions. Other ways include holding ecosystem-level events based on different perspectives.

One perspective could be discipline based, for instance all UX professionals across teams in the ecosystem could meet to discuss ecosystem-wide concerns. A perspective could also be system specific, or customer-segment specific, or based on an enterprise concern like DevOps or security.

The most effective use of ecosystem-level Kanban I have witnessed has been when groups of people organically collaborate around the flow of work to make more strategic decisions, updating the board if needed and often leaving a marker to indicate that a change was made. Once a week, people gather to hear from everyone who left a marker on the board. The board was then cleared of all markers.

IN SUMMARY

- Scaling Agile by defining a more complex or comprehensive methodology will only result in reduced agility. Instead, scale Agile by expanding the scope of Agile concepts used to manage teams.
- Agile long-term planning expands team-level sprint planning to a larger scale using longer backlogs and graduated increments of time and size.
- Simplify planning by scheduling work according to the throughput of the teams doing the work.
- Ecosystem-level Kanban expands team-level Kanban to visualize the flow of larger increments of work across an ecosystem of teams that work closely together.
- Ecosystem events are defined to manage a cadence of meetings to facilitate alignment across teams and their stakeholders. The exact number, frequency, and outcome of ecosystem-level events will vary based on the context of the ecosystem.

8

Teaming in an Agile Ecosystem

In the first section of this book, I made the point that Agile teams are a great, safe way to start learning the skill of teaming. Notice the word *start*. In an environment of any scale, teams will often need to collaborate on shared outcomes. Grouping teams into an Agile ecosystem is a reasonable way to try to contain these dependencies. But this approach does not eliminate cross-team work; rather, it just compartmentalizes a portion of it. This means collaboration will need to occur across teams. People in an ecosystem will need to learn how to engage in teaming with each other.

Throughout the book, I've pointed out some common patterns and anti-patterns that I've seen in my work with various organizations. In this chapter, I'll dig deeper into these and other patterns that explain different ways team members can interact and collaborate across teams.

AUTONOMY, NOT INDEPENDENCE

Remember how we talked about there being no such thing as independent teams? That's a pretty sad thought if we want teams to self-organize. Well, the good thing is that teams do not need independence;

what they really need is autonomy.

When talking about a team operating in the context of a larger organization, it is important to separate team independence from team autonomy. An independent team works in isolation of everyone else. It is an island. It does not pay attention to assets it cannot leverage. It duplicates the work of other teams, and it falls prey to NIH (not invented here) syndrome. This kind of team gets into trouble, as it doesn't learn from the mistakes made by other teams.

In contrast, an autonomous team is a team that has *the freedom to determine how much independence it needs* to self-organize toward value creation. Team autonomy isn't about going it alone; it is about giving the team the autonomy to choose who it depends on. An autonomous team can and will depend on other teams, but it will do so with purpose. It will do so to reduce its cognitive load and to get help with the things that other teams do better. When autonomy is granted, independence becomes a question of choice. It is safe to say we want teams to be as independent as they can be.

It is hard to start out teams with the level of autonomy we may want them to have. Autonomy is about responsibility. Giving teams the power to be responsible is one part of the equation, but teams using that power to behave responsibly is just as critical!

ORGANIZING CONSTRAINT

Manage Cross-Team Dependencies Through Teaming

When teams have autonomy, they are expected to work closely with other teams to collaborate on shared outcomes. We do not want teams to constantly be handing off work to other teams. The old fire-and-forget model does not work. Not between functional departments, and certainly not between autonomous teams.

Instead of handoffs, it is better to manage these dependencies through interactions based on *teaming*, but this time our teaming takes place *across* teams.

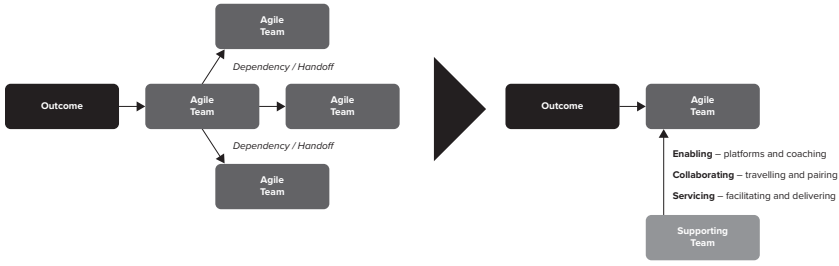
Yes, we can have team members team across teams. In fact, it is critical that team members do so, especially teams that are within an Agile ecosystem. In that vein, there is a new twist in our teaming tongue twister.

So, stated fully:

Team members need to team with other team members from other teams to be an effective team at greater-than-team scale.

TEAM MEMBERS TRAVEL TO THE WORK; WORK DOESN'T TRAVEL ACROSS TEAMS

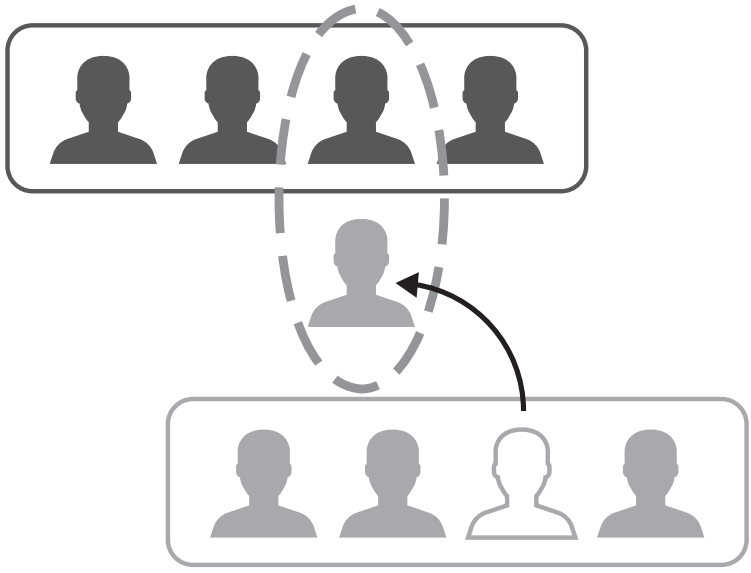
Perhaps one of the most counterintuitive ideas introduced to me by David J. Anderson's work on Kanban is that work does not travel across teams. *People travel toward the work.* If we want the concept of Agile teams to scale, then we do everything we can to avoid working in a silo. We need to scale the concept of teaming. This means people need to pick up and leave the comfort of their own team. They get as close as they can to the work and the people they are depending on to get the work done.



Don't let work travel across teams; travel to the work instead

In simpler situations, we can think of this traveling in terms of one team acting as a client while the other acts as a kind of provider. The team that requires something from another team is the customer, and the team with the dependency is bringing value to that customer. The

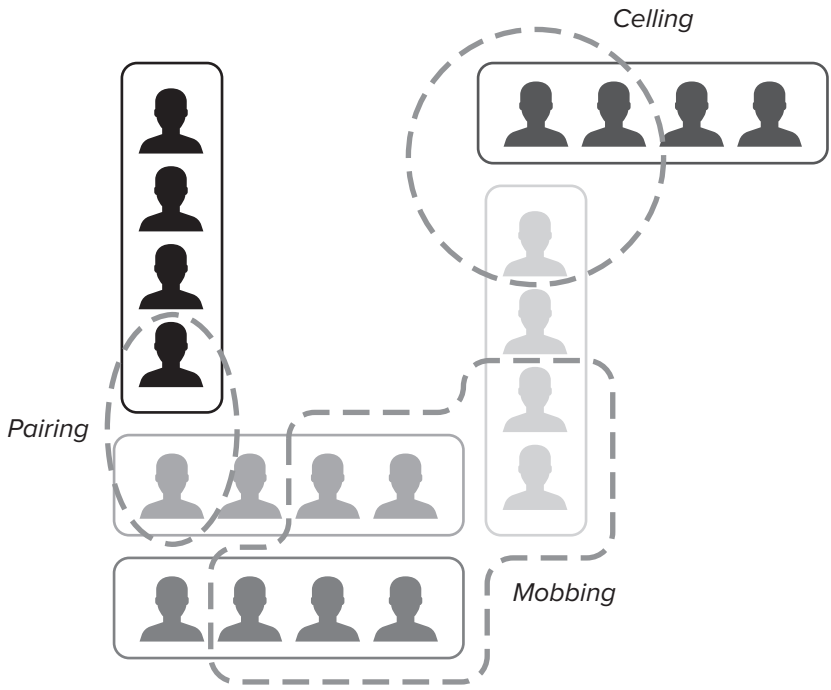
provider travels to the client to team with them. It is important to note that we are talking about an Agile form of customer. An active customer who will work closely to ensure value is created, not an order-giver that hands things off and forgets about the work until it is done.



The provider travels to the client

The real world is often messier than this simple depiction of client and provider. People from different teams will need to temporarily come together for all kinds of reasons. Shared discovery, integration, problem solving—the reasons are endless. When teams need to share an outcome, they can take advantage of pairing, mobbing, celling, or a myriad of other forms of swarming to effectively work with each other. This can be hard to pull off, especially for teams new to the ecosystem concept. The trick is to create space for ecosystem members to engage and interact with each other outside of their teams. Both formal events (e.g., ecosystem-level planning, stand-ups, retros, etc.) and informal events help to bring this larger team sense.

There are a number of patterns we can look to in order to better understand teaming across team boundaries; this we will do, as we progress through this chapter.



The need for teaming will always defy your team structure

In a nutshell, Agile teams work because they provide a means for people to engage in highly collaborative work. They provide a number of methods *that enable teaming*. Agile scales when people can take teaming to the next level and when individuals have the confidence to leave the comfort of their own team and to team with a larger group of people. Agile scales when people exhibit the social density necessary to step outside of their officially slotted teams and to team around the value instead.

MINIMIZE DIRECT DEPENDENCIES; MAXIMIZE INDIRECT DEPENDENCIES

One way to manage dependencies is to think of value-creation activities as either direct or indirect activities. Even a team that fully owns its market-value creation will likely need help on what I will call *indirect*

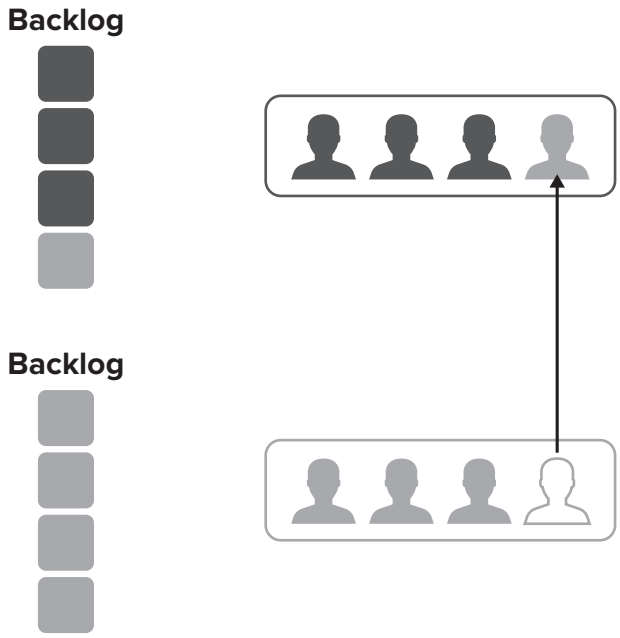
or enabling activities. Think of a sports team. The team is responsible for winning, but a lot more activities go into winning than playing the game—talent scouts, logistics, getting people into the sporting venue, career counseling. However, the team doesn't necessarily want to do all of these things. The team likely wants to focus on the game. Likewise, in an organization of any scale, teams will want to maximize independence on direct-value creation and to retain autonomy for everything else, while actively receiving support from other teams.

There are three patterns that I have seen people use to effectively collaborate across teams, especially within a common ecosystem: the traveling team worker, the enabler, and the service provider.

THE TRAVELER

The first is what I call a traveling team worker, or traveler for short. A traveling team worker is exactly what they sound like: they spend a lot of their time collaborating with members of other teams. They are temporary teammates that show up and provide a particular set of expertise.

Travelers are team workers in an Agile team where there is a dependency between their team and another team. Travelers resolve that dependency by becoming a temporary member of that team until the dependency is resolved. They do work by traveling to the team where the work is and collaborating closely with members of that team.



Travelers emerge when another Agile team has a need for the traveler's specific capability. Maybe the traveler's skills are highly specialized, are hard for that team to learn, take a long time to master, or require extended training, intimate knowledge, and extensive experience to be effective. There may be a limited number of people that have this skill set, and it is challenging to increase that number.

Becoming a temporary member of the team solves this problem. Dedicated team workers in teams within the same ecosystem will often travel to other teams in the ecosystem to collaborate with them. This happens when it is not possible for two or more teams to completely isolate their work from each other. In the case where a team has a discrete dependency on another team, it makes sense for a team member from one of the teams to go into traveler mode and act as a team worker on the other team. This allows us to minimize situations where teams work on a shared piece of work in isolation from each other.

Travelers allow you to minimize handoffs in the face of demand that does not lend itself to stable, static teams. One of the biggest mistakes people make when organizing teams is to structure teams based

on capabilities and then allow work to be distributed across multiple teams, consequently relying on cross-team coordination and handoffs between teams. We know handoffs reduce a team's ability to self-organize and respond to feedback.

Remember, handoffs reduce agility.

An alternative option is to use the idea of travelers instead; that way we *move people to the work*, rather than moving work to the people.

Getting back to our payments group in Scotiabank, their digital channels team was facing deactivation of a critical, customer-facing payments service due to noncompliance with mandated enterprise security requirements. In parallel, a separate team was already in the process of reengineering the legacy payment-processing system, replacing it with a modern payment product solution. This provided an opportunity for the digital team and payment modernization team to work toward a common outcome. Given the interdependencies involved, working separately would not be ideal. After some initial discovery, the majority of the work appeared to be in the digital channel team's domain, with a smaller but still significant amount of work still required from the payments modernization team. A couple of payment modernization developers agreed to travel to be a part of the digital team until the work was completed.

Further examination of the digital channel team's backlog revealed features that would require the payments modernization team to work closely with the digital channels on shared solutions. The traveler concept became recognized as the preferred approach to working with other teams and was a better alternative than the previous approach of handing off work.

Many object to the notion of a traveling team worker and ask, How is this different from old-school project management with temporary teams and matrix management? It's a good question. Perhaps the biggest difference is that travelers are not told to travel by some project manager or figure of positional authority. Travelers *choose* to travel to work with other teams. Remember, in most cases traveling happens between members of a common ecosystem or tight-knit groups of teams. This ecosystem has a level of social density, so people are not going off to work with strangers. As a result of traveling, familiarity and bonding increase across a smaller, tight grouping of teams over

time. In addition, we want ecosystems and their teams to use Agile practices to both visualize and limit their WIP at both the team and ecosystem levels. This practice should include traveling work as well as dedicated teamwork. The practice of limiting WIP prevents the abuse we often see in familiar project practices. Visualizing work enables team members to identify when and where they need to travel.

It is important to note that traveling is not always easy and can cause stress for the traveler. The person must be comfortable working with the new team and have an aptitude for fostering an environment of openness and collaboration. Care must be taken to check in if people are constantly switching from team to team, as burnout can quickly become a very real problem. Again, traveling can work well when it is a mode of operation *chosen* by the person acting as the traveler. When traveling is mandated, we often see poor collaboration and protective behavior on the part of the traveler.

Traveling Intent Owner

Very much like a traveling team member, in the case of the traveling intent owner, the team member of another team brings an outcome, a cohesive increment of value, or a piece of demand with them in order to facilitate its completion by the team they are traveling to. The owner of the intent travels to the other team and works closely with the team, owning the definition and validation of success for the work. The experience of Brent Reynolds at CCC is a perfect example of the traveling intent owner:

> *We had a number of business-facing pods that were responsible for end-to-end ownership of specific business outcomes. The business pods included a mix of product, marketing, and operations people. Members of business pods often generated new ideas, which we called "intent." The person who generated a new piece of intent was also responsible for owning it, ensuring both delivery and operations were successful. We could call these people Intent Owners.*

Some of the business pods were focused on improving the digital experience for our customers. These pods would have sister pods from our digital factory assigned to them as part of our planning process we held every few months.

Anyone in our business pod could become an Intent Owner for work going to their sister digital pod. As such, the digital pods didn't have a permanent product owner representing the business. Instead, the business, in the form of an Intent Owner, would go to the digital pod. They would literally pick up and move to the digital pod to play the role that we see dedicated product owners often playing on Agile teams.

By traveling, people from the intent-forming pods were able to help the more delivery-focused teams and bring in their business expertise and business ownership at the right time. And we avoided messy handoffs. It took some time, but the people in the intent-forming pods developed a strong appreciation for the digital pods they were working with, and vice versa.

ANTI-PATTERN

The Ad Hoc Team Member

The reason many Agile enthusiasts react negatively to the idea of a traveler is that they fear a return to the old days where everyone felt like an ad hoc team member. The team was never stable. People came and went at the behest of management, often with very little collaboration or even communication with the team.

An ad hoc team member is the dark side of traveling. An ad hoc team member shows up unannounced and disappears with very little notice. The ad hoc team member is told where to go by functional managers and told when to leave. Where

adequate notice is given, choice is not. Teams don't have any choice in how people are moved.

Ad hoc team membership kills any sense of teaming; it cripples morale and erodes trust. Remember, for any kind of traveling to work, we must address the needs for the people impacted. They must be included, to the maximum extent, in the decision and have a hand in defining what it looks like. Ultimately, we need to move to a place where traveling to a team is a choice.

THE ENABLER

Another common pattern in Agile ecosystems is the enabler. An enabler also travels to teams, but their mandate is primarily to assist the team in improving their capabilities. Enablers coach, advise, and provide platforms for self-service. Traveling team workers are focused on doing, whereas enablers focus on guiding. This distinction will feel important in some cases and irrelevant in others; in some cases I have seen people simply use the term *traveler* to describe both patterns of interaction.

Most organizations have a small number of people who possess a particular set of capabilities. These capabilities can serve as foundational building blocks for other teams, so organizations want to make these capabilities available to teams in ways that give them a strategic advantage in the market—perhaps increasing safety, security, or automation. The small group of people who possess these capabilities collaborate with other teams as *enablers*.

Enablers are responsible for supporting multiple teams, typically within a single ecosystem, to be more effective at a particular cross-cutting capability deemed important to the enterprise. They enable this capability by providing guidance, a platform, relationships, and knowledge.

As an enabler, you are responsible for bringing your knowledge to team members. You may also provide a platform so that teams can self-serve. You are not responsible for doing the work of teams; you are

responsible for helping team members be more effective.

Enablers are a special kind of traveling team worker. They show up with a mandate to teach, to advise, and sometimes to verify that teams are improving in a specific capability. They help manage organizational risk by helping other teams to deal with that risk.

Enablers help teams to be effective by empowering them to operate in a self-service mode. Technology platforms, bodies of knowledge, teaching aids, for example, are all tools of the trade for enablers.

Inside of ecosystems, we often see senior system and domain experts acting as enablers. Our positional leadership, our managers, and their managers, when acting as servant leaders, are playing the role of enablers. Agile coaches are another example of enablers.

One of the biggest challenges with enablement is that it is confused with authority. For enablers to be effective, they mostly direct their effort to support and service; you cannot force someone to be enabled and expect the results to last.

As with travelers, there are subpatterns of the enabler in many organizations. Two variants in particular are worth a closer look.

API Enabler

When team members help other members of teams use their APIs, they are using the enabler pattern. Teams can struggle to create APIs that are used appropriately and effectively by other teams. This is often because API teams focus on building the API, but when it comes to providing the support required for other teams to make effective use of the API, enablement is often an afterthought. Enablement is critical for shared APIs to be successful. An API enabler thinks of their API as a product and considers the developer experience as a primary requirement when building the API.

In the ATB business banking program, the customer relationship management (CRM) squad had numerous dependencies on APIs developed by the banking API team. To manage these dependencies, the API team made sure to include at least one member of the CRM squad in all of their cadences, such as planning and stand-up meetings. Tara Gamache, a member on the API team, discusses how they

better enable the CRM team to use their APIs:

> *We select an API dev to do a "transition"/"walk-through" of the service with the CRM dev looking to use the service. Several artifacts, made available through a code repository in Git, are used to guide the discussion: 1) a swagger (.yaml file), 2) design implementation details in markdown that contain the orchestration flow, API rules, provider service mappings, scope, reference docs, etc., and 3) a set of automated tests. The CRM developer reviews the artifacts, then meets with the API developer and walks through them together so that the CRM developer has an opportunity to ask questions and further their understanding.*
>
> *In my experience, our enablement process works really well for new consumers of an API, especially if they have never consumed a digital end point before. There is a lot for them to be aware of (security tokens, the client IDs, secrets, the end point request/response structure, the debit card encryption). Lots of moving parts that are not easily understood by just calling the URL of an end point. As part of the review there are often gaps and new requirements from CRM that get revealed. We are able to come together quite quickly and determine how to best meet these CRM requirements, create stories for the effort, and knock those out of the park.*

Capability Enabler

Enabling a team to use the APIs you build is one thing, but delivery teams can find themselves in a situation where it makes sense to train another team on how to deliver on their stack, system, or what have you. By doing this you empower that team to break their dependency on you completely.

In the ATB business banking program, the mobile team was

starting to become frustrated with some of the work they were doing. They were tasked with taking the features by the other teams in the program and adapting them from the original desktop version over to mobile. This approach was painful for a number of reasons. Despite the best efforts of other teams, the mobile build almost always revealed defects in the code produced from these other teams. In effect, the mobile team ended up acting as a kind of extra QA for these other teams. Every other team in the program acted as a source of failure demand (new work that comes from not doing work correctly the first time) for the mobile team, who had to invest a lot of cycles to triage bugs found as part of their development cycle. If you asked the mobile team to talk about their experience, you would hear the following:

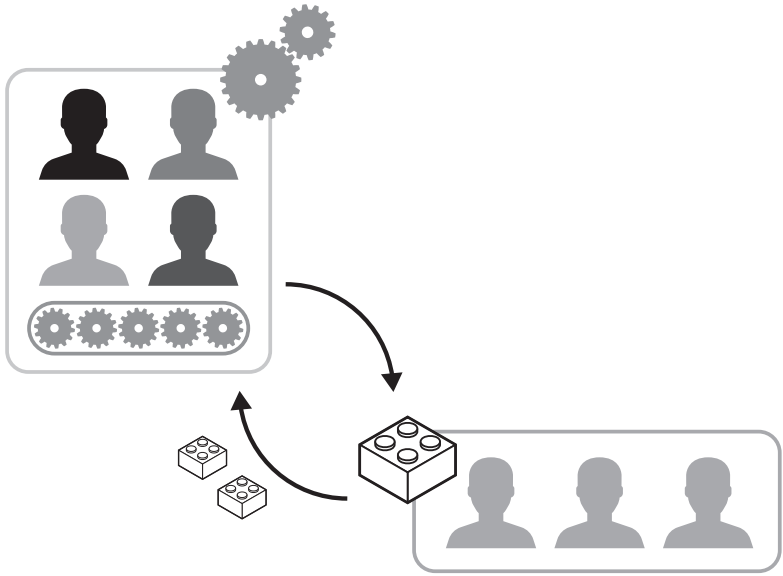
> *The other teams didn't develop for mobile—and they handed it to us. We tried stealing devs from the other teams, we had jacks of all trades for all payment rails, and it wasn't working well. For us, it took double the amount of time navigating through someone else's code. They would throw their trash onto the island—as long as they didn't have bugs in their backlog . . . we ended up with a huge trash heap.*

The mobile team started thinking about different ways they could approach the problem and came up with a mobile-first enablement approach. They would train and otherwise empower other teams to design, build, and test directly for mobile, and then use the ability of their platform and tools to enhance the solution for desktop sites. The mobile teams spent several months training teams on the nuances of mobile design, the specifics of testing automation using mobile devices, and how to quickly enhance their mobile code to handle a desktop view. The teams found that developing mobile first and enhancing for desktop after required less work than starting with desktop and trying to simplify for mobile afterward. More importantly, it was a lot more effective for a single team to own a feature across both mobile and desktop. There were fewer handoffs, and domain knowledge, testing, and so on didn't need to scatter across teams. The mobile team participates in various shaping and design sessions to help facilitate

sharing and consistency of experience. The team is also the first line of support for things like the environments, deployment pipelines, and any other shared mobile assets.

THE SERVICE PROVIDER

The final pattern we're going to look at in this chapter is the service provider. Service providers perform just enough collaboration with a requesting team to understand their ask, then perform the bulk of work required to complete that request without the other team, perhaps conducting periodic feedback sessions with the other team. Despite what the Agile literature may tell you, all organizations will have some people engaging effectively as service providers to other parts of the organization, but be careful with the service provider, as in many cases it can be an anti-pattern if we want people to deliver value through hypercollaboration.



These workers service the needs of other people in an organization by fulfilling an order request, largely in isolation from the person making the request. Service providers work closely with other service

providers who have similar capabilities and do not work closely with the team requesting their services.

Service providers need to fulfill requests from many different teams. Their individual work requests tend to be smaller in duration/effort. There is limited value in collaborating closely with the client to complete the work; all service providers need is an understanding of the clients' needs, after which they can complete the work, perhaps getting feedback on the interim work product.

Work can be thought of as traveling to the team in this model. Service providers may offer up-front and periodic consults to requesting teams, but the vast amount of their work is completed independently. The majority of their time is spent in their home-base team.

Of all the collaboration patterns presented so far, relying on this one will lead *to the least amount of organizational agility*. Consider other team collaboration patterns if you want to demonstrably improve organizational agility.

Service providers reduce agility because every time you are relying on them you are creating a handoff! Handoffs erode self-organization. They cause lead times to go up, and they require coordination, typically management coordination across the teams.

Most service providers will not have your context. They won't understand why they are doing something for you and will not understand why it is important to your customer. This lack of context causes poor results that often miss the intention of the person who needs the work completed. The result is rework and reduced efficiency, the very opposite of the reason we want to use a service provider in the first place.

You will find some situations where the service provider is the best way to go. A team needing to make small, simple changes to their API may be an example. A good service provider takes work on because it is easier for the person making the request. As soon as working in isolation is causing an issue, they should travel to the team making the request and engage more collaboratively.

For those interested, Matthew Skelton and Manuel Pais, in their book *Team Topologies*, discuss some concepts similar to the patterns I have mentioned above, as well as some other concepts mentioned in this book. They use a different taxonomy than I do, but come to some

similar conclusions.

IN SUMMARY

- In an enterprise, it is more important to maximize team autonomy—in other words, their decision-making power—than to achieve true independence from other teams.
- Dependencies across teams are best managed through teaming, encouraging team members to the work, rather than work being dispersed across teams. Where possible, turn direct dependencies that rely on work into indirect ones that rely on platforms and knowledge.
- The traveling team worker, or traveler, is someone who travels to another team for a short period of time to closely collaborate with them on a piece of work.
- The enabler is someone who helps another team build their expertise in a capability that they possess deep knowledge of, perhaps a skill, role, or API.
- Unlike teams, the right Agile-style event cadences for ecosystems will be much more dependent on your context. For instance, teams that are tightly coupled on a program will use different events than teams that are more loosely coupled by a common mission.
- Potential ecosystem events can include mirroring the team-level events (e.g., planning, stand-ups, retros, and reviews) as well as potentially ecosystem-only events such as shared stakeholder replenishment, long-term planning, team-to-team synchronization, rotating stand-ups, and open spaces.

9

Reteaming in an Agile Ecosystem

If you remember, I asserted that team stability is not the be-all and end-all many Agile advocates claim it to be. Organizations made up of teams need to become much better at evolving their team structure, growing teams, shrinking them, disbanding, or creating teams. We need to be able to arm our people with the mindset and skills to organize around value and around outcomes. This means any static organizational design will get in the way, as outcomes change, markets surprise us, technology advances, and new opportunities come our way.

An organizational model based solely on products will be wrong, a model based on customer experiences or customer segments will be wrong, and so will any other model based on anything but the market outcomes your organization is trying to achieve at this exact point in time.

STABLE TEAMS ARE EFFECTIVE TEAMS

So, first off, you need stability, absolutely. Without stability people in teams do not have a chance to, well, team. Without stability, teams do not get the chance to gel, nor to achieve high performance. Rework

and distraction from part-timer syndrome tend to be the norm without the dedication of team members. Learnings that the team gained and any adoption of new culture and values also tend to evaporate once a team is disbanded.

INDEPENDENT TEAMS ARE EFFECTIVE TEAMS

On top of being mindful about stability, we need to be mindful that it is harder for teams to effectively self-organize if they don't own a major portion of their scope of the work required to accomplish their outcome. If a team has to hand off work to other teams in order to be successful, if they can't make a market decision on their own, then they have reduced their ability to independently self-organize. They now need to organize with another team.

However, independence can erode with time. Again, experience with teaming across teams helps here; indeed, cross-teaming patterns where team members learn to be traveling team workers and enablers is a solid first step for teams to handle the mismatch between team structure and demand.

But even with good cross-teaming in place, we eventually get to a point where our current team structure is in the way. Our teams risk becoming an artifact of bureaucracy rather than an indicator of how people are actually working together. This can become expensive, teams can slow down, and it can become harder for them to react to feedback.

I personally have borne witness to how a nicely designed ecosystem of Agile teams, left unchecked, devolves over time into a mess of dependencies. Sometimes in a matter of mere months. No team could deliver any item of value on their own, and no team could own any particular outcome.

The reality is that your team structure will always be catching up to changes in the demand feeding your teams. Eventually, your teams end up working on the wrong thing, projects will need multiple teams to work on them, or teams can't work on something, as they lack the skills or access.

So, we have two problems:

1. It is very challenging for teams to work well unless they exhibit some form of stability.

2. It is also very challenging for teams to self-organize if they always have to work with other teams to get anything done.

The reality is that *team stability and team independence are concepts that contradict each other.*

INTRODUCING DYNAMIC RETEAMING

In her book *Dynamic Reteaming*, Heidi Helfand strongly asserts that "whether you like it or not, your teams are going to change. . . . Recognizing team change as a natural occurrence is a key point. . . . In essence, team change is inevitable, so we might as well get good at it." Helfand describes dynamic reteaming as the act of continuous change on teams. These changes could be as simple as the addition or removal of a team member, pulling team members off multiple teams to form a new team, or even dissolving teams. Helfand lays out a strong case for how we can encourage teams to reflect on their own compositions and determine how they might change their structure to be more effective. She points out how teams can pursue incremental reflection and adjustment to not only their product and process but also their team structure.

In fact, Helfand provides a much richer motivation for people to embrace dynamic reteaming, sharing numerous examples whereby thoughtful and conscientious reteaming resulted in an organizational platform for improving career growth, fostering a whole-team mentality, avoiding team ossification, and making it easier to integrate newcomers. According to Helfand, reteaming helps teams "learn together and do things they couldn't do before. Reteaming brings possibility."

In this chapter, we will look at different ways you can scale out Agile practices to help team members and management to facilitate the act of dynamic reteaming. I'll show how various teams with an ecosystem have facilitated an *incremental, evolutionary, and frequent approach to designing team structure*. The chapter will focus on how these approaches balance the stability and independence of teams, but don't forget the other benefits mentioned in *Dynamic Reteaming*. In

fact, I highly recommend reading Helfand's work; it is a refreshing and inspiring read. It certainly has injected a fresh passion into a good deal of this book.

ORGANIZING CONSTRAINT

Continuously Balance Team Stability with Eliminating Team Handoffs

We can empower the teams in a given ecosystem with the insight needed to collaborate on how and when to reteam. The thoughtful design of teams may not eliminate cross-team dependencies, but we can thoughtfully adjust to minimize them.

The refinement of team structure can be held at a cadence as part of the initial exploration of new opportunities. When we refine our makeup of teams, we need to take care to minimize the impact that refinement has on team stability. We want to limit the changes we make to our team structure to those that allow us to minimize handoffs across those teams.

This is a balancing act, and not always an easy one, or even a possible one. Sometimes we end up sacrificing team independence for team stability, or vice versa.

It is a counterintuitive truth that team independence trumps team stability.

So, stated fully:

Balance team stability while deliberately evolving team structure to minimize handoffs between teams.

THE AGILE SKILL OF TEAM GARDENING

Think of the team design approach we take as an act of *gardening*: we make microimprovements over time with an eye to adapting the simplest structure we can that meets the needs of the environment. As we spot fractures in our team structure, we adapt to smooth them over. If and when more-turbulent change comes our way, our people

have the natural resilience to contribute to a larger shift in how we are organized.

The good news is that people with experience in Agile teams and Agile teaming already have a lot of this inborn resilience. Because of the way they are organized, they have a variety of tools in hand to help them garden.

It is just that they are more familiar with gardening at a smaller scale than what we are talking about now. At the team level, we called this gardening *teaming*, the act of grouping into pairs of twos and threes to accomplish a fine-grained outcome. At the ecosystem level, we call this gardening *dynamic reteaming*, or simply *reteaming*, the act of several teams in an ecosystem making deliberate changes in their team roster and/or team structure to enable the teams to better self-organize around current and upcoming work.

There are limitless ways people can reteam around value, but I have also seen some patterns. These patterns can be thought of as different formations of traveling team workers, but I have seen these occur often enough to call them out specifically. For teams that are ready, take a look and experiment with some of these.

SHARED DISCOVERY AND INTEGRATION

Perhaps one of the most obvious things multiple teams need to do when working on a larger engagement together is collaborate on understanding what needs to be done, as well as validating that what was done matched that understanding.

A Kanban system can be used to indicate when an epic, feature, slice, or whatever is ready for shared discovery, after which point teams go to work on their separate pieces, then come back to integrate their work and conduct shared testing. Exactly who participates in shared discovery, integration, and testing will depend on the context.

While larger programs will invariably require some form of shared work across teams, this pattern should not be encouraged. Instead, try to configure the team so that the smallest possible number of dependencies exists. Don't lean on this pattern as an excuse to not try to form around the work.

THE CROWD SWARM

As I mentioned previously, swarming is even more powerful when a crowd—a group that does not belong to any one team, group, or department—comes together to swarm on a wicked problem.

The Kanban community has really made crowd swarming shine, showcasing numerous examples where attending an ecosystem-level or higher-level Kanban made it clear to people that they needed to form together outside of familiar organizational boundaries to get something accomplished.

PART-TIME EXPERTS

The part-time expert spends enough time with two teams to understand the context of both, making themselves useful as a part-timer. When one of the teams faces a spike in demand, or unexpected complexity, the part-timer switches to full time on that team and significantly increases the team's capacity until things get under control.

As an application architect, I had used this pattern to great effect way back in the day. My boss was concerned that the program we were running was suffering from some self-induced churn. We had scaled fast to six teams, and it was getting hard for the technically senior people to have any kind of handle on the code being produced. We were seeing some wild stuff in the field. So my boss put a couple of our most senior developers directly under my supervision. Their remit was to inject some engineering excellence into the teams. I did this by moving the developers to the teams that needed them the most. I assigned each senior developer to two teams. In periods when there were lulls, we would look for opportunities to reuse software and collaborate on developing some framework-type code that could help; in periods of team stress and crunch, the developers would step in and help any way they could. We ended up really helping these teams, and in turn got the buy-in to let us help teams by introducing some common stuff and refactoring the existing codebase.

THE STABLE TEAM-FRONTED POOL

In this pattern, a subset of the team is stable. For instance, we might have a separate stable team that possesses all the skills needed to deliver on one customer channel, such as mobile for casual clients, web for power clients, and thick user interface (UI) for operational users. The stable teams require workers that are highly specialized, but each team has inconsistent demand. For instance, the channel teams may need integration developers with deep knowledge of working with the legacy back-office product. Because the need is inconsistent at the team level, but relatively stable at the ecosystem level, the back-office workers are organized into a pool.

When new work is brought into the team, someone from the shared pool assists with discovery and determines the amount of pool work required. The team staffs up or down from the pool as needed. Kanban WIP limits are used to ensure the pool is not overloaded.

THE DYNAMIC FEATURE TEAM

With dynamic feature teams, our ecosystem has completely moved away from the idea of official stable teams. Teams are determined by an ecosystem-level Kanban. When a new item is pulled into discovery, a subset of the ecosystem does enough exploration to determine who is required to work on it to completion. This information is shared with the ecosystem members, who can self-organize into a team. Visualization helps individuals understand when and where to become part of the team. When the work is done, people go back to the pool. While this dynamic teaming can sound overly, well, dynamic, in reality patterns form quickly. Teams tend to not completely dissolve and reform; usually they evolve. Teams may shrink and grow a bit. Memory and flow serve to guide how people in the ecosystem team, not static team boundaries that may or may not have outlived their usefulness.

Stephanie and Ruth, two of our product owners mentioned previously, eventually worked with their teams to create dynamic feature teams based on skills needed on a domain-by-domain basis. The initial

discovery process was a shared practice at the program level, but since different members of the program had specific domain knowledge and expertise, dynamic feature teams were quickly assembled in real time from a shared pool of people. It was these teams that did the detailed discovery and development. In their case, a feature team would usually be composed of a business analyst and two to four developers who would possess a range of legacy application and modern app engineering skills.

As Stephanie recalled:

> *I think a lot of our success comes from the fact that it was the responsibility of team members to choose the feature cell they wanted to join.*

Ruth agreed:

> *The demand for learning, being cross-trained, the pairing on new tech, and learning all the new domains meant our entire team leveled up.*

ANTI-PATTERN

Service-Oriented Value Network

Just because the new model of delivering value is the value network, not the hierarchy, don't be fooled into thinking your value network can tolerate a large number of handoffs before breaking down. I had to learn this the hard way, and it was indeed hard. A more conservative client wanted to keep their functional structure but was open to operationalizing it with Kanban. They were open to working on flowing smaller increments of value and even trying to limit how many of them passed through their system of work. But departmental responsibility had to stay intact; the client wanted specialized workers and hard boundaries between the business, business

engagement, and delivery. We were hopeful that a Kanban system could help the client shift this mindset over time to something more aligned to a culture of teaming and collaborating.

We ended up with a value network with all kinds of handoffs. We collaborated with teams to define explicit intake and output interfaces for each service for each team. We tried to show how demand would scatter across individual teams to be processed independently, and then would aggregate into a cohesive unit of value before being shipped to the customer.

It was a disaster. No amount of coaching or coordination could get this value network to work. Things flowed back and forth, and back again. Most boards overlapped massively with other boards, and they were all out of sync with each other. What a headache. Do yourself a favor: if you want to deliver with agility, keep your value network as flat as you can. If you start encountering more than a few hops required to deliver value, start being seriously concerned.

IN SUMMARY

- While stable teams are a worthwhile goal, overly focusing on keeping the team roster stable in a changing market can mean we end up with dependencies across teams over time.
- These dependencies erode the independence of the teams and reduce their ability to deliver with autonomy.
- Dynamic reteaming is the act of continually changing teams to solve a variety of challenges, including the need for teams to be as autonomous as they can be.
- There are various patterns team members can use to decide how they want to reteam in order to work with the fewest possible handoffs across teams.

- Above all, avoid falling into the trap of laying out your teams so that they become a complex and interdependent value network, where nothing can get done without the work of multiple coordinating teams.

Part III

The Enterprise

10

From the Core to the Edge

If we remember our story about CCC, we talked a bit about how the partnerships business had reorganized their more traditional, functional-oriented organization to one based on the idea of pods and lanes (their terms for teams and ecosystems). The rest of CCC likewise underwent a shift toward Agile-style pods over several years. This shift was all-encompassing and included the remainder of their business units, their operations teams, and their support functions. Brent Reynolds, the senior executive responsible for CCC's largest business unit, in addition to standing up their digital studio, felt strongly that CCC was ready to move past command and control, and worked tirelessly with this team to put a structure in place that would put his people on the front line of decision making.

Brent had seen Agile work inside of his organization at the project level, but translating that to the rest of CCC would require greater thought:

> *Thanks to our focus on compliance remediation, we had a lot of pent-up demand; the business had basically ground to a halt as we put out the fires. There were more things than we ever wanted to do before, and we didn't know how we could accomplish all of*

this. We had a large number of audacious goals, and we knew command and control was getting in the way. We had to think about how we were going to take the idea of Agile pods and apply to them our business-as-usual activities, as well as our support and compliance groups, groups that were not necessarily on board with the change, at least at first.

A NEW ORGANIZING MENTAL MODEL FOR THE MODERN AGE

I have spent a lot of time discussing why teams are an awesome answer to the question of how to organize around uncertainty. I then expanded on the idea of the team and of teaming by introducing the Agile ecosystem. But larger organizations likely won't stop there. There are benefits in getting some form of collaboration and alignment at larger and larger scales. If we remember Dunbar's numbers, some form of social interaction is still possible even at larger scales, such as groups of 150, 500, or more.

The unfortunate reality is most organizing structures at both larger and smaller scales are often built around the ideas of control, compliance, power, and authority. But differentiated market performance for an enterprise is about increasing responsiveness across that enterprise. It requires a systematic effort to introduce cross-cutting concerns without also introducing a plethora of bureaucracy, gates, and controls that cripple feedback and learning. When we scale, we still need to succeed in a world of constant change. Many organizations have responded by systematically decentralizing decision making to the teams that have direct access to market actors external to the organization. They have made self-organization around market outcomes a first-class organizing principle across their enterprise.

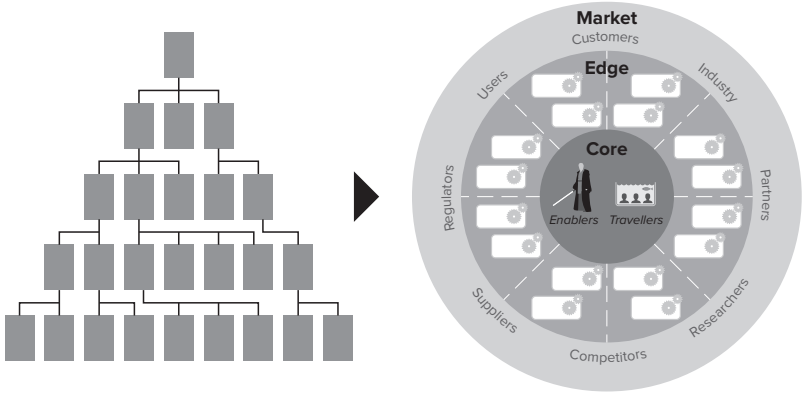
Brent shared his thoughts on some of the initial forces that would shape this new organization for CCC:

The way I see it, any functioning business needs to focus on four key things if they want to survive: it needs to

grow its market share, it needs to run things in a way that is profitable over the long term, it needs to effectively manage risk, and finally, an organization needs to really obsess about the customer. The problem is, in a traditional organization these concepts become the concerns of distinct and separated departments in the organization. Business unit leads often go for growth: you have finance people obsessing, arguably, over profitability, but really it's about expense containment; operations groups often take point on managing risk; and people in product and marketing groups act as advocates for the customer. The problem with traditional organizational design is you end up with battling business units. All of these things are important; how do you create an organizational design that balances all of these at the top and all the way down? It's impossible. I needed to take these ideas out of the hierarchy and figure out how to bring them to teams.

Scaling the idea of autonomous teams with direct market access requires us to move away from the traditional mental model we typically use for organizations. Let's pause and take a second to imagine a new one. Instead of thinking about a hierarchy of *control and authority*, let's think about a network of *markets we serve* and *outcomes we want to achieve*. Niels Pflaeging, in his excellent book *Organize for Complexity*, provides one such mental model, one that I will refer to and build upon throughout this section of the book.

Using this model, we go from imagining the organization as a hierarchical triangle, one where direction flows from the top to the bottom, to a value network segregated into circular "zones." We place people and teams into these zones according to their markets. Feedback from markets flows from the outermost zones toward the centermost zones. Functional departments are de-emphasized; cross-functional teams gain prominence. Teams act as cells in a value network that self-form, organize, and connect to deliver value.



From top to bottom, from outside in

A core idea from Pflaeging's model is that *market pull* is what connects our teams across an organization. The activity of stakeholders external to the organization initiates market pull. One or more market-facing teams, placed at the "edge" of the organization, respond to this pull and as a result may request assistance from one or more support teams placed in a "core" zone, creating market pull inside of the organization. In effect, this new mental model for the modern organization is that of a *value network*, where feedback flows inward from the market, inward to the teams on the edge, and then eventually further inward to support teams at the core of the organization. This new *outside-in* organization is better suited to respond to the needs of rapid market learning, co-creation, and meeting unique customer needs.

FOUNDATIONAL ORGANIZING STRUCTURES FOR THE NEW ENTERPRISE

The *market* is the outermost part of this organizing structure, made up of all groups that exert external pressure on our organization. Users, customers, researchers, regulators, and even competitors are all examples of market actors.

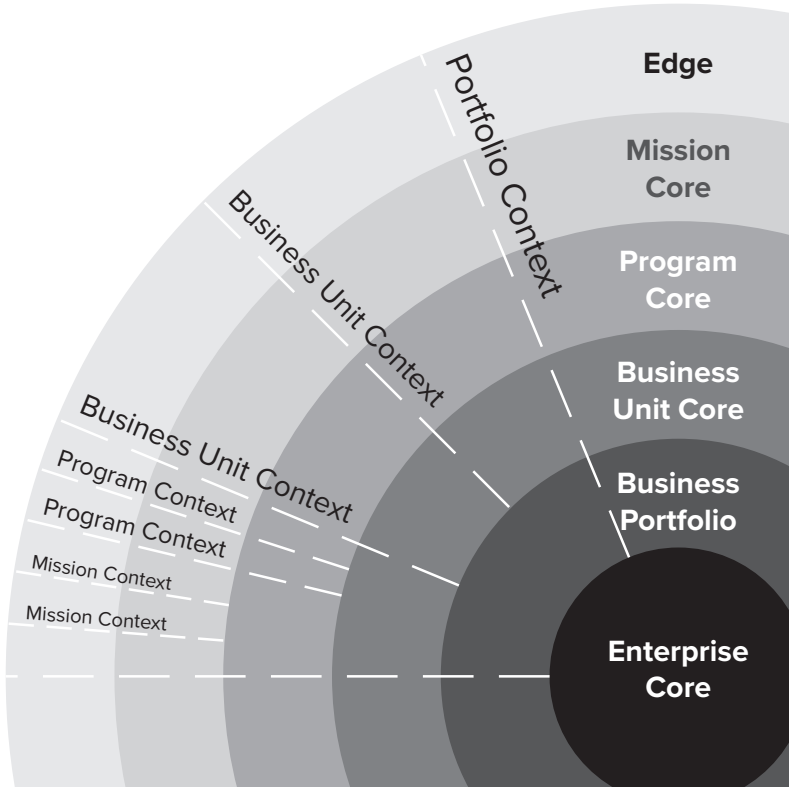
The next zone is the *edge*, made up of every team that directly

engages with people in the market. Edge teams are organized to meet external market needs and can be formed around discrete outcomes, steps in the customer journey, market segments, products, or even specific technology systems. Choose the approach that allows edge teams to be as autonomous as they can be, to possess the skills, knowledge, and permission required to make value creation decisions quickly and to adapt to markets that face constant change.

The edge in turn acts as a market to the innards of the organization, known as the *core*. People in the core serve the edge, providing common support services such as security, people ops, legal, finance, infrastructure, and common platforms. The core is different from traditional support groups in that the services provided by the core are strictly voluntary. The core has to treat the edge as its market. This means if an edge team can find a better service outside of our organization, then it is free to do so.

Everyone inside the organization is in a *team*. Each team may or may not look like a classically defined Agile team, but the need for everyone to feel like they share an outcome with a smaller group of peers is paramount. Teams are defined by the markets they serve and the market opportunities they want to achieve, not by capabilities or skills. In fact, for our teams to be successful, they require a range of skills. Without diversity, team members don't have a reason to work together!

As mentioned previously, *context boundaries* allow us to arrange teams into larger groups based on a common context. Using concepts like Dunbar's numbers and the idea of bounded contexts, we can then map boundaries around larger organizing constructs rather than single teams. For example, a context boundary of 150 people may define an organization dedicated to a line of business in a big enterprise. A context boundary of no more than 50 people may represent a discrete mission in the organization (an ecosystem of teams), while a context boundary of no more than 15 may represent a grouping of two or three tightly affiliated teams. In this way, context boundaries often nest inside of each other.



When context boundaries are used, you will likely find it necessary to model a number of *outer and inner cores*. An outer core is considered to be dedicated to a specific context boundary—for instance, a single ecosystem. It is "closer" to the edge, directly serving a smaller number of associated edge teams. In contrast, an inner core serves a larger portion of the enterprise, perhaps multiple ecosystems, an entire organization, multiple organizations, or even the entire enterprise.

WHAT'S IN A NAME?

In Pflaeging's book *Organize for Complexity*, these zones have different names: the sphere, the periphery, and the center. I use alternative terms simply because they better resonated with most of the clients I have worked with. Use whatever terms will resonate best with the

people you are working with. Some alternative terms for each of these zones that we see in the field include:

- **The teams:** pods, squads, labs, bands
- **The market:** external context, environment, situation
- **The identity:** sphere, organizational boundary, enterprise
- **The edge:** periphery, market-facing teams, customer teams, value centers, feature factory, digital factory, garage
- **The core:** center, support, shared services, the glue, infrastructure
- **Context boundaries:** ecosystems, tribes, guilds, missions, programs, portfolios, business units, subsidiaries, divisions, studios, garages, lanes

Whatever terms are used, the new outside-in team model has definitely gained traction among many organizations in recent years. That said, it is true that the *official* hierarchical organizational model is not going away anytime soon. To date, only a small percentage of organizations have made a significant jump to this new outward-in concept. While the stories behind these organizations are inspiring, the majority of organizations are not planning for an accelerated move to decentralization and self-organization.

That does not mean we can't use this model to guide organizational leaders forward. We just need to *combine this mental model with approaches that can get us there incrementally*, at the pace that our organization can tolerate change. Over time we can decrease the significance of the hierarchical model by focusing on an *operating model* that is based on a better appreciation of how humans perform well. We can make incremental changes to our official organizational structure so as to better reflect our new mental model, or at least have fewer barriers that interfere with it.

With that out of the way, let's revisit some of the zones and discuss them in more detail.

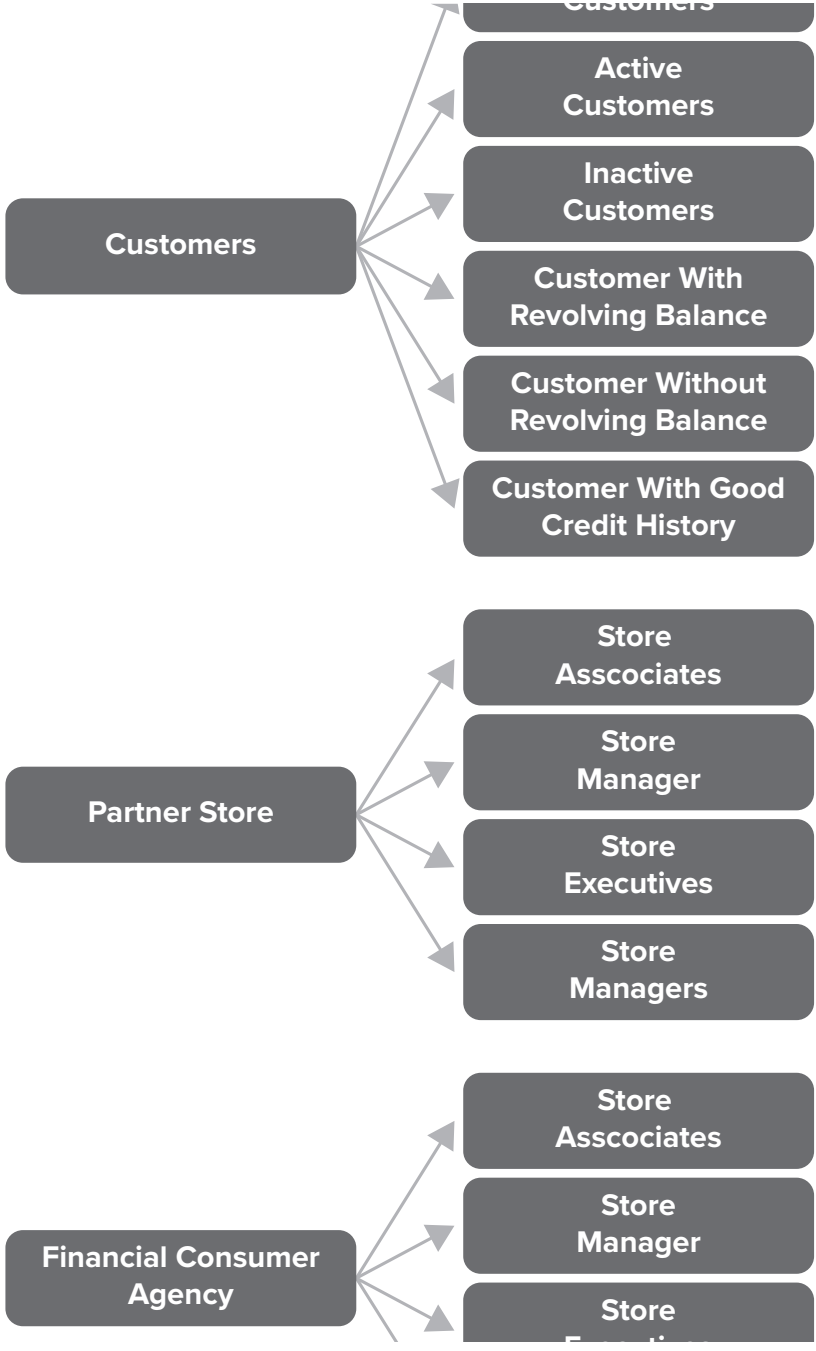
THE MARKET

The market are all the groups that exert external pressure on your organization. This list is not limited to customers and users but includes all stakeholders that influence the way your organization provides services to the market. This list includes:

- **Users** of the products and services that you provide
- **Customers** who pay for those goods and services, either directly through some form of monetary compensation or more indirectly by providing some other form of contribution, such as their time, knowledge, or presence
- **Supporting organizations** that help you engage with or better understand the market you are in, such as research institutions, educational institutions, industry guilds, technical standards committees, and so on
- **Regulators** who place constraints and rules of conduct and validate compliance for organizations working in a specific market
- **Competitors** outside the organization that serve the same or similar market

When redefining an organizational structure, we should start by identifying all of the players that our organization must interact with in order to deliver value to the market. What roles do each of these market participants play? What is our primary means of engaging with them? Are there opportunities to exercise tighter feedback loops and better co-creation?

Considering the team impact map we showcased previously, we can start to brainstorm a more exhaustive list of market actors. This is still not complete, but hopefully it paints a better picture of what we mean by market actors.


Customers
Active Customers
Inactive Customers
Customer With Revolving Balance
Customer Without Revolving Balance
Customer With Good Credit History
Partner Store
Store Asscociates
Store Manager
Store Executives
Store Managers
Financial Consumer Agency
Store Asscociates
Store Manager
Store

THE IDENTITY

The identity establishes a common identity that directs self-organization toward value creation.

Unlike other "zones" in our organizational mental model, the identity does not define where people are placed or who they engage with. In contrast, the identity contains normative artifacts that describe who we are and, more importantly, *who we want to be* as an organization.

I am not aware of any organization that has had serious success in moving toward agility without paying attention to organizational identity.

Identity done poorly gives us propaganda. We get meaningless platitudes posted on walls, principles that leaders don't understand and don't follow. Identity done well gives us some artifacts that inform the right culture and represent how leaders are actually behaving, or at least how they are striving to behave. There is a strong emphasis on articulating how the organization can foster a culture where people are empowered to make decisions for themselves. Good identity artifacts give us something to point to when we fall back into familiar old ways of thinking and behaving.

There are some great examples of artifacts we can use to help establish our organizational identity. Almost all of these artifacts are informal and focused on telling a story, not on creating a false prescription of artificial precision.

Good identifying elements include:

- **Business models** formed using the practice of business model generation[22]
- Taking the time to establish a common set of shared values and beliefs[23]
- Full participation of **common rituals**, such as shared morning updates, stand-ups, or periodic open spaces

22. Alexander Osterwalder and Yves Pigneur, *Business Model Generation*, New Jersey: John Wiley & Sons, Inc., 2010, https://www.strategyzer.com/books/business-model-generation.
23. https://www.notion.so/agilebydesign/defining-agile-through-values-beliefs-and-behavior-26123c2f57fc47eba4940513baee00fe.

- Publishing a "**letter to ourselves**," a note from the CEO of the company to all its employees that explains the kind of organization we are, what we want to be, and our values that will get us there
- Developing and sharing a **culture handbook**
- Developing and sharing an **organizational brand proposal**
- A shared **backlog of customer problems**

Poor identifying elements include:

- Detailed strategies, blueprints, or road maps
- Anything more than a couple of pages
- Principles that are not immediately recognizable in the everyday actions of positional leaders
- Anything that cannot be challenged, updated for context, or revised based on learning

Remember: identifying elements foster alignment and *inspire culture*; they don't lay out a recipe for people to follow. They answer questions like: What are the biggest goals for our business unit? How can I organize around the journeys?

Brent recalled how he took an "outside-in" design approach to think about CCC's new organizational structure:

> *We started with a set of OKRs for each business unit and explored them from the perspective of various customer journeys. We did this until we had a reasonable but still relatively short-term road map of problems that we wanted to solve.*
>
> *The result was a very tangible set of outcomes for specific customer segments, such as graduating our subprime customers to a better credit score and giving them access to better products, improving the customer onboarding experience for our digital customers, providing a more real-time experience, or launching a brand-new mobile app. Attaching measures to each*

objective—like revenue generated, customer graduation numbers, and so on—was key as it gave us all something to rally around. We then proceeded to design our teams around these OKRs.

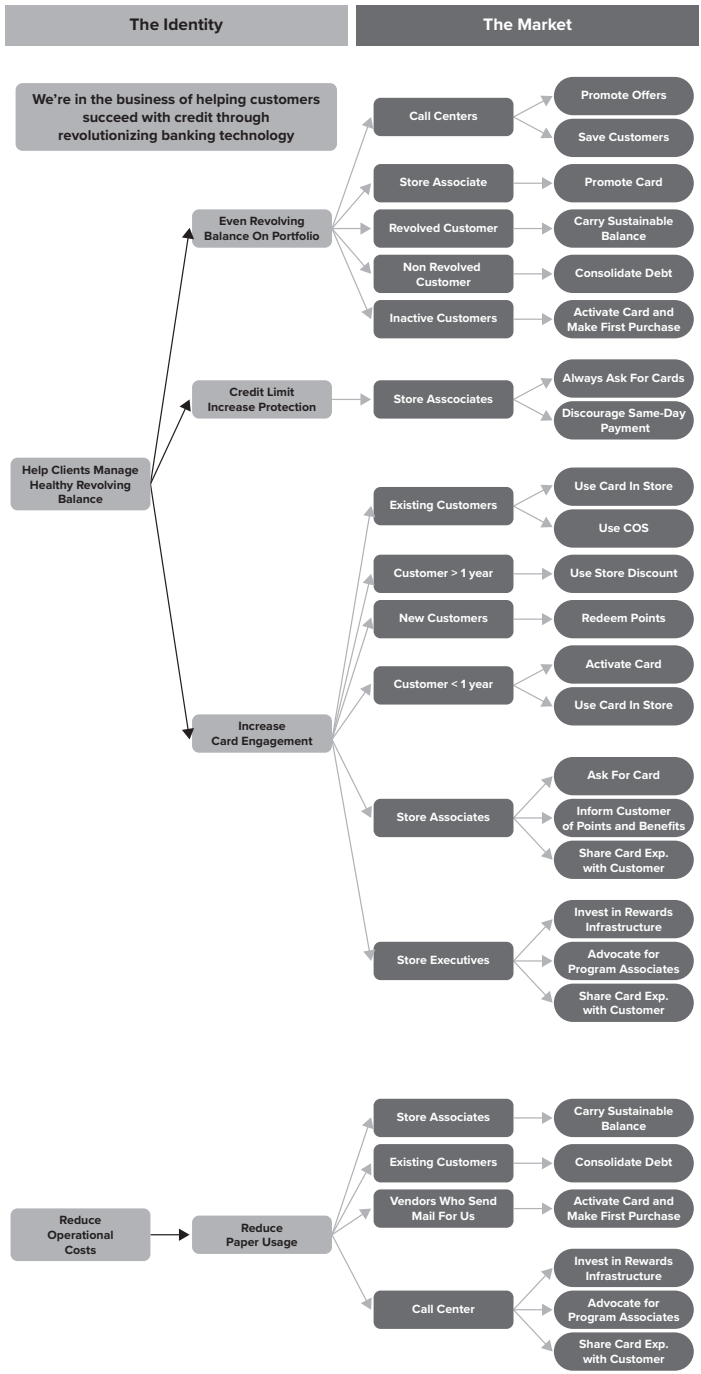
ORGANIZATIONAL PURPOSE

Perhaps the most important aspect of establishing an organizational identity is the articulation of a common, meaningful purpose.

Meaningful purpose motivates people to work together to achieve it. It gives us a sense of belonging, a community to engage with, and a chance to create together. Motivation, autonomy, and self-organization all become a matter of course. When people can contribute to a meaningful purpose, the need for expensive and crippling bureaucracy slips away and people use values, not rules, to guide their decisions.

There is a lot of recent research that backs this up. We can point to DDI's Global Leadership Forecast, EY Beacon Institute and HBR's *The Business Case for Purpose*, or BetterUp Labs' Meaning and Purpose at Work for evidence linking stronger purpose to increased employee motivation, better organizational financials, and improved customer loyalty. Organizations that have a strong purpose naturally gravitate to an Agile-oriented mindset.

Taking that thought, let's extend our team impact map with the group's first attempt at a more meaningful purpose. We can then connect the various organizational goals we came up with previously in our map to our core purpose. This is an incredibly powerful way to ask ourselves some compelling questions. Do our organizational goals make sense when framed next to our purpose? Do the behavior and impacts we want for our market actors better all parties involved? Are we being authentic in our actions and outcomes?


The Identity
The Market
We're in the business of helping customers succeed with credit through revolutionizing banking technology
Help Clients Manage Healthy Revolving Balance
Even Revolving Balance On Portfolio
Call Centers
Promote Offers
Save Customers
Store Associate
Promote Card
Revolved Customer
Carry Sustainable Balance
Non Revolved Customer
Consolidate Debt
Inactive Customers
Activate Card and Make First Purchase
Credit Limit Increase Protection
Store Asscociates
Always Ask For Cards
Discourage Same-Day Payment
Increase Card Engagement
Existing Customers
Use Card In Store
Use COS
Customer > 1 year
Use Store Discount
New Customers
Redeem Points
Customer < 1 year
Activate Card
Use Card In Store
Store Associates
Ask For Card
Inform Customer of Points and Benefits
Share Card Exp. with Customer
Store Executives
Invest in Rewards Infrastructure
Advocate for Program Associates
Share Card Exp. with Customer
Reduce Operational Costs
Reduce Paper Usage
Store Associates
Carry Sustainable Balance
Existing Customers
Consolidate Debt
Vendors Who Send Mail For Us
Activate Card and Make First Purchase
Call Center
Invest in Rewards Infrastructure
Advocate for Program Associates
Share Card Exp. with Customer

THE EDGE

The edge is made up of every team connected to the market. The teams in the edge possess the skills and have the authority to perform all of the roles required to make decisions based on market interactions. This means *every* market interaction! If a senior executive is responsible for maintaining a relationship with a strategic partner, she does not do so from the center. If necessary, she allocates the ownership of that partnership to an edge team. She may also join that edge team in order to perform her responsibilities as part of that team, even if she is only joining that team on a part-time basis. This is important! If we want our new organizational model to thrive in the face of constant change, we must move all market-oriented activity to the edge.

It is important to note that each market participant should connect to at least one edge team. Mapping edge teams to market participants is one of the first acts we should perform as part of organizational design.

There are many responsibilities that can be performed by edge teams. Some of these include:

- Building new product features, deploying them to the market, and validating them
- Customer onboarding, maintenance, and support
- Market research, strategy, and business model generation
- Marketing and creative endeavors
- Vendor and partner management
- Engagement with regulatory bodies
- Participation in professional, standards, and technical groups
- Organizational investments
- Actuary, credit, and other financial models that impact products and services
- Operational activities that have a direct impact on market participants

This is a long list! But it's important to think holistically about how we want to organize our people into teams within the edge. It is

extremely important to try to get this part of our organization right and to continually refine how edge teams are structured based on the latest feedback from the market.

A big part of getting this right is that the above list is *not* a list of teams. Things like market research or partner management are best thought of as components that are placed in teams. Any of these capabilities may be represented as organizational departments or other entities in your organizational chart, but they make for poor edge teams, as you cannot deliver a real market impact with only one of these groups. When modeling edge teams, consider how these capabilities can be combined to deliver value to the market.

We also want to design our organization so that teams own as much as they can. This includes delivering new features, operations of daily services, marketing, customer engagement, and support. In this way, they act like microorganizations in their own right.

Some approaches to organizing edge teams include organizing teams based on:

- Organizational missions or **outcomes**
- **User life cycle** (e.g., user awareness, user activation and revenue, chargeable activity, billing and receipts, support and customer care, etc.)
- Specific **customer/user segments**
- **Products** offered to customers
- Common **platforms/systems**

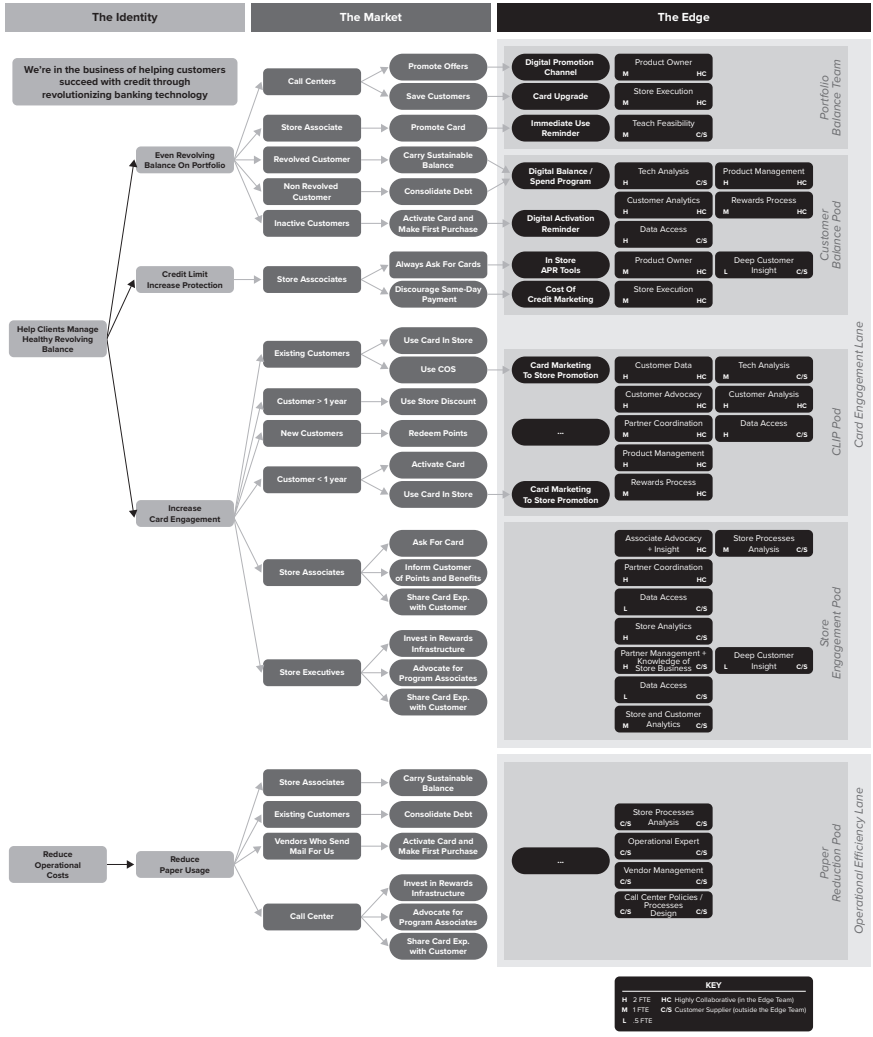
There are pros and cons to organizing your teams based on any of these attributes. It's fair to state that I have ordered this list in terms of agility. In other words, when you organize by outcome, you will find it easier for people to collaborate, self-organize, and get market feedback than if you organize by system or platform. But again, context matters. There are many short-term, and even strategic, reasons to organize by customer or product or even platform instead. The most important question to ask is which team design will help teams operate with the most autonomy and reduce the impact of a complex web of real-time dependencies.

Brent described how they went about putting the teams together:

> *We basically took the team-of-teams concept and cascaded it. We created lanes [another word for Agile ecosystems] for each customer segment, so a lane dedicated to subprime customers, another lane for upmarket cards, and another for customer acquisition and onboarding. Each lane was made of anywhere from two to four pods, which were responsible for achieving one or more OKRs attached to the lane. We mapped the skills required to accomplish objectives and tried to secure full-time team members where we could. But this met some limitations quite quickly. It was simply impossible to provide some of the skills we needed on a full-time basis.*

When looking back at our original team impact map, we only needed to make some small changes. Besides having a larger scope than our first version, we went out of our way to identify some capabilities for each proposed team. We marked each capability as either being highly collaborative or being provided as a service by another team. Highly collaborative work, marked as "HC" on the capability in question, means that we want this capability to be provided by someone inside the edge team, most likely as a *dedicated team worker*, but perhaps as a *traveling team worker* (from our previously mentioned patterns). Customer/supplier means we have identified the capability as being provided as a service from another team, a supplier team if you will (marked as "C/S" on the capability in question). This allows us to discuss what and who we want to be part of an edge team and who we want to be in a core team.

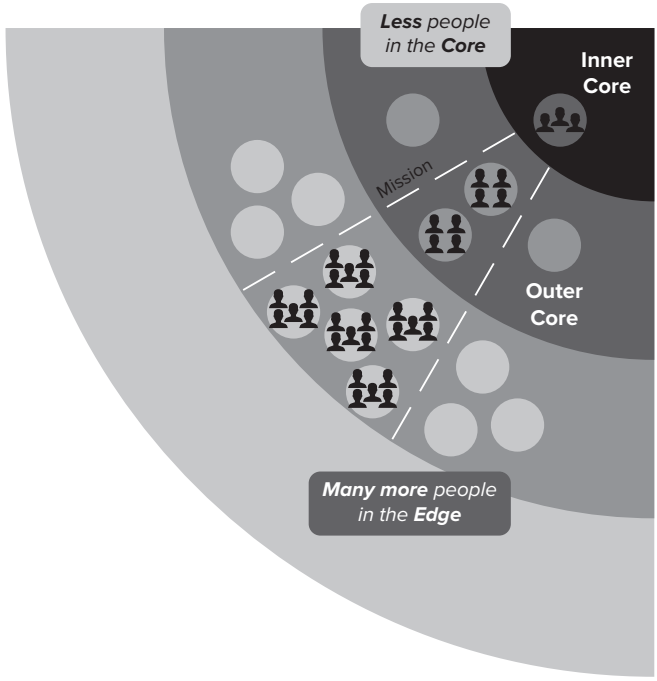
You may also notice that in some cases the deliverables have not been filled out. Because teams were fleshed out as part of the exercise, they felt comfortable mapping capabilities directly to customer impacts. This was enough for them to get a preliminary understanding of the initial team structure, both the makeup and the number of people to be placed into various teams. As the team impact gained shape, participants were also able to group teams into common lanes (their term for an ecosystem of Agile teams).


The Identity
The Market
The Edge
We're in the business of helping customers succeed with credit through revolutionizing banking technology
Help Clients Manage Healthy Revolving Balance
Even Revolving Balance On Portfolio
Credit Limit Increase Protection
Increase Card Engagement
Reduce Operational Costs
Reduce Paper Usage
Call Centers
Store Associate
Revolved Customer
Non Revolved Customer
Inactive Customers
Store Associates
Existing Customers
Customer > 1 year
New Customers
Customer < 1 year
Store Associates
Store Executives
Store Associates
Existing Customers
Vendors Who Send Mail For Us
Call Center
Promote Offers
Save Customers
Promote Card
Carry Sustainable Balance
Consolidate Debt
Activate Card and Make First Purchase
Always Ask For Cards
Discourage Same-Day Payment
Use Card In Store
Use COS
Use Store Discount
Redeem Points
Activate Card
Use Card In Store
Ask For Card
Inform Customer of Points and Benefits
Share Card Exp. with Customer
Invest in Rewards Infrastructure
Advocate for Program Associates
Share Card Exp. with Customer
Carry Sustainable Balance
Consolidate Debt
Activate Card and Make First Purchase
Invest in Rewards Infrastructure
Advocate for Program Associates
Share Card Exp. with Customer
Digital Promotion Channel
Card Upgrade
Immediate Use Reminder
Product Owner M HC
Store Execution M HC
Teach Feasibility M C/S
Portfolio Balance Team
Digital Balance / Spend Program
Digital Activation Reminder
In Store APR Tools
Cost Of Credit Marketing
Tech Analysis H C/S
Product Management H HC
Customer Analytics H HC
Rewards Process M HC
Data Access H C/S
Product Owner M HC
Deep Customer Insight L C/S
Store Execution M HC
Customer Balance Pod
Card Marketing To Store Promotion
...
Card Marketing To Store Promotion
Customer Data H HC
Tech Analysis M C/S
Customer Advocacy H HC
Customer Analysis H HC
Partner Coordination M HC
Data Access H C/S
Product Management H HC
Rewards Process M HC
CLIP Pod
Card Engagement Lane
Associate Advocacy + Insight HC
Store Processes M Analysis C/S
Partner Coordination H HC
Data Access L C/S
Store Analytics H C/S
Partner Management + Knowledge of Store Business H C/S
Deep Customer Insight L C/S
Data Access L C/S
Store and Customer Analytics M C/S
Store Engagement Pod
...
Store Processes C/S Analysis C/S
Operational Expert C/S C/S
Vendor Management C/S C/S
Call Center Policies / Processes Design C/S C/S
Paper Reduction Pod
Operational Efficiency Lane
KEY
H 2 FTE
M 1 FTE
L .5 FTE
HC Highly Collaborative (in the Edge Team)
C/S Customer Supplier (outside the Edge Team)

ORGANIZING CONSTRAINT

Move People to the Edge

Time to introduce another organizing constraint. In general, as we move to a more Agile way of working, organizations focus the majority of their people on direct market value. We see organizations transition from people working primarily in functional departments, moving out of teams devoid of market contact. Instead, they do most of their work in edge teams. In organizations with healthy autonomy and decentralization, the amount of centralized support required goes down, way down. The number of teams that work with the market goes up. The end result is more market value gets created in your organization.

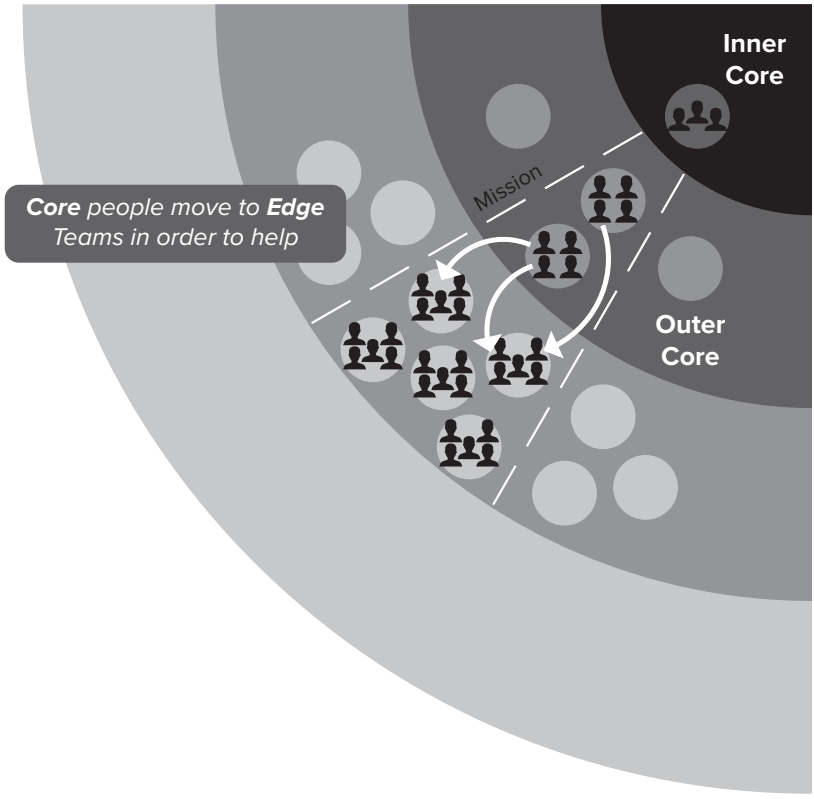


The edge is where it's at

So, stated fully:

Place the majority of people in your organization in edge teams that are dedicated to achieving market outcomes.

THE CORE



The core serves the edge

The core is made up of teams that are intentionally deprived of market contact. These teams serve the edge, providing services that the edge chooses to use. While we want to maximize each team's ability to self-organize around outcomes—and to do so with the minimal amount of dependency possible—there will still be value in taking advantage of economies of scale for some forms of work.

Most people would agree that it would be wasteful for every team to define their own hiring process from scratch, or for each team to have their own means to pay employees. Technology teams may benefit from leveraging common technical platforms like deployment pipelines or security and testing automation systems. It would be equally hard to argue that each team can independently define and maintain the overall organizational identity without some dedicated help to pull it together. Common capabilities like UX or engineering can also benefit from dedicated focus that spans multiple teams.

Virtually everything that you want to have some kind of oversight, consistency, economies of scale, or strategic perspective can go into the core. Some examples include:

- Regulation and compliance
- Finance or budgeting
- People operations (also known as HR)
- Legal
- Security and privacy
- IT
- Coaches
- Leadership and management
- Any capability management (marketing, engineering, UX, product ownership, etc.)

Unlike edge teams, you may have teams or ecosystems of teams focused on one, or often just a subset of one, of these capabilities. That is because the very purpose of core teams is to provide these discrete capabilities to teams, packaged as internal products and services that edge teams can consume. That being said, we often see the supporting functions modeled on very siloed departments—for instance, departments requiring handoffs across multiple teams to deploy a piece of infrastructure. Modeling core teams around the services they provide to the edge allows us to flatten this structure and grow more cross-functional structure in the core as well as the edge.

DEFINING CORE CAPABILITIES

As you define edge missions and teams, you will likely want to move some capabilities into teams that can focus on common, shared concerns. These are what core teams and core missions are for. When designing core organizing structures, you use the exact same process that you would follow when designing organizing structures for the edge. In this case you use edge teams and missions as the market participants for your core teams and missions. Use the needs and wants of edge teams to define goals for your core. You can then map skills and roles to your core goals and further define and refine core teams and core missions. Of course, defining your edge and core will be an iterative process; you don't need to finish figuring out your edge before getting started on putting up some core items.

Going back to CCC's Agile journey, Brent and team struggled with how to get all the skills needed across all of the pods:

> *We had a set of critical capabilities that every pod required, but it wasn't necessarily a full-time commitment, so we set up supporting pods whose responsibility was to bring these capabilities to the market-facing pods on an as-needed basis. For example, we knew almost every pod would need to occasionally engage with legal, so a pod of lawyers was responsible for assigning a traveling/enabler lawyer to a lane if necessary, and if possible from a resource perspective. In some cases a lawyer could become part of a team for a short period of time. Later on, as we had fewer legal requirements in our work, we could move to assigning a single lawyer to several lanes, or even to our entire business unit. We followed the same approach for other functions like PR, or compliance.*

POSITIONAL LEADERS AS A CORE TEAM

One way of thinking about the positional leaders in your organization

is that they are part of a core team. A collection of managers, along with who they report to, can think of themselves as part of a core team. This perspective provides a more enlightened view of managing people, one that asks managers and executives to shift their style to one of servant leadership and providing value to the people in market-facing teams. Remember: the core serves the edge!

Brent talked about how engaging with his leadership team to collectively shift their leadership approach was critical to successfully making the transition from hierarchy to teams:

> *I spent a lot of time with leaders throughout the journey. We put a cadence in place to provide our leaders with the opportunity to explore what it meant to work and to lead with our humanity. We spent a lot of time on pure trust-building exercises; we shared very personal stories with each other, stories about abuse, drug addiction, cancer. Over time our team was able to be vulnerable with each other. It was this vulnerability that allowed us to practice various conflict management instruments like the Thomas-Kilmann Framework, and the Five Dysfunctions of a Team. The idea was to model the behavior right at the top, to showcase how our associates could balance assertiveness versus cooperation, and to work weekly on any of the nits getting in the way of collaboration.*
>
> *We then went to work on a management system with an eye to serving the people in our pods. We held weekly events to help tackle sources of delay being faced by pods, we reviewed results with teams on a monthly basis with an eye toward improving financial literacy on the pods, and quarterly we met off-site in an open-space format to co-create our approach to tackling strategic outcomes and improvements.*

ORGANIZING CONSTRAINT

The Core Should Treat the Edge as a Market

Most people would recognize core teams mentioned (like HR, legal, etc.) as the traditional centralized departments and organizational units you would see in a classic hierarchical organization. So what makes the core different? The key design principles that make the core so powerful as an organizing element is that we want the product and services provided by the core to be strictly voluntary.

Remember that we want the core to treat the edge as its market. This means if an edge team can find a better service outside of the organization, then it is free to do so.

This is a huge shift for many organizations! But it is required if we want to avoid the complexity management trap. In traditional organizations attempting knowledge work (including customer-facing and business-facing application development), we see most attempts at governance struggle to be effective. Many efforts to standardize on common services provide very poor value per dollar. Reuse efforts almost always break down at some point.

This is because they ignore the complexity management trap. Complexity in an enterprise is increasingly a function of uncertainty, unanticipated change, more diversity, and continuous turbulence. By its very nature, complexity can't be centrally managed; solutions to complexity can't be standardized. You can't centrally plan against complexity either. It is impossible for support teams in the core to deliver against all the unique requirements across your organization. Whether it is responding to all the distinct market changes across the business or creating something that makes every team safe and effective, winning the complexity game through centralized management is unrealistic.

As we have stated, if we want to increase market agility in

the face of uncertainty, then we need to decentralize the organization into market-facing teams. We create edge teams that are empowered to interact directly with the markets they service and have the autonomy to make product decisions based on market feedback.

Market feedback is not meaningful unless market consumers have the choice of whether or not to use our company's products and services. But inside the organizational firewall, leaders tend to ignore this reality. When setting up internal support services, product teams are forced to use every centralized service that the enterprise has on offer. These services are internal monopolies inside the organization and, as a result, are almost always a poor fit for the teams' needs. In the real world, we know that monopolies benefit nobody but the monopoly. Inevitably customers end up being the ones that suffer when forced to deal with a monopoly.

If we want teams in the core to balance short-term and strategic needs of the teams in the edge they serve, then they need to be subjected to the same market forces that edge teams face.

The ultimate expression of this idea is that core teams manage their own profit and loss and charge edge teams for their services. Edge teams need to be able to choose to not use specific core team services and seek alternatives if their needs aren't being met.

Choice acts as a forcing function for core teams to improve their offering by providing the services that edge teams need and want, rather than core teams guessing and imposing a poor service.

So, stated fully:

Allow edge teams to choose which services from core teams to use. Orient core teams to serve edge teams.

INNER AND OUTER CORES

Likely you are working in a larger enterprise that contains capabilities and services that you are mandated or highly encouraged to use—they just come with the package of working in your enterprise. In these cases, it is helpful to model these services as coming from one or more inner cores. Examples of inner cores include enterprise-level services and managed business unit–level capabilities. Neither are likely to be far along the Agile journey, but you may be surprised.

When working in this context, all services with which you are able to facilitate change can be placed in your organization's outer cores. These services can create a good team experience, they can be thought of as voluntary, and they are deployed in a highly collaborative manner. Services from inner cores are more likely to be mandated, to offer poor value, and to suffer from a *throw-it-over-the-fence* mentality.

Services that are provided to a small number of teams, perhaps in a single ecosystem, can be thought of as belonging to the outermost core. Services that are provided to a larger number of ecosystems, a whole program, a portfolio of work, or a whole business unit can be thought of as belonging to cores that are closer and closer to the center—in other words, they are in an inner core of the organization. This concept nests, support teams that support a more narrow set of edge teams are said to be closer to the edge, while support teams that support more edge teams across multiple context boundaries (departments, business units, etc.) are thought of as being deeper inside the organization.

At CCC, Brent recalled how the various support pods stayed close to market-facing pods:

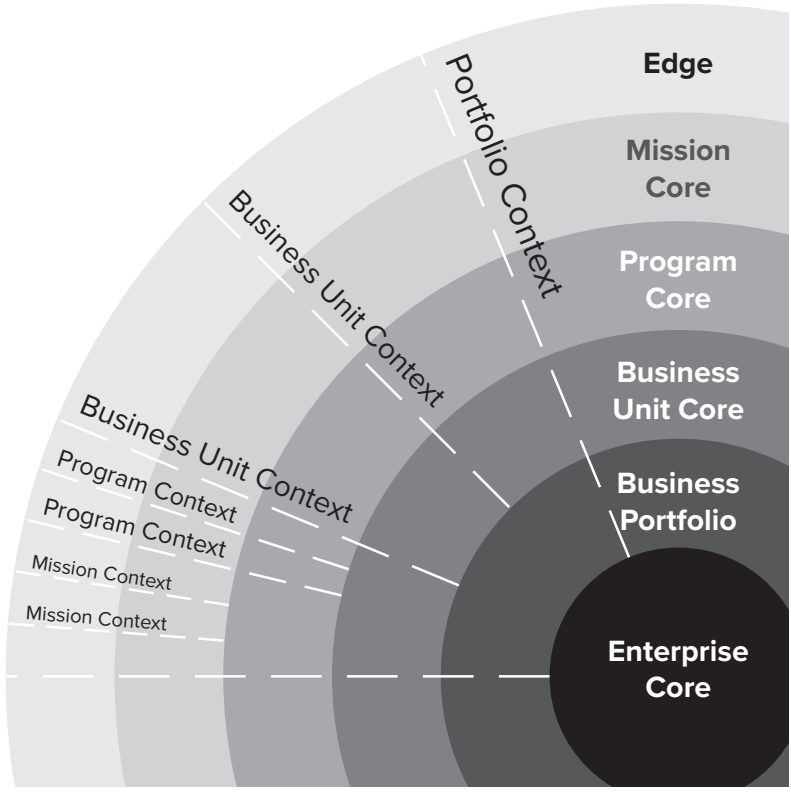
> *One of the key capabilities we knew we would need early on for the branded business was digital delivery capability. We stood up a digital lane, known as the digital studio, that would deliver digital features for pods in the growth as well as upmarket and onboarding or acquisition lanes. We managed to grow the studio to about nine feature teams in about a year. Although the digital teams were supporting the business lanes, I*

thought about them as being an extension of those pods. In most cases we would assign a business pod to a sister digital pod for a long period of time. Those two pods would act as one extended team. In other cases the digital and business pods would collaborate closely with each other, for example by attending shared stand-ups. I thought of some of our other supporting functions as being conceptually more distant from the business lanes and pods they supported. For instance, we had several creative pods responsible for designing marketing-related assets. These pods supported the entire business unit as a whole and also reported to a common creative function in the organization. HR and finance were also examples where people were provided to look after the entire business line as a whole.

If we go back to our team impact map, an example of an inner core may be a set of teams responsible for increasing technical agility across different lanes or ecosystems. (See image on next spread.)

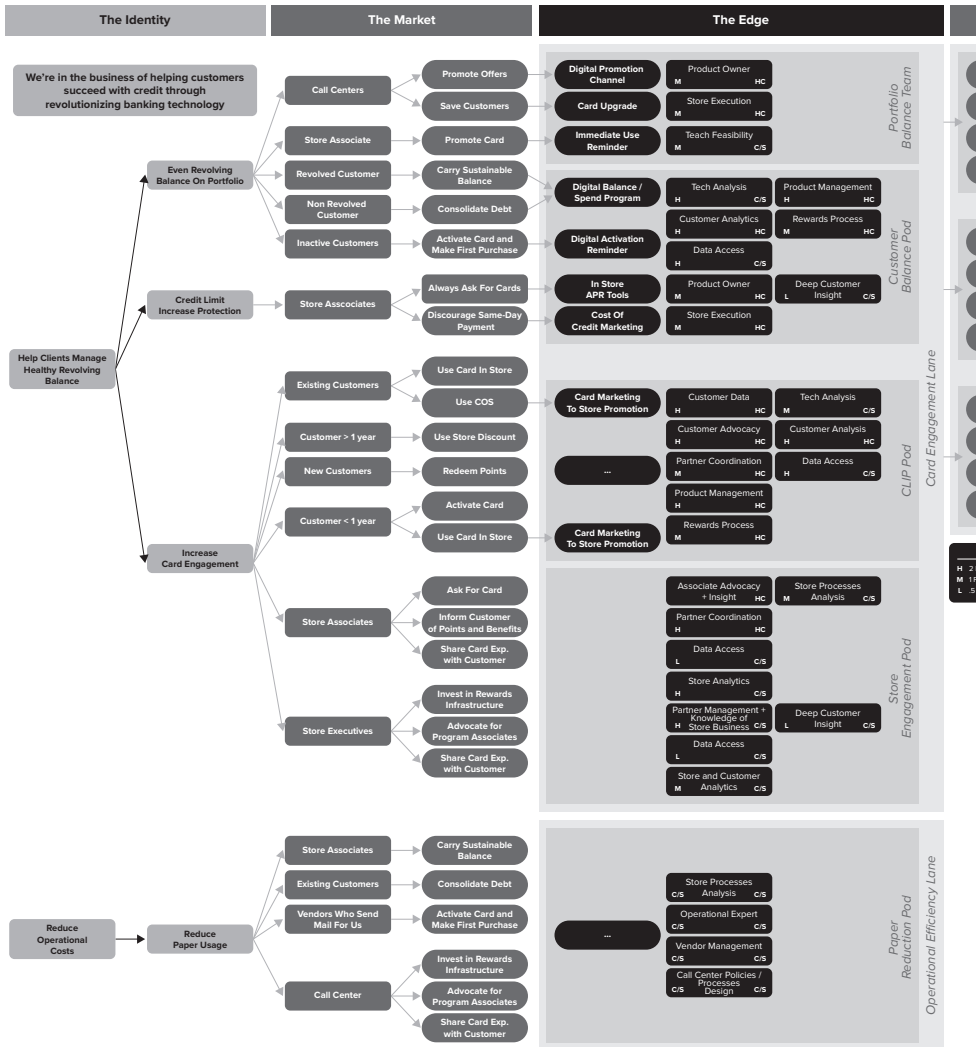
CONTEXT BOUNDARIES

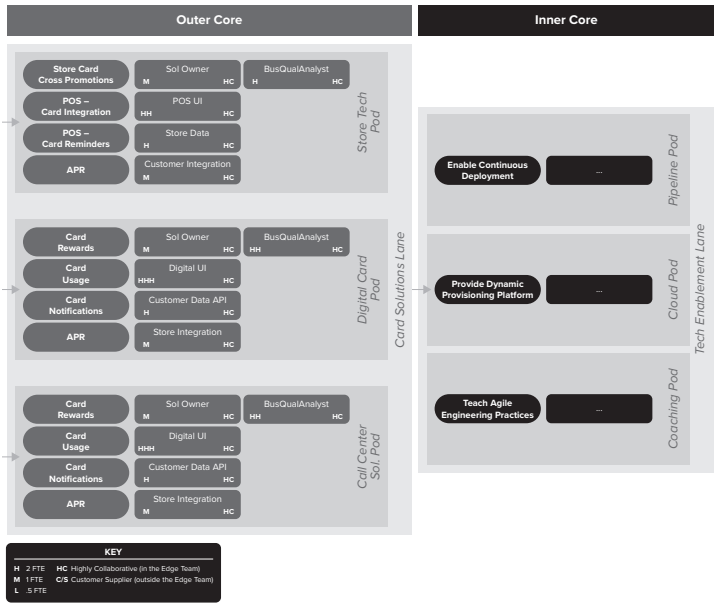
As you go through the process of defining teams and missions at the edge and various inner and outer cores, use context boundaries to define larger structures. Some of these structures will be mandated as being part of your organization's enterprise; some of these will be structures that are defined as you facilitate organizing for agility.



Context boundaries follow the same narrow-to-broader categorization that cores do. In fact, all wider context boundaries larger than a team will often have at least one core team responsible for managing that context boundary. Often there will also be other core teams allocated to the core zone defined by a wider context boundary. Again, context boundaries can fall into familiar categories from smaller to larger, paying attention to Dunbar's numbers:

- Mission/program (i.e., an Agile ecosystem)
- Business unit
- Business portfolio
- Enterprise


The Identity
The Market
The Edge
We're in the business of helping customers succeed with credit through revolutionizing banking technology
Help Clients Manage Healthy Revolving Balance
Even Revolving Balance On Portfolio
Credit Limit Increase Protection
Increase Card Engagement
Call Centers
Store Associate
Revolved Customer
Non Revolved Customer
Inactive Customers
Store Asscociates
Existing Customers
Customer > 1 year
New Customers
Customer < 1 year
Store Associates
Store Executives
Promote Offers
Save Customers
Promote Card
Carry Sustainable Balance
Consolidate Debt
Activate Card and Make First Purchase
Always Ask For Cards
Discourage Same-Day Payment
Use Card In Store
Use COS
Use Store Discount
Redeem Points
Activate Card
Use Card In Store
Ask For Card
Inform Customer of Points and Benefits
Share Card Exp. with Customer
Invest in Rewards Infrastructure
Advocate for Program Associates
Share Card Exp. with Customer
Digital Promotion Channel
Card Upgrade
Immediate Use Reminder
Digital Balance / Spend Program
Digital Activation Reminder
In Store APR Tools
Cost Of Credit Marketing
Card Marketing To Store Promotion
...
Card Marketing To Store Promotion
Product Owner
Store Execution
Teach Feasibility
Tech Analysis
Product Management
Customer Analytics
Rewards Process
Data Access
Product Owner
Deep Customer Insight
Store Execution
Customer Data
Tech Analysis
Customer Advocacy
Customer Analysis
Partner Coordination
Data Access
Product Management
Rewards Process
Associate Advocacy + Insight
Store Processes Analysis
Partner Coordination
Data Access
Store Analytics
Partner Management + Knowledge of Store Business
Deep Customer Insight
Data Access
Store and Customer Analytics
Portfolio Balance Team
Customer Balance Pod
CLIP Pod
Store Engagement Pod
Card Engagement Lane
H 2 FTE
M 1 FTE
L 5 FTE
Reduce Operational Costs
Reduce Paper Usage
Store Associates
Existing Customers
Vendors Who Send Mail For Us
Call Center
Carry Sustainable Balance
Consolidate Debt
Activate Card and Make First Purchase
Invest in Rewards Infrastructure
Advocate for Program Associates
Share Card Exp. with Customer
...
Store Processes Analysis
Operational Expert
Vendor Management
Call Center Policies / Processes Design
Paper Reduction Pod
Operational Efficiency Lane


Outer Core
Inner Core
Store Card Cross Promotions
POS – Card Integration
POS – Card Reminders
APR
Sol Owner M HC
BusQualAnalyst H HC
POS UI HH HC
Store Data H HC
Customer Integration M HC
Store Tech Pod
Card Rewards
Card Usage
Card Notifications
APR
Sol Owner M HC
BusQualAnalyst HH HC
Digital UI HHH HC
Customer Data API H HC
Store Integration M HC
Digital Card Pod
Card Rewards
Card Usage
Card Notifications
APR
Sol Owner M HC
BusQualAnalyst HH HC
Digital UI HHH HC
Customer Data API H HC
Store Integration M HC
Call Center Sol. Pod
Card Solutions Lane
Enable Continuous Deployment
...
Pipeline Pod
Provide Dynamic Provisioning Platform
...
Cloud Pod
Teach Agile Engineering Practices
...
Coaching Pod
Tech Enablement Lane
KEY
H 2 FTE
M 1 FTE
L .5 FTE
HC Highly Collaborative (in the Edge Team)
C/S Customer Supplier (outside the Edge Team)

ORGANIZING CONSTRAINT

The Core Should Travel to the Edge

Another way the core differs from the traditional organization services model is how edge teams engage with core teams. In the traditional enterprise model, the market-facing people often send work to support people. The support people do their thing and send the result back. In the case of compliance and governance, market people submit their work for review and present it to support areas, who give their blessing to proceed.

As uncertainty goes up, this increasingly becomes a recipe for failure. Work returned by support teams often does not fit the needs of the market teams. Reviews are subverted to meet internal expectations of the support teams' processes and standards, not the actual risks to the market. Support people have a hard time getting the right market context they need to effectively support edge teams.

In our new model, we flip the direction of how people and work move. Work goes to an edge team and stays there. When support is requested from a core team, someone from the core team physically (or, in recent times, virtually) picks up and goes to the market-facing team to provide the service. They become part of that team for the duration of the request. We call this person a *traveler*. Alternatively, support teams can provide platforms or knowledge to allow a market-facing team to perform the service themselves. We call this *enablement*. Core team members may do both.

Only in rare cases does allowing work to travel from an edge team to a core team and back represent a good choice to make. It may make sense for services offered by the support team to be highly commoditized and repeatable and to require very little collaboration between teams.

So, stated fully:

Avoid edge teams handing work to support teams in the

core; encourage people in core teams to travel to edge teams to collaborate on the work.

IN SUMMARY

- The new organizational model moves away from the ideas of control, compliance, power, and authority and toward organizing around market-facing teams and systematically decentralizing decision making, from a hierarchy of control and authority and toward a circular network of markets we serve and outcomes we want to achieve.
- Market value is pulled outside-in through various organizational zones: the edge zone pulls from the market, and the core zone pulls from the edge. Each inner zone serves the outer zone.
- The market contains all external market actors, the edge contains all market-facing teams, and the core contains all support and centralized functions. The identity is a boundary that contains all the artifacts used to describe who and what the organization is trying to be.
- The market includes all users, customers, support organizations, regulators, and even competitors.
- The identity contains elements that foster alignment and inspire culture. Use lightweight and inspirational artifacts, not heavy road maps or blueprints. Articulating a meaningful purpose is a very powerful exercise when it comes to establishing identity.
- Most people should work in the edge, with teams being organized around missions, outcomes, user life-cycle steps, customer segments, products, or even platforms/systems.
- The core contains all the teams that provide services and support to the edge. Unlike traditional enterprise support, the edge can choose not to use the core, and as a result the core teams must provide an excellent enterprise

experience to the edge teams.

- Use context boundaries to define subsets of an overall organization, identifying discrete parts—for instance, all the teams under a single line of business. Context boundaries may have outer core teams dedicated to supporting that context boundary.
- We can lay out a new organizing model by extending our team impact map and placing elements on the map into market, edge, and core zones respectively.

11

Organizing Patterns to Collaborate at Scale

When thinking about operating with agility at scale, we often fall into the common trap of wrapping our Agile teams with traditional organizational structure, processes, and methods. We create two distinct worlds: one where people in Agile teams try to operate with high collaboration, continual feedback, and self-organization, and one where work gets handed over to functionally oriented groups for servicing or review.

In reality, we can reimagine much of our support structure, the people in the core of our organization, in a way that is much more conducive to scaling and supporting Agile teams in an enterprise setting. We can start by populating teams in both the edge and core of our organization, using one of the team-type patterns mentioned below:

- **Agile team:** exactly what it sounds like, a stable, self-organizing, cross-functional, market-facing team
- **Enablement core:** a team of enablers who are focused on increasing capability and safety in the organization
- **Traveler pool:** a team of people that are poised to travel to work closely with other teams in a specific expertise
- **Service center:** a team in which team members work

within their own team to deliver value at the request of another team or internal stakeholder

- **Community of practice:** a group focused on sharing knowledge and experience, one that is based on employee contribution and grassroots voluntary activity

A simple way of thinking about these patterns in the context of the edge and the core is that edge teams are primarily made up of Agile teams and ecosystems of Agile teams. The core is primarily made up of teams of enablers, with some traveler pools and a few service centers. Of course, this is not a hard-and-fast rule. Your ecosystems on the edge can and will have a mixture of team pattern types, and your core will have a unique mixture of enablement cores, traveler pools, or service centers depending on their stage in the Agile journey.

These patterns can also be thought of as the stance that a team can take when servicing a different team. This idea of a team taking a particular stance is important, as an individual team will likely take different stances depending on the context. For example, a team may consider its ultimate goal to be being an enablement core, but it finds itself playing the role of a service center in order to assist low-maturity internal clients that are willing to fund the work at a premium. It is for this reason that the team-type patterns are a direct extension of the team collaboration patterns found previously in this book. For instance, a traveler pool is a dedicated pool of traveling team workers, organized into a common pool that one or more ecosystems may draw upon. Likewise, an enablement core is a home for enablers.

What are the differences between the terminology at various levels? Use the team-type patterns to describe the overall stance of the team, if there is one. Use the team collaboration pattern to denote how a team wants to collaborate with another particular team, in order to provide a particular service. In this chapter, I'll annotate the team impact map we have been following along, in order to showcase both team-type patterns and team collaboration patterns at play.

ALLOWING TEAM-TYPE PATTERNS TO EVOLVE OVER TIME

Before we get started, I think it is important to talk again about visionary pragmatism. In this chapter, I'll once again take a very hard line on industrial versus modern organizing concepts: treating people as thinking, breathing, and living beings, whose creative collaboration needs to be nurtured, is key. This organizing style is in direct contrast to the assumption that if we just gave people better instructions, they would do better work. I'll continue to be fairly transparent about the outcomes of machine management in the modern world.

But it is also important to acknowledge the very personal struggle of enabling this new way of thinking. Many of you who are reading this book are doing so because you work for a large, traditional organization. You and your organization are likely attempting to use Agile in some way but are not seeing the results. In some cases, agility has made it really obvious where things are broken, making the ideal and the reality of your situation all the more real.

In many situations, you will not have complete carte blanche to restructure all or even a significant portion of your organization. Or you may get consensus on only incremental change. You may also be facing systemic issues in your organization that make it seem impossible to reorganize using some of the more progressive ideas presented in this book. Leadership and stakeholder support may not be there.

You are not alone in wanting to or more likely needing to tread carefully. Despite all the Agile buzz, there are many more "failed" Agile transformations out there than successful ones. Successful change requires pragmatism and incrementalism. It requires an evolutionary approach to change, moving at the pace the organization can tolerate. Many of the organizing concepts provided in this book, including these team-type patterns, exist to support a gradual move to a new model. The team collaboration patterns mentioned previously and the team-type patterns described in this chapter are best thought of as stepping-stones to improved agility. In an enterprise world, presenting a means to charting the paths to agility is even more important than showcasing the solution.

The reality for many of us is we work in organizations where most

if not all support functions are monopolistic and order-centric. A lot of teams suffer from ad hoc team members or are fractured by conflicting directions among departmental bosses (department workers). Work requires navigating a maze of standards enforced by governors.

If the world were perfect, we would jump to the new world by staffing everyone, or almost everyone, in market-facing Agile teams and positioning those people as dedicated team workers. But assembling all the skills required to create value in an enterprise often spans multiple teams, so we group them into ecosystems.

Not all people have the same experience and skill, so we support them with enablers. Not all teams will be able to learn all capabilities, so we pair with them through travelers. Some work is just distracting to a team's mission or purpose, or it threatens to overwhelm their cognitive capacity, so we offer to do these things through service providers.

As we move away from legacy technology, manual processes, and arcane business models, the need for these constructs can disappear. But this will be a long road for some. In short, team patterns are a stepping-stone toward increasing agility.

When faced with what seems like hard, immovable boundaries in your organization, one solution is to simply try to "shift left": from limited ownership to shared ownership, and from shared ownership to team ownership. Then encourage others to shift left in as many situations as they can. Find opportunities of trust and safety, and work to create them.

With those high-level considerations aside, let's now consider each of the team-type patterns mentioned earlier in this chapter one by one.

THE TRAVELER POOL

A traveler pool is a dedicated group of people with similar capabilities who spend the majority of their time traveling to various Agile teams to provide assistance. This dedicated pool is made up of *traveling team workers*.

Traveler pools allow you to minimize handoffs in the face of demand that does not lend itself to stable teams. A common mistake

people make when scaling Agile teams is to structure teams based on components and capabilities, and then to allow work to be distributed across multiple teams, relying on cross-team coordination and handoffs between teams. A better alternative is to place some of your people into pools of travelers instead, encouraging *people to move to the work*, rather than moving work to people.

Imagine we have a web team that delivers features on a website that touches across a wide range of customer experiences and business products. The website needs to interface with a number of different back-end systems depending on the touch point and/or the product being exposed. When delivering digital channel website changes to the market, the digital team often needs to ensure that a supporting change is made to one or more of these supporting back-end systems. But no two website changes hit the same back-end system across any particular calendar quarter. Most of these back-end systems could be considered legacy systems, or at the very least highly proprietary.

Making changes to each of these systems is done by a separate group of people, all of whom possess very distinct and specialized knowledge both from each other and from your garden-variety object-oriented developer.

Now let's scale the problem out to a more realistic enterprise scenario. Imagine we also have a similar mobile channels team, an IVR (interactive voice response) team, and a call center support channel team. These teams are also responsible for building systems that touch on the same range of products and customer experiences that the web team does, and need to, likewise, make similar types of changes across the various back-end systems that the web team is making.

In this case, organizing people who can make changes to these back-end systems into separate traveler pools, one for each system, could be a good way to go. Ensuring a feature works end to end across multiple systems is a complex endeavor—one that is best performed without handoffs across teams. Organizing people into pools ensures that the ad hoc nature of the work does not cause collaboration to suffer.

Other good examples where I have seen traveler pools do well is with dedicated UX professionals, customer research and analytics, and larger program-level UAT expertise. At ATB, we pooled UX

professionals to be the dedicated members for a set of digital delivery teams developing the front end for a banking modernization program. In this engagement, multiple customer journeys had been established not only to deliver feature parity from a legacy perspective, but also to include enhancements resulting from customer-driven feedback. At the beginning of this program, the UX group acted as its own separate vertical, building high-fidelity design patterns that were handed off to the downstream digital delivery teams. This caused numerous problems associated with handing off work, rework, delays, and blocked teams.

As the program progressed, designers shifted their approach to moving to the work instead. Once a UX need was determined for a team, a designer would join that team's daily stand-ups, collaborate on writing user stories, and sketch designs with product owners and lead developers.

A common objection to this type of arrangement is managing the load of members of the pool. One way to prevent overburdening and alignment to value is through backlogs, throughput forecasts, and using WIP limits, just like Agile teams can. Pool members may also hold stand-ups and retrospectives, reserve WIP for technical debt, or make other internal improvements.

When asked about his own visual workflow management, Tristan Campbell, a traveling member of the UX team at ATB, recounts how he operates as an effective traveler for the various banking digital teams:

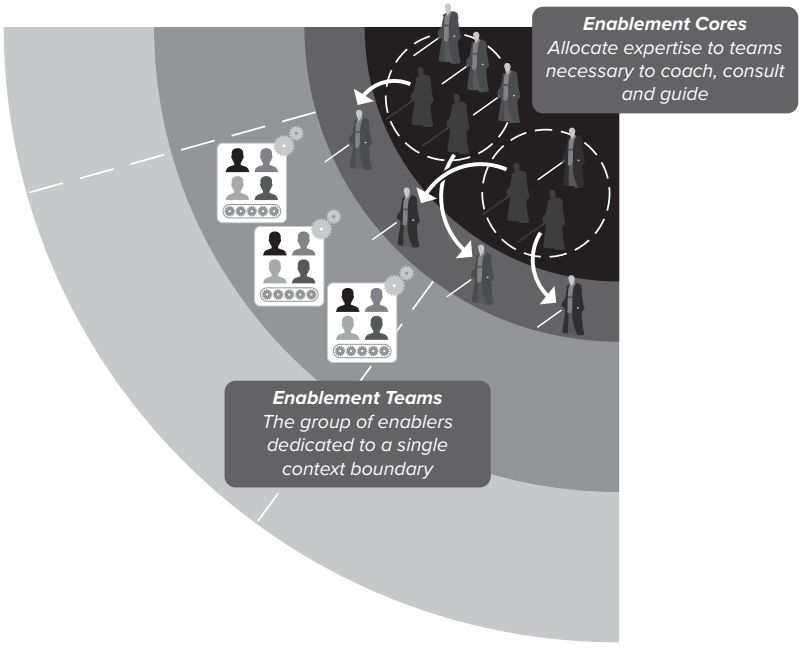
> *One of the things I do to help the UX team keep on top of all the requests coming from the digital teams is a visual backlog, using techniques like GTD [get things done] to help the UX team prioritize any upcoming work. When I personally work with teams, I try to do so in a way that minimizes the amount of handoffs between me and the digital teams I am working with. For example, I try to closely collaborate with the team on minimally detailed sketches, creating them on the spot if I can. I want to avoid separating myself from the team, only to return a finished and polished deliverable that may look great*

but misses the mark. If corner cases arise, I'll garner insights from other teams as fast as I can, including research teams, content teams, and other stakeholders, and travel back to the delivery team with an answer, enabling that delivery team to continue.

Being a member of a traveler pool is not always easy and can cause stress for the member of the pool. It takes resilience and experience for a person to be comfortable switching from team to team and being able to work in an open and collaborative manner.

THE ENABLEMENT CORE

If we remember the enabler collaboration pattern from our previous chapter on teaming, enablers are dedicated to providing capability to team members. At scale we can consider providing a home base for similarly skilled enablers, known as an enablement core. Enablers spend only a little of their time with other core members. They are often assigned to watch over a closely related set of teams, perhaps one or two ecosystems' worth of teams. They help to bring order to their sector of the galaxy, kind of like the Jedi, or the Green Lantern Corps.



An example would be the recruitment and hiring function in HR. In a traditional organization, many teams would be mandated to go through an HR representative to perform the initial screening, set pay bands, write job descriptions, and engage with talent firms. In contrast, when acting as an enabler, HR would provide a platform to make this job easier for teams. Outside of some minimal rules to ensure compliance with laws and avoidance of reputational risk, HR would not mandate its involvement in hiring, but rather it would offer help, including:

- Advice on how to effectively navigate the hiring process
- Technology that streamlines the logistics of interviewing, assessing, and finally hiring a candidate
- Sharing hiring assets (i.e., job postings) that other teams have created
- Coaching on practices related to hiring candidates

The enabler focus puts the team being enabled in the position of

being a customer of HR, as opposed to a team subservient to HR. This ensures that HR services stay healthy and suited to the needs of the team.

Getting Agile teams comfortable with using DevOps is an excellent example of how an infrastructure team can graduate from a service provider team to a pool of travelers, and then to an enablement core. They can thoughtfully increase the level of self-service that teams can perform through a combination of platform, permission, and coaching.

The DevOps team could use a highly visible Agile-style backlog to shape their demand to match the enablement model they want to achieve, including groundwork such as getting permission to provision infrastructure, aligning with vendors and providers, marketing the offering, rolling out a platform, and building a body of knowledge.

TRAVELING VERSUS ENABLING

Strictly speaking, a traveler pool is set up to make it easy for individuals to move to a team to do work in a highly collaborative way with members of the team they are traveling to: by pairing, mobbing, and so on. Likewise, enablement cores are set up to provide a home base for people to move to teams to advise, educate, coach, and sometimes validate work.

This can be a somewhat nebulous and even artificial distinction. As travelers start to engage with the same teams over and over again, those team members can start picking up the traveler's skills. It starts to make sense for the traveler team worker to take a different stance: that of an *enabler*. They can pair, mob, or otherwise collaborate with the team to facilitate the transfer of a skill set. As capability sharing starts to happen, the traveler is able to shift their approach and they move away from doing the work to guiding someone else on how to do the work.

Likewise, it can be a struggle for people providing services to other teams to move them into enablement mode, at least at first. Often the teams need to reach a certain level of maturity, as teams will likely ask or even demand services from the enablement core in ways that do not

fit this enabler pattern.

Sometimes this means working closely with teams and being hands-on, for either a short or longer period of time.

As an example, a team of Agile coaches should operate as the archetype of an enabler. Yet coaches may need to temporarily play the role of product owner or scrum master if there's a desperate need. A data enabler may be asked to help the business by creating reports rather than teaching them how to use the reporting platform. None of these activities leads to enablement of those teams, but sometimes what is practical trumps what is pure.

What both of these situations point to is that teaching others is often accomplished through a do-it, teach-it, pair-on-it, and then coach-it model. Enablement cores can think of doing service center and traveler work as the tip of an enablement spear. What distinguishes an enablement core from other team types is their intent. Did the group come together to do work or teach others to do the work? Teams helping other teams through traveling can start reserving capacity to build up a platform or knowledge artifacts to enable a self-serve approach. They build better APIs and tests around their codebase. They can then travel to customer teams to first show, then pair, then advise, and finally occasionally support.

ANTI-PATTERN

The Governor

The opposite of the enabler is the governor. The governor may bring education with them, but the focus really is on enforcement. Whereas an enabler will come with the team experience in mind, the governor is focused on team conformance. Enablement is built on the idea that if we want to alter behavior, then we need to be helpful. We need to bring capability, we need to make it easy for people, and we need to add value to the everyday lives of the people we want to influence. If behavior does not change, then we look to alter the system,

its incentives, and the way it recognizes people. The governor cares but little about such things, even if they do not admit it.

The governor uses the carrot and the stick—mostly the stick—to ensure compliance. The governor uses past problems to justify a draconian view of people and displays very little trust in them.

In most organizations even the most enlightened of leaders struggle to balance playing the role of governor versus being an enabler. Decades of cultural debt, a maze of bureaucracy, a mindset of ego and winning, a lack of safety and trust all get in the way of making a positive impact as an enabler. Sometimes our instincts or our environment will cause us to take on the role of a governor. The good news is we can always examine our circumstances in an attempt to understand how we can better lean away from the stance of the governor and lean toward the stance of an enabler.

Jonathan Galperin, a former technology director at Scotiabank, recognized the need for a set of enablers to help teams reduce development toil through pipelines, container-based deployments, and otherwise automating development tasks that acted as a source of frustration for teams in the business payments groups. Jonathan describes the initial reason the team was put together.

> *Originally our teams relied on a centralized platform team that was responsible for the build and adoption of sets of capabilities dedicated to making developers' lives easier. Things like deployment pipelines, cloud environments, monitoring, even a common-event platform were all on the menu. The promise was amazing, but the reality was not sufficient. The centralized group just didn't take a customercentric view point when it came to their solutions. The first versions of their pipelines only worked on Linux, cloud only worked on Windows; nothing worked in our context. An incomplete feature*

> *set is not such a bad thing in and of itself, but the centralized platform team seemed to be focused on mandating adoption regardless. Teams would be measured and constantly challenged on why adoption rates were so low. The centralized teams thought mandating adoption was the way to go.*

While Jonathan sorted out funding and socialized the idea, he asked Vlad Vracaric to play the part of enablement team lead and to start putting a team together.

Vlad reflects on his initial perception of what the new team needed to do differently:

> *Originally our teams relied on a centralized platform team that was responsible for the build and adoption of sets of capabilities dedicated to making developers' lives easier. Things like deployment pipelines, cloud environments, monitoring, and even a common event platform were all on the menu. The promise was amazing, but the reality was not sufficient. The centralized group just didn't take a customer-centric viewpoint when it came to their solutions. The first versions of their pipelines only worked on Linux, cloud only worked on Windows, and nothing worked in our context. An incomplete feature set is not such a bad thing in and of itself, but the centralized platform team seemed to be focused on mandating adoption regardless. Teams would be measured and constantly challenged on why adoption rates were so low. The centralized teams thought mandating adoption was the way to go.*
>
> *The customer engagement model was lacking, to say the least. The only means of communicating with the centralized platform team was through Jira tickets. It would take weeks for them to circle back on things. As you can imagine, teams were quite frustrated with the group. To make matters worse, critical items, at least to us, would just disappear off their road map*

with no notice or discussion. These weren't small slips: things promised years ago still have not come to fruition. All of this wouldn't have mattered if teams were simply permitted to seek out alternatives. It felt like we were being derailed by false promises in an attempt to keep us from pursuing our own path. The platform group had the right thinking in terms of cloud, pipelines, microservices, and so on, but they were obsessed with planned reusability and overindexed on the centralized management to achieve it.

Indeed, this mandated, monopolistic approach to services is often a default posture for many organizations looking to modernize their engineering stack. Centralized organizations build a set of product features but without being able to take into account the context of their customers. Serving the needs of an entire enterprise with a one-size-fits-all solution is an exercise in futility. Rather than acknowledge this reality, centralized teams shift to mandating that teams use their product, regardless of usefulness. Instead of enablement, we get government in its worst possible form.

Vlad discusses how his team, the platform engineering team, took a dramatically different approach:

We started with the mindset that we existed to serve teams and to enable them. We spent a lot of time early on with the teams, listening to them and learning about their context. What tools were they using? What products? What did their tech stack look like? Where were they suffering? Where was the toil? We then addressed low-hanging fruit. If teams were having trouble integrating code, we would put something in place to make that easier. If versioning was an issue, we would train people up on better Git usage. As an example, one of the teams was working with an off-the-shelf package that was configured through Excel. This made version management impossible through a typical Git-based source code repo. To solve their issue, we wrote a tool

that would convert Excel binaries to XML on check-in and support the merging of different people's work. The dev team loved the result because it saved them countless hours and let them get closer to a typical developer experience.

We took this idea of delivering an amazing team experience to the fullest we could. We spent the majority of the time working where the dev teams worked. We traveled to pair with them, as well as to provide support. We developed real relationships with the teams. We made heavy use of the same kind of visual management using Kanban that the dev teams did. We even took the extra step of linking work on our boards to the work on their boards. Some commented on the overhead, but I thought the extra transparency was worth it; it allowed us to navigate the context of many different teams while staying focused on the goals that mattered to them.

According to Vlad, no one was mandated to make use of their services:

We constantly marketed our wares, which actually wasn't all that hard. We took part in tech lab showcases that ran biweekly in the payments group, and we shared an ROI model that showed the payback period of their investment in the adoption of our solutions. We even estimated bespoke features using cost of delay based on developer time saved. Finally, we measured the usage of our features and used that to guide our focus, even when it was painful to do so. For example, a payments-specific testing automation tool we built is not getting any real usage from our teams, so we now need to determine how to respond, including the possibility of abandoning the effort.

Vlad feels that being an enabler in an enablement core team is not

always easy, but the results are worth it:

> *Part of our responsibility was to act as a go-between for the dev teams and the centralized platform team. While this was valuable for the team, it was quite frustrating for us. In some cases, we might have taken the concept of a good team experience a little too far and taken on more ownership of implementing automation than we should have. In some cases, we had to work extra hard to unwind a bit of learned dependency on us, and get teams to invest in the effort required to implement something that would make their lives easier. Overall, taking the enabler approach has been worth it. We've helped teams solve problems by focusing on providing solutions that were fit for purpose and not mandated from the top. Helping people is always useful. Mandating that someone take your help, never helps.*

THE SERVICE CENTER

Like the traveler pool and the enablement core, the service center is the home base of a team of service providers that are providing a similar service.

Unlike the enabler and the traveler, the service provider works within the service center with other service providers that have similar capabilities and does not work closely with the team requesting their services.

Service centers are often about cost reduction; they are cost centers. This only works when work is considered to be highly repeatable and does not vary greatly from request to request. In these cases, economies of scale can be applied to share demand, reduce cost, and increase efficiency.

Service centers handle requests using an approach that involves a discrete request to a provider and then waiting for a response, typically using some form of order ticketing mechanism. A work ticket is

placed on a queue by a requesting team, and a service provider working in the service center picks the ticket up off the queue, performs the work required, and then moves on to the next ticket.

Service centers will strive to provide the best service they can for the teams they support, providing clear service expectations and working closely to collaborate with requesters to ensure outcomes are met. But beyond this, service centers do not pair or otherwise team on the work with other teams.

In general, you want to avoid using service centers to complete complex work if you want to avoid rework and reduced efficiency, which destroys the very reason you want to use a service provider in the first place. It is better to use service providers where the request is straightforward and even simple.

Most back-office functions are organized into service centers. They should not be. The majority of functions provided by HR, legal, compliance, security, IT, and so on would be an order of magnitude more effective if they were set up as enablement cores or, at a minimum, traveler pools.

You will find some situations where the service provider is the best way to go. Activities that require manual processing or manual intervention may need the economies of scale offered by this model. As software becomes more pervasive, and we continue to automate our workflow, the need for such service providers should continue to diminish.

The service provider pattern provides us an option when it doesn't make sense to negotiate a more hands-on, collaborative solution. It is also the option we turn to when people are unwilling, at least for now, to abandon their existing organizing structure. Anyone responsible for defining a more Agile organizing structure needs to describe the option where we are keeping a certain part of our organization rooted in the familiar. It is better to represent these parts of the organization as they are now, as opposed to some idealized future.

Service centers can increase their internal agility by adopting practices such as backlogs and visual management systems to introduce transparency and achieve better customer engagement, with an eye toward considering different collaboration patterns.

Speaking from my own experience, early in my Agile change consulting days, I was a lead consultant at a larger firm. Senior-level consultants were assigned admin assistants to help them with the logistics of meetings, booking firm facilities, and other support duties. In the early days, an admin would be more or less permanently assigned to one of three senior-level folks. While this arrangement assured an intimacy and familiarity that was appreciated by both parties, services provided by admins would vary. Quality also varied greatly, and it was difficult to manage workloads. Some admins were overloaded, and some had a good deal of spare time. There were limited opportunities for admin assistants to learn from each other.

When the firm announced that all admin function would move to a service-based function, where a request would be sent to a common email queue, I was pretty skeptical. It appeared that we were moving in the direction of creating handoffs for the sake of cost reduction and maximizing admin utilization. The devil, it appears, is in the details. Requests were, in fact, routed to a subteam of two or three admins based on the service line of the consultant. Within that team, the request first went to the admin that had the most familiarity with the

requesting consultant. If that admin was too busy, one of the other admins would take over, getting any necessary context, such as quirks or preferences of the requester. Administrators all sat together and were able to collaborate on more-complicated support requests, mentor each other, and otherwise operate as a team. I was blown away by how well the service provider approach worked. It preserved the best of both worlds in terms of both familiarity and always being able to count on someone being available to help. The service became more expansive, lead times continued to drop, and I grew to appreciate that there was a team of people I could turn to. They were all sitting together, so when I had a more complex ask, like helping to put together a miniconference for some clients, I could literally visit their workspace to see whether and how they could support. I was constantly and pleasantly surprised at the kind of support a team can provide versus an individual.

SERVICE CENTER FROM THE OUTSIDE/ AGILE ON THE INSIDE

A team may seem like a service center from the perspective of the customer team making the request. The servicing team provides a common service using a request-response approach to the rest of the organization. Internally the team also appears to be composed of *dedicated team members.* They are cross-functional, self-organizing members of what appears to be a classic Agile team, at least when looked at from the perspective of that team. Most Agile teams in IT departments fall into this category. But all too frequently we see few to no team members having direct market contact; work comes to them from various organizational stakeholders or market proxy roles instead. While the team may exhibit great internal agility, it is very challenging for them to exhibit great market agility.

The IT Agile team, like other service centers that are tasked with complex market-oriented work, tries to make up for this lack of market contact through highly collaborative and frequent shaping and review sessions with their organizational stakeholders. For many organizations, this service center from the outside/Agile on the inside approach

is a solid step forward. That being said, if your team does not have direct market contact, it is not really an edge team in charge of market value but is part of the core providing support.

As previously stated, Agile teams can mitigate the impact of being engaged as a service center by collaborating closely with their organizational stakeholders who have this market access. Another approach is to use the previously mentioned concept of a *traveling intent owner*. A traveling intent owner is an organizational stakeholder of the team that spends enough time with the team to bring their market connectivity with them. Again, this is a reverse kind of traveling: the person that needs skills/capabilities, in order to deliver value, travels to the team who will deliver the value. They collaborate closely with the service center team. The effect is that the service center team operates much more closely to the spirit of an edge team, especially as more business stakeholders spend more and more time with the service center.

ANTI-PATTERN

The Order Taker

All too often we see internal support and servicing teams take on the stance of an order taker. Order takers focus on completing tasks, with scant thought to customer service. Thoughtful service requires consideration and care. Is it easy for the customer to initiate a request, give feedback, and get updates on the request? How long is the request expected to take? What happens to the customer once the request is complete? What risk or errors are likely to happen? How can we effectively

communicate with the requester? How are problems with requests resolved so they don't happen again? Does the service meet the needs of the requester? Order takers rarely ask these questions, and never address them to a satisfactory level. Work fed to order takers can disappear into a black box, sometimes never coming out the other side.

ANTI-PATTERN

The Monopolist

An important attribute that the enablement core, traveler pool, and service center all have in common is the principle of choice. A team chooses to be enabled, they choose to bring in help, they choose to send work over the fence. We know that without choice, experience suffers, and it suffers catastrophically. If a team has no choice but to rely on another team, then that team is a monopolist. Sometimes we can get good service from a monopoly, by hiring amazing talent that really gets the idea of TeamEx (short for team enterprise experience). But this is not sustainable: invariably all monopolies become self-serving and more about protecting their interests than the interests of their clients. This is true outside of the enterprise when dealing with external market actors, and it is no different inside your enterprise.

It's worth noting that most internal teams will face implicit pressure to use internal support functions even when not mandated to do so. A bit of customer centricity will go a long way in convincing many teams to make use of these internal support functions versus going outside of their organization or rolling out their own.

COMMUNITY OF PRACTICE

A community of practice (CoP) is best thought of as a connected network of individuals who benefit by collaborating on a shared capability or common set of skills or methods. A CoP is often thought of as virtual grouping, one where membership comes from interested contributors from other teams. The purpose of a CoP overlaps with that of enablement cores. CoPs rely on learning through consensus and grassroots involvement. A CoP may have some small funding and an extremely small staff to foster the management of the community, such as running community events or operating an area for the community to share knowledge. But unlike enablement cores, the majority of the work is a result of the community, from people who are actively applying their knowledge in the field.

MAPPING TEAM PATTERNS TO TEAM TOPOLOGY PATTERNS

Again, it is worth pointing out that Matthew Skelton and Manuel Pais have some very similar concepts in their book *Team Topologies*. Some of the team patterns in this book map reasonably well to the four fundamental team topologies mentioned by Skelton and Pais.

Agile team: Skelton and Pais call this type of team a *stream-aligned team*, meaning the team is aligned to delivering a stream of value, typically aligned to a business domain or organizational capability.

Enablement core: Amazingly, Skelton and Pais include the concept of an *enablement team*, which is almost identical to this book's concept of an enablement core. The enablement team is composed of specialists in a given technical (or product) domain, and they help bridge the capability gaps of existing teams. Skelton and Pais also have a separate topology type for *platform teams*. In their words, platform teams "provide internal services to reduce the cognitive load that would be required from stream-aligned teams to develop underlying services." I consider platform providers to really be an advanced form

of enablement and therefore consider both enablement teams and platform teams to be variants of enablement cores.

IN SUMMARY

- Organizations embarking on an Agile transformation should tread carefully when introducing team-type patterns; incrementalism and pragmatism are key to successful adoption.
- Travelers contained in traveler pools help minimize handoffs and encourage people to move to the work, rather than moving work to people.
- Enablers from the enablement core are dedicated to providing capability to team members; they can pair, mob, or otherwise collaborate with the team to facilitate the transfer of a skill set.
- A service provider pattern, harnessed within service centers, is often best used when requests are straightforward and when it doesn't make sense to negotiate a more hands-on, collaborative solution.
- The enablement core, traveler pool, and service center all possess the principle of choice. When team A relies on the decisions made and takes instruction from team B, then that makes team B a monopolist, which is an anti-pattern.

12

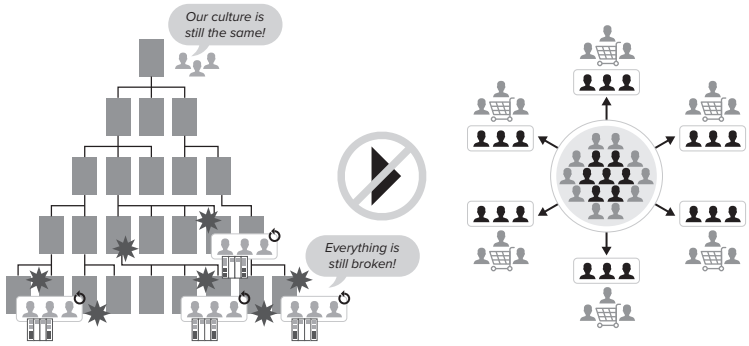
The Blade and the Handle

Many of you reading this book work in large, traditionally structured enterprises. While your team(s) have likely been through some form of Agile adoption, it is very likely that, after a period of momentum, your Agile journey has stalled. If you have solid support from positional leadership and you have attentive stakeholders, you may have successfully put an "Agile bubble" in place—a kernel of self-organization, trust, and transparency within the larger industrial machine. But an Agile bubble is often assaulted from all sides:

- Positional leaders are occupied with fending off the toxic culture.
- Teams are constantly frustrated by arcane and dysfunctional processes.
- Flow is impeded by dependencies on teams external to your bubble.
- Functional groups fight to protect their own turf.
- The recognition system rewards individuals, not teams.
- Budgeting, planning, and forecasting are locked in an annual cycle of artificial precision.
- The organization starts mandating everyone to use a standardized Agile methodology and DevOps tool set.

- Many in management struggle to exhibit the structured skills required to get started.

Adopting team-level Agile practices can only take us so far if we want to deal with these issues. To make progress, we need to reexamine our structure and organize around an entirely different set of principles.



PUSHING PAST THE INERTIA FACING SYSTEMATIC CHANGES

But changing toward a decentralized, market-driven organization requires a significant shift in mindset for many who have grown up in a more traditional organization. We often encounter a mixture of concern, skepticism, and even outright disbelief—"this could never work here."

To truly change our organizing structure, we need to fight inertia and resistance from defenders of the status quo. People often have a lot vested in their position, who they report to, the existing structure, and how they are currently working. All it takes is for one actor with some organizational power to impede progress. And in the traditional organization, these actors are everywhere.

So how do we move forward? How do we move to organizing models like the ones suggested so far? How do we get to the idea of edge teams with direct market access? How do we move support teams

toward the idea of a core typified by excellent experience and choice?

There is no easy answer, at least not one that I am aware of. It comes down to increasing the intensity of the dialogue. We need to discuss the impact that the current structure has on our people and on our customers. We need to challenge the pervasive mindset that permeates the organization at every opportunity. More importantly, we need to empower others to have the same discussion. We need to create opportunities to change the conversation.

There are various change methods and change tools you can use to facilitate this discussion. But don't let any change framework fool you: shifting mindsets and attitudes won't happen unless you can present a compelling narrative. Good change approaches help structure that narrative, but the hard work is still yours.

A METAPHOR FOR AGILITY: THE BLADE AND THE HANDLE

Let's start by framing our context in terms of the edge and the core. How well does your team, ecosystem, or larger organizational group align with the design constraints expressed by your new mental model for getting organized? How well do your Agile teams fit into the idea of an edge team? Do they have the permission and capability to navigate their market with precision? How well do your support groups fit into the idea of a core team? Are they enabling market-facing teams to own the decisions and activities required to sustainably deliver value for their markets? When our teams and organization don't line up, what kind of problems are being created? Where is a lack of ability to navigate the market and limited ownership creating the most impediments for teams? Which of these impediments can you address in the near term, and which impediments require more thought, more alignment, and more buy-in?

Going with the idea that agility can be thought of as being a function of market precision and good control, we can use the metaphor of a kitchen knife for Agile teams, ecosystems, and larger organizing structures. We want everyone within the enterprise to sharpen their **blade** and increase their control of the **handle**. Teams do this

by examining their permissions, accountabilities, capabilities, and collaborations with the rest of the organization to understand what barriers can be removed and how best to remove them. While solutions to these problems are often context specific, the barriers to a dull blade and a poor handle are often common.

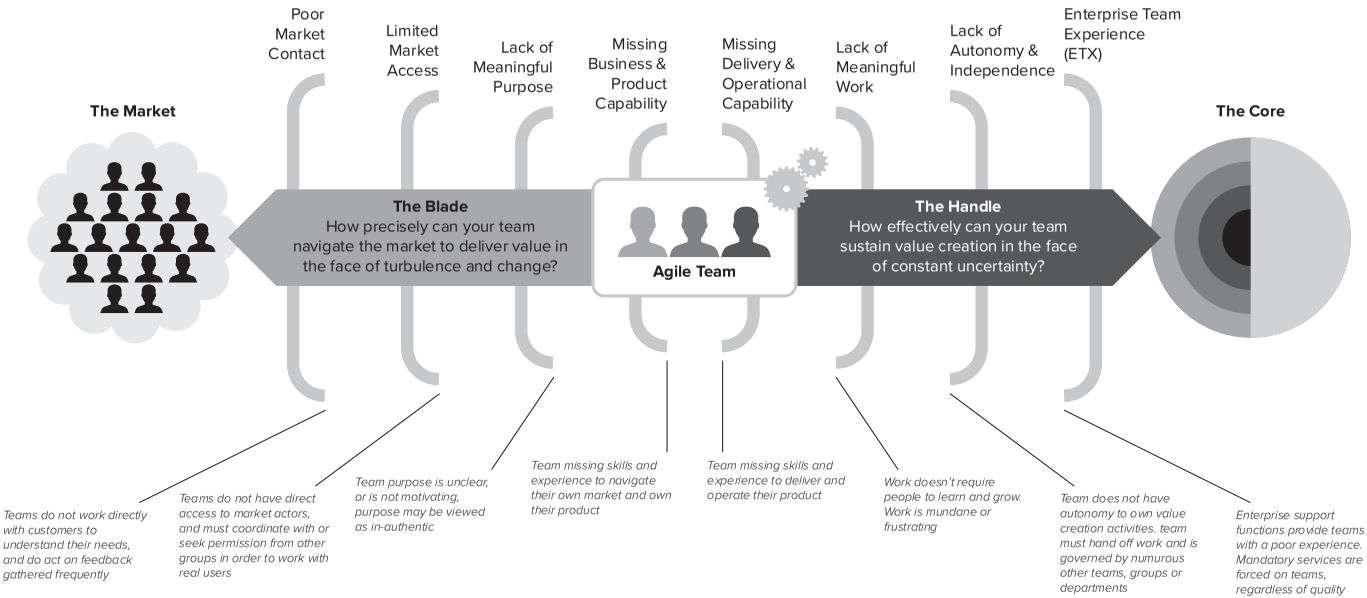
A Dull Blade

- **Poor market contact:** Teams do not work directly with customers to understand their needs, and do not act on feedback gathered frequently.
- **Limited market access:** Teams do not have direct access to market actors and only engage with proxies and/or must seek permission from other groups in order to work with real users.
- **Lack of meaningful purpose:** Team purpose is unclear or is not motivating; purpose may be viewed as inauthentic, an exercise in propaganda.
- **Missing business and product capability:** Teams are missing the skills and experience to navigate their market and own their product.

A Poor Handle

- **Poor enterprise experience (EnX):** Enterprise support functions and governance groups provide teams with a poor experience. Mandatory services and mercurial governance are forced on teams, regardless of quality or value.
- **Lack of autonomy and independence:** Teams do not have autonomy to own value-creation activities. Teams must hand off work and are governed by numerous other teams, groups, or departments.
- **Lack of meaningful work:** Work doesn't require people to learn and grow. Work is mundane or frustrating.

- **Missing delivery and operational capability:** Teams are missing the skills and experience to deliver and operate their product.


Poor Market Contact
Limited Market Access
Lack of Meaningful Purpose
Missing Business & Product Capability
Missing Delivery & Operational Capability
Lack of Meaningful Work
Lack of Autonomy & Independence
Enterprise Team Experience (ETX)
The Market
The Core
The Blade
How precisely can your team navigate the market to deliver value in the face of turbulence and change?
Agile Team
The Handle
How effectively can your team sustain value creation in the face of constant uncertainty?
Teams do not work directly with customers to understand their needs, and do act on feedback gathered frequently
Teams do not have direct access to market actors, and must coordinate with or seek permission from other groups in order to work with real users
Team purpose is unclear, or is not motivating, purpose may be viewed as in-authentic
Team missing skills and experience to navigate their own market and own their product
Team missing skills and experience to deliver and operate their product
Work doesn't require people to learn and grow. Work is mundane or frustrating
Team does not have autonomy to own value creation activities. team must hand off work and is governed by numurous other teams, groups or departments
Enterprise support functions provide teams with a poor experience. Mandatory services are forced on teams, regardless of quality

GETTING THE DIALOGUE STARTED

The simplest way to introduce this frame on existing Agile teams is to bring a list of open-ended questions to the team that help articulate where a dull edge and a poor handle are impeding them.

Here are some starter questions, but, again, tailor what you ask to your needs and context.

The Blade

- How precisely can your team navigate the market to deliver value in the face of turbulence and change? In other words, how sharp is the blade of your team?
- Is the purpose of the team clear and well understood? Does your team feel like it has a purpose that is authentic? Is your team's purpose reflected in the choices, actions, and decisions taken by the team as well as stakeholders and leaders?
- Does your team have direct and frequent access to real users and other market actors? How many other groups are you required to coordinate with or seek permission from in order to work closely with real users?
- How does your team gather feedback from your users? How many users do you gather feedback from? How often does feedback result in a change in the team's direction?
- Does your team feel it has the skills and experiences it requires to work with market actors and other stakeholders to shape, prioritize, guide, and validate the creation of value? Is your team able to easily acquire, as needed, new skills required to own their product?

The Handle

- How effectively can your team sustainably own value creation in the face of constant uncertainty? In other words,

how effective is the team's handle?

- How often does the team do the work that it loves to do versus the work it dislikes? How often are there opportunities for the team to solve complex problems (growth, creativity, and passion)?
- How much autonomy does your team have to plan, build, manage, and own? How easy is it for your team to closely collaborate with the team it depends on to deliver value?
- How many other groups, departments, or teams must your team hand off work to in order to create customer value? How closely connected (organization, operating model, organizational chart, culture) is your team to the teams it depends on?
- What are the support groups your teams are forced to use, and which support groups can the team choose to use or not use (including both internal and vendor groups)? How would your team members describe the customer experience when dealing with these support groups?

A simple workshop can be facilitated, perhaps during a retrospective, to allow team members to answer these questions and highlight areas that they would like to improve. Above is an example taken from a team-level session, where the team identified various barriers that were impeding their agility.

SCALING OUT THE DISCUSSION

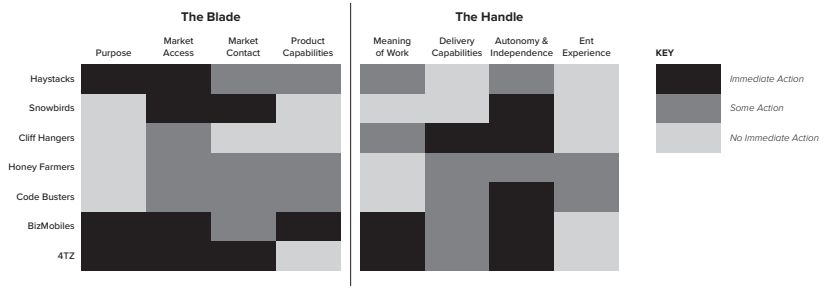
It can also be useful to get a wider perspective on the structural impediments facing a large number of teams, perhaps across one or more ecosystems. Surveys can complement workshops to synthesize input from a larger audience, which is important if we want to demonstrate that we have the buy-in for larger changes.

ATB was in the midst of delivering on a large multiteam program dedicated to modernizing the digital banking experience for its business customers and allowing the company to move off their existing legacy platform. The scrum masters on the engagement felt that, while the development of new features was going well, there were some systematic and structural barriers that were interfering with the program's ability to meet the actual business objectives of the program.

The scrum masters wanted to highlight opportunities where they could sharpen the edge and solidify the handle for teams, and for the business banking program as a whole. They wanted to make sure that any assessment they did reflected the perspectives of people working in the teams, as opposed to their or management's opinion. The scrum masters asked all team members to answer a brief survey, then to participate in a workshop to discuss their answers, identifying structural impediments and improvements as they went. They then consolidated the output of the workshops into common themes that fed a program-wide improvement backlog.

Throughout this process the scrum master team had to continually remind themselves that they were not running an assessment. Their opinions on the improvement opportunities were to have no bearing. Rather, the whole point was to synthesize where the team members wanted to improve. The result was a view of where the teams wanted to initiate change. This was important to the scrum master group: if

any structural changes were going to stick, they believed that people on the ground needed to believe in those changes and to drive them.



When summarizing the feedback from the teams, the resulting view was not so much a summary of maturity as it was a heat map of impeding activity. The program management and team members collectively understood where people were going to spend time to improve the way they worked. The ATB business banking program ended up with a healthy backlog of improvements, which as of this writing are being worked on by both teams and leaders in the organization. Here are a few of them:

- People in organizations will often have very different perspectives on the **impact and breadth of their organization's purpose**, with many focused on delivering like-for-like features as opposed to user outcomes. So we want to align on what our **purpose truly is and how it relates to our customers**.
- We have very **few opportunities to interact directly with the real customers**; all user access is limited and filtered through the design and research team. So we want to engage much more closely with the research groups to increase our touch points with customers, including **user testing and observing real users** interact with released features.
- We have an inconsistent architecture approach, opaque road-mapping process, and poor documentation, which

is causing **significant frustration and interfering with our ability to work in an autonomous way**. So we want to establish a **common upstream flow of work for all squads** that involves our squad members in early ideation, planning, and solutioning activities.
- We have **inconsistent access to supporting environments** and the teams that support them (e.g., SAP, DevOps, AES), so we want to **train our team members on how to use these technologies** in the ATB context and reduce our reliance on external teams.
- We believe that we are prevented from expanding our skill sets to product design and architecture because they **fall outside of our individual job descriptions**. So we want to change the way we **define jobs, roles, and titles to support a more flexible, learning-based approach**.

The scrum masters managed the delivery of improvements by facilitating the attendance of an improvement Kanban with a simple "prepare, introduce, and learn" flow. Cadences were held at both the team and program levels. Working on the same board made it easier for team members to participate in improvements that crossed teams.

LEAN CHANGE

The approach to improving organizational agility discussed above is an example of how to approach change in an open way. It represents the idea of change co-creation, introducing change in small iterations, and validating the change frequently. These are principles of what I call the lean change method. *The Lean Change Method* is a book I wrote that showcases the use of inclusive and Agile approaches to facilitate change. In a nutshell, lean change asks you to:

- Co-create your change with all stakeholders being impacted by the change.
- Validate your change by introducing change in small increments.

Lean change also asks you to operate your change using Agile methods: change canvases, change impact maps, change Kanbans, change stand-ups, and so on. *The Lean Change Method* goes into great details and examples, but if you have some Agile experience and a bit of imagination, you are more than capable of coming up with your own exact approach. The above approaches are all valid examples of a customized lean change.

The lean change method will ask you to work very closely with executives, management, and team members to explore problems and express solutions as a hypothesis. We then introduce changes as experiments. We want to see whether our intervention is worth continuing or should be altered or outright abandoned. We start validation with a focus on shared observation and dialogue. Did the people impacted feel like their lives have improved? As teams mature and we scale out the change, we can start considering more quantitative metrics. Did the change reduce the impact of delay? Is lead time going down?

A lot of this is old hat to classically trained lean folk, but a big difference from the classic lean material is that lean change takes a page out of lean start-up principles. The emphasis is on lighter-weight tools, starting out with qualitative experiments, and only moving to quantitative measures after we are confident that we are shifting the mindset through plain old observation.

IN SUMMARY

- Many Agile transformations get stuck, whereby surrounding functions, centralized groups, and so on aren't participating in structural change. The result is an Agile bubble surrounded by a legacy organization.
- We can make progress toward a more Agile organizing structure at the team, ecosystem, or any level by intensifying the conversation around increasing market precision and strengthening value-creation activities in the face of constant change.
- Using a kitchen knife for inspiration, we can think of improving agility as the act of sharpening our edge and

increasing the effectiveness of our handle on value-creation activities.

- Sharpening the edge requires that we explore and make improvements related to meaningfulness of purpose, market contact, market access, and our product and business ownership capability.
- Increasing effectiveness of the handle requires we explore and make improvements to meaningfulness of work, autonomy and independence, our enterprise experience, and delivery and operational capability.
- We can facilitate the dialogue required to improve the way we are organized through informal discussion during retrospectives, polling perspectives using surveys, and other methods that are open and collaborative.
- The exact technique is less important than following an open and inclusive approach to change that allows us to introduce incremental changes that we can validate quickly and iterate on if necessary. The lean change method offers a useful framework for this type of change approach.

13

The Software Organization

As organizations consider how to provide better value in the face of complexity, they often turn to software to deliver more and more services to their customers. It's a plain fact that software is eating the world. Digitizing customer experiences, automating back-office processes, marshaling information into insight and even real knowledge—all are driven by software. Software allows us to operate in a world of automated service that can interconnect with each other to bring about an era of mass customization. Paradoxically, the era of the knowledge worker and the human-based organization is being made possible by our advancements in the machine.

Because of the pervasiveness of software systems in the modern organization, no serious book on organizational design can be considered complete without a real look at the synergistic effects that organizational design and software system architecture have on each other. Too many organizations ignore the impact that organizational structure has on software design and vice versa, to their detriment. Many organizations that have truly scaled for agility have done so with scant attention to published Agile methodologies. Think the Amazons, Googles, and Netflixes of the world. What these organizations have done is treat their software and their organizations as a single system. They organize around the software they want to create, with the

knowledge that they will write software according to how they are organized.

This chapter will lay out some of the theory and approaches you can take to align the two systems of software and organizational structure. Much of this chapter is taken from my experience in applying concepts taken from the excellent book by Eric Evans titled *Domain-Driven Design: Tackling Complexity in the Heart of Software*. The book is known for its excellent treatise on how to build software that mirrors the language of the business. But the book is more than that. *Domain-Driven Design* discusses how models of software relate, integrate, and depend on the teams that build and run them.

When I originally planned to write this chapter, I expected to have to do a fair amount of work taking many of my experiences and approaches and refining them to something a little more concise and shareable. Then I came across a recently published book, *Team Topologies: Organizing Business and Technology Teams for Fast Flow*, by Matthew Skelton and Manuel Pais. This book has some glaring similarities to mine in that it covers team and interaction patterns to scale with agility. Amazingly, we have come to some very similar solutions, and we even happen to use the exact same terminology in a few places (we both use the enablement concept). There are some differences, of course, but conceptually I feel the books are very aligned.

Team Topologies does an amazing job of laying out how to apply the concept of domains to align team structure with software system structure. Rather than reinventing the wheel, I'll be leveraging some of their work. A big shout-out to the great work Skelton and Pais have done to move this body of work is warranted. It certainly made writing this chapter easier.

THE PERIL OF IGNORING CONWAY'S LAW

Let's start with a discussion on Conway's Law. Way back in 1960, Mel Conway observed that when separate groups in a larger organization worked together on larger systems, they would tend to break up systems they were working on into parts so that each group could work on their own piece as independently as possible. Or to quote Conway:

"Any organization that designs a system will inevitably produce a design whose structure is a copy of the organization's communication structure."

When Mel came up with this law, he defined a system more broadly than just software systems, but it is in the world of software that Conway's Law has gained real prominence. Open-source advocate Eric S. Raymond and software celebrity James Coplien are among the many that have observed the impact of Conway's Law on software systems. James Coplien stated in his 2004 book, *Organizational Patterns of Agile Software Development*, that when organizing structures (for instance, teams, departments, or subdivisions) do not closely reflect a product's essentials, and the relationships between organizing structures do not reflect the relationships between product parts, then trouble ensues . . .

Over the past two decades of experience in the software world, I have witnessed the dysfunction that takes place when we ignore the impact of Conway's Law. I've seen countless teams developing redundant code in the same system that provides the exact same business functionality, but inconsistently so. I've seen teams dramatically increasing code complexity through multiple translation layers design so that each team can avoid "infecting" their code with another team's code. I've come across dependency "magnets" that result in multiple teams being unable to develop or test any functionality without involvement from every other team.

Skelton and Pais, in their book *Team Topologies*, rightly point out that when you design a dependency between two parts of a software system, you also need to think about the ideal communication path for the two teams in the organization that build or own those parts of the software system. When the overall organization is small, this is not much of a concern, but as you scale the number of teams that are building software, you create real fragility when team boundaries and module boundaries are inconsistent with each other. Team communication grows exponentially, and the need for management-style coordination, and even control, goes up.

Perhaps one of the biggest sins we see when organizations don't pay attention to Conway's Law is the creation of the monolith. When communication is poor across multiple teams working on the same

software system, we end up with overlapping modules, poor abstractions, leaky interfaces, and dependency magnets. Despite the best intention or well-thought-out design, over time our software solutions devolve into a mess.

When multiple teams do duplicative yet often incompatible work, they lose the ability to make decisions without input from every other team. Or worse, they make decisions on their own that negatively impact other teams. They lose the ability to change the system without impacting everyone else. Monoliths aren't all accidental: indeed, an industrial organizational mindset, with its emphasis on standards, instructions, and conformity, naturally leads to what Skelton and Pais describe as monolithic thinking, which is a:

> *"one size fits all" thinking for teams that leads to unnecessary restrictions on technology and implementation approaches between teams. Standardizing everything in order to minimize variation simplifies the management oversight of engineering teams, but it comes at a high premium.*

If we accept that the wrong organizing structure will have severe impacts on our software architecture, which in turn will impede our organization from effectively delivering value using that software, then it becomes fairly obvious that you can't design an effective organizing structure without a deep understanding of software systems. If you leave organizational design in the hands of your HR department, then HR will also (accidentally) decide your software architecture. You want the opposite to occur: you want your architects, senior developers, or whatever you call your people with deep software system knowledge to intentionally define teams around the software architecture you have or want to have.

This approach, dubbed the "Inverse Conway Maneuver" by Skelton and Pais, is an intentional act of reconfiguring the team intercommunication according to the boundaries we want between the different parts of our systems. The frequency, intimacy, and bandwidth of communication paths shape the kinds of solutions we devise. When one or two teams closely collaborate on a system, we are more likely to

see chatty module boundaries (distinct software modules that send too many fine-grained methods to each other) or perhaps a system with few distinct partitions. When multiple teams "intentionally" operate more independently, we are more likely to see the team integrate through well-defined APIs or to be loosely coupled through publishing and consuming events. When we have a mismatch between teams and separate system parts, we get a mess. But the idea here, according to Skelton and Pais, is to use the synergy between the system and the organization "to our strategic advantage. If we want to discourage certain kinds of designs . . . we can reshape the organization to avoid this."

THE PERILS OF FOCUSING ON SOFTWARE INTERNALS

Historically, we have seen an emphasis on focusing software architecture on system internals, decoupling the technology layers as much as possible in an attempt to create a layered, or "tiered," architecture. Systems were divided into a minimum of three loosely coupled layers: the presentation, business logic, and data layers. Often additional integration, orchestration, or workflow layers were added to the mix. In an attempt to follow Conway's Law, we would create separate teams to develop and possibly maintain the distinct software for each specific layer of the system. We ended up with a presentation layer team, a business logic team, and a data team. Sometimes more than that.

This approach seemed sound to many at the time, but really it is the product of the inward-focused thinking we see from an industrial mindset. We saw many of the same problems that we do from a functionally siloed industrial organization. This is because most system software changes typically impact multiple layers of a software solution. For instance, if we need to expand our shopping cart functionality to accommodate discounts from external marketing campaigns, that will impact the presentation layer, the business logic layer, and the data layer. Members from all three teams will need to closely communicate to make sure that the changes that take place across the system are consistent and do not contradict each other.

In contrast, when we ask the teams to make changes to how the solution manages product inventory, we would want that change to

have low to little impact on adjacent functions, such as how we manage customers or perform billing. But in our current organizational design, concepts such as customers, billing, and inventory are all worked on by the same teams. Whether intentional or not, we are more likely to have chatty interfaces and poor partitioning across these functions, and unforeseen dependencies between these concepts are likely.

Stephanie Dimovski was one of the product owners leading an ambitious initiative to modernize the point-of-sales system being used by in-store pharmacists. She described the problems she faced this way:

> *Inconsistency in prescription, disbursement, and patient data had been shown to be costing actual lives, so it was important to increase timeliness and accuracy of information across all pharmacies in all provinces across Canada. We had an immediate focus on building real-time integration between the store points of sale and the Alberta health patient and prescription tracking system. Because of the focus on safety, our engagement had some pretty tight timelines. Previous teams on this program had suffered from some previous hiccups, late delivery, missed scope, things like that. That put this engagement in a situation where we were asked to deliver something in a matter of months.*
>
> *Early on in the engagement, the decision was made to restructure the teams in our "mission" [their word for ecosystem] according to architectural layers so that we could focus engineers on individual components that they would be experts in. While the product owners felt there was a risk that teams would lose the context of the feature as it traveled across the front-end and back-end teams, we agreed to try it out with an open mind.*
>
> *In the end, the approach did not provide the gains that leadership had hoped for. Teams did indeed lose context as a feature traveled across teams and many handoffs occurred. Teams did not have ownership of a feature and therefore didn't need to be accountable to the end state of that feature. Really it was a bit of a*

disaster. The deliver-by-contract approach may have sounded good on paper, but it required so much up-front design to understand who could work on what. It quickly became a nightmare to know who was responsible for which pieces. No one had ownership of when an end-to-end piece was going to be done. There was no tangible business "thing" you were building. And our throughput plummeted, to about seven stories a sprint from an average of sixteen.

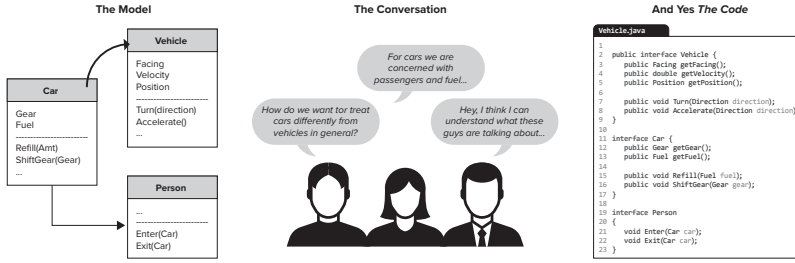
INTRODUCING DOMAIN-DRIVEN DESIGN

Domain-driven design (DDD for short) is based on the idea that the structure of our software systems should reflect the way business experts think about their business. Software should mirror business concepts and subjects, business activities, business state, and business rules. Infrastructural and cross-cutting concerns should be treated as they are, which is as system internals.

In a domain-driven design approach, you partition modules first by business concepts, known as business domains. Things like customers, billing, orders, and fulfillment are all potential ways to partition the system. As systems scale, the rate and reasons to introduce change are more likely to be different across these business domains than across software layers. For instance, changes to your supplier code will likely change the way your fulfillment code works but have little impact on the code you use to acquire new customers.

In contrast, UI, workflow, and data access are considered to be internals, and while consistency may be desirable for these pieces, it is far more important to have well-managed interfaces between the business-oriented concepts than to obsess over internal consistency between all code belonging to a software layer.

Domain-driven design tackles solution complexity because it focuses on the areas of the system that are most subject to change—in other words, the business domains. It places emphasis on capturing the knowledge that is most likely to require experts outside of the field of software.



A team is participating in domain-driven design when they model, communicate, and develop according to the core business-domain logic of a solution and do so using a common domain language, known as a *ubiquitous language*. Software specialists work intimately with domain experts to capture the domain model, using domain terms, and embed the domain terminology into their code.

The benefits are systems designed to flex and evolve in a way that reflects how the business will change. Major changes will create major system changes; minor business changes will require minor system changes.

ORGANIZING AROUND DOMAINS

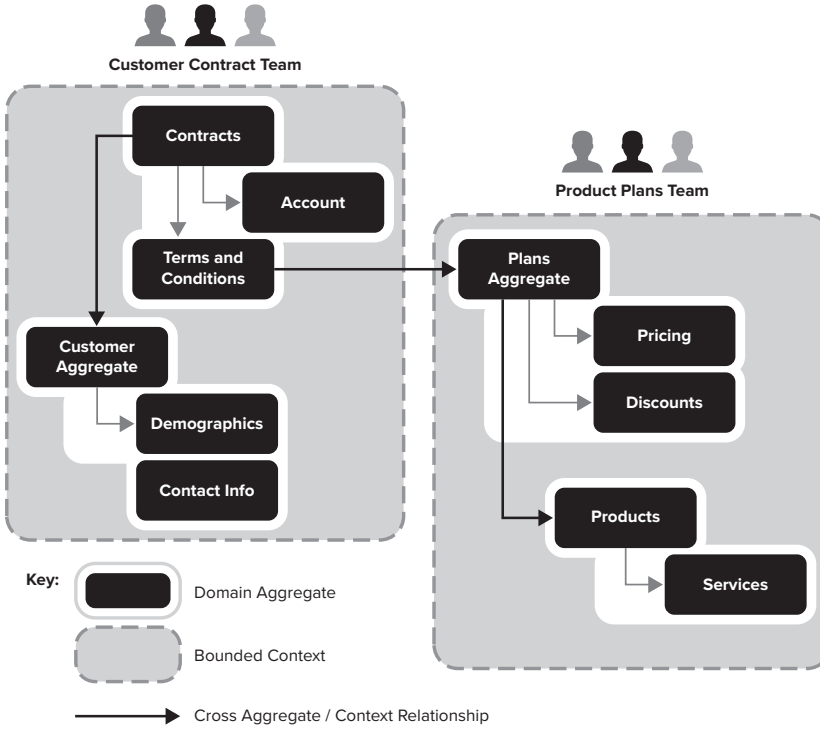
If we accept that structuring our system architecture according to business domains is the preferred approach, then we can in turn start thinking about structuring teams according to discrete *domain aggregates*. A domain aggregate can be thought of as a set of domain concepts that are very tightly interrelated with each other. It can be a customer and their demographics or a shopping cart and all of the independent order items in it. Domain aggregates are made up of domain entities and all of their invariants—in other words, dependent domain objects that are retrieved together, updated together, and otherwise change together. The Inverse Conway Maneuver is most effectively applied when we think about partitioning software into separate domain aggregates. This allows us to organize teams so that they can be focused on the software required for a limited number of these

separate business domain aggregates.

In larger systems, DDD recommends that we place our domain aggregates into separate *bounded contexts*. A bounded context places a hard boundary around a domain aggregate, or possibly a small number of highly dependent domain aggregates. In a nutshell, we define bounded contexts according to distinct but possibly related domains of knowledge. A bounded context could be set up around a payments domain aggregate, a customer, accounts and agreements, or charging and billing. Bounded contexts can nest, with larger ones representing entire systems and a smaller one representing a single domain service.

We may then refine our bounded contexts by grouping smaller domain aggregates into larger composite domains and dividing larger domains into separate but related component domains. We explore how different domain aggregates relate to each other and what the dependencies are between them. This allows us to suggest bounded contexts that can serve as the basis for a team structure that minimizes the number and impact of dependencies that cross teams.

Structuring teams around bounded contexts is a good way to achieve an *acceptable* level of independence when multiple teams are working on a large-scale system. While we may want teams and ecosystems to be truly independent, complete independence is often an impossibility. In the real world, we often have to settle for *decoupling the dependencies* across teams and ecosystems of teams. Defining bounded contexts around decoupled domain aggregates allows us to achieve this.



Defining bounded contexts for teams and the aggregates they work on

Ruth Nielsen, another product owner on the initiative, talked about how reorganizing according to bounded contexts helped to turn their program around:

> *We continued to advocate for owning how we wanted to structure ourselves. We showed the impact of the development-by-contract approach to our throughput. Eventually we were given the reins back.*
>
> *Right away the other product owners and I agreed that we couldn't focus on getting components done in isolation; if we just built the button, or just the message, and so on, we wouldn't have a product. Instead we structured our work so we could incrementally show how user activity was translated to the Alberta*

> *Information System and back again. This is what the province cared about, and this is what our stakeholders cared about. There was no point showing something to the province if it wasn't working end to end. So we organized around building end-to-end functionality. We organized around user feedback.*
>
> *We ended up forming teams around distinct but related domains, each within their own bounded context. Each team was expected to work cross-functionally, delivering an end-to-end slice of functionality that demonstrated a complete journey from in-store POS to the provincial disbursement information system and back again.*

Ruth, Stephanie, and the rest of the leads across the teams collaborated on how to go back to a cross-functional team structure like they had before, but with a subtle yet distinct difference. Team members would form into smaller feature cells around discrete bounded contexts.

Ruth discussed how the approach kept things manageable:

> *Teams were tightly defined and people could work within a small group, according to a tight set of features within a particular bounded context. Team members got to learn the entire stack without being overwhelmed by the entirety of what the program was doing. This made developing end-to-end flows a lot easier to do. After a few weeks of setup time, the teams began to gel and throughput started to increase to almost thirty stories a sprint!*

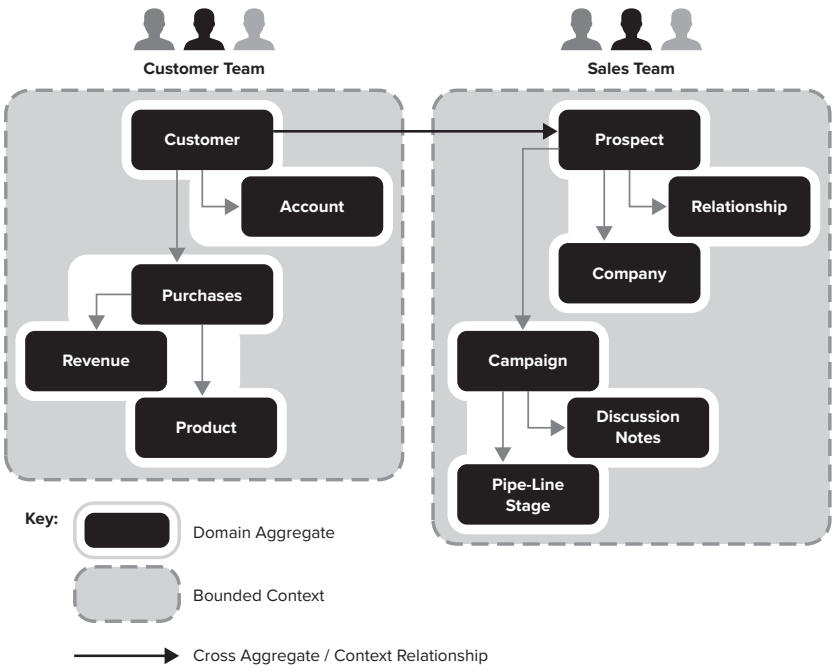
Stephanie added:

> *Allowing teams to form around specific domain-based contexts increased the understanding of the value the team was providing, toward end-to-end value being returned. There was no going back to a siloed way of*

working. We gave the teams ownership of something tangible, something to be proud of; we were successful because their work had meaning.

REPRESENTING THE SOFTWARE ARTIFACTS OF YOUR BUSINESS

An important aspect of a bounded context is that it represents a *hard boundary*, one where all the aggregates within the boundary are part of a common consistent context. They represent a common view that is only considered "true" within the boundary. Different representations of the same concept are allowed, and even to be expected, across bounded contexts. What this means is that the software used to represent these concepts will be consistent only within a boundary. Thus, we need to be explicit about dependencies across bounded contexts, as well as in the definition, translation, and integration required when crossing bounded contexts. This is important when thinking not only from a software perspective but also, more importantly, from the perspective of what these concepts mean to the development teams and business experts that work within a particular bounded context.

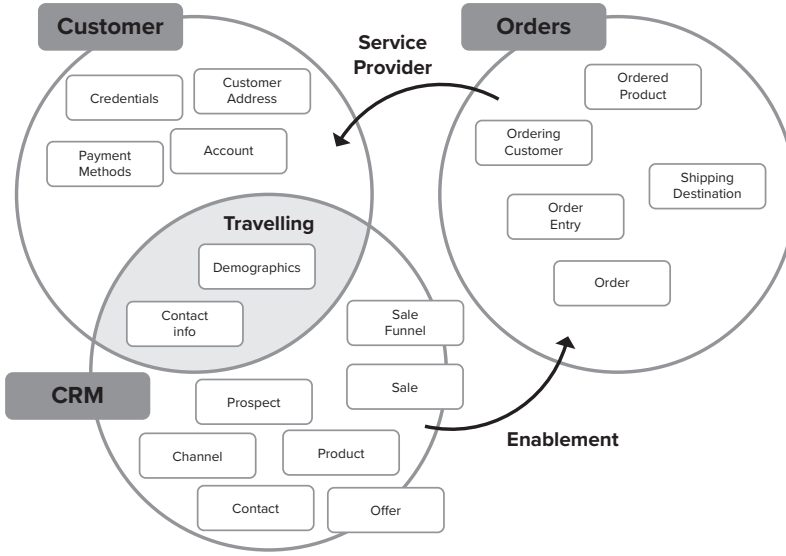


Two views of the same thing

Within a bounded context, we ask software and business experts to examine each domain aggregate and agree on what *services* (or operations) are being provided by the aggregate, what domain entities are being managed, what orchestration happens, and how these domain entities within each aggregate can be retrieved, created, and stored into a *domain repository*. Concepts like what is considered a valid state or various other rules, workflow, and conditions are also explored. The idea is to create a programmatic API that is expressed purely in business concepts and business terms, such that if a business expert can see past the programmatic syntax, they could see a living expression of their business.

The relationship, dependencies, and interactions among dependencies between bounded contexts can be illustrated by using a *bounded context map*. A bounded context map can show not only the

relationship between the system boundaries but also the interaction model being used by the teams that own the interaction model.



In the above diagram, where the CRM and customer contexts overlap, we have what Eric Evans calls a *shared kernel*, whereby impacts to elements in the kernel impact both teams and require intimate collaboration and shared code ownership; daily integration is common, and perhaps a single repo or pipeline is in use. In the case of this example, the teams agree that they need to act as *traveling team workers* and come together to work on demographics and contact info. This approach is appropriate when two teams are both working on interrelated domain concepts for the first time or when major changes to both teams' representation of these domain concepts are being made. Workable solutions will require trial and error across both teams; intense collaboration will be required. Shared kernels can also come from accidental architecture, dependency magnets, and other forms of tight coupling. Where possible, try to refactor your solution to minimize this form of coupling.

Many bounded contexts will be separate but still have some form of dependency with another bounded context. Expressed as arrows

connecting the boundaries, these dependencies in this case are more decoupled than the shared-kernel variety, and changes to ordering code may only require minimal changes to customer code, which is already expressed as a well-formed, stable customer API. In this case, the customer team can act as a *service provider* to the orders team. The customer team can gather requirements from the orders team, preferably in the form of acceptance tests, implement the changes needed, and review/demo the results with the orders team. Eric Evans calls this a *customer/supplier relationship*.

Finally, we will have cases where a bounded context will require no change to the API or underlying code in order to support the needs of the team/context with the dependency. In this situation, the supplying team is really responsible for acting as an enabler. In our current example, the orders team is being enabled by the CRM team as they work on the sales domain. The CRM team is responsible for providing a well-tested API that is easy and obvious to use and for providing great support to the orders team when they have issues or questions. Eric Evans calls this integration approach the *published language* method.

Personally speaking, the idea that context boundaries and bounded context maps are an effective way to organize boundaries across multiple teams on a large program is something I have been working with clients on for over fifteen years at the time of this writing. Demarcating context boundaries based on distinct domains in a way that respects Dunbar's numbers gives us an approach that lets us decouple our organization into separate pieces that can free them to experience change at different speeds from each other, while at the same time bringing people together based on their ability to collaborate. Using context boundaries to group teams, code, and other knowledge together was my very first stab at trying to think of more organizing structure that enabled rather than stifled agility.

IDENTIFYING DOMAIN AGGREGATES AND CONTEXT BOUNDARIES

For the most part, aggregates and boundaries can be identified by

simply taking an inventory of the most important subject areas of your business. Then, we can ask how coupled these concepts are from a business perspective. Working closely with business subject matter experts, it is relatively easy to come up with such a list. General concepts like customer, account, billing, and product come to mind, but make sure to find topics that are specific to the business you are writing software for. For instance, in the world of pharmacies you would likely have domains around prescriptions, disbursements, patients, and medical providers. The point here is many technologies try to build solutions around abstract models and then struggle to adapt these to the specifics of a problem space. I recommend you model your solution and organize your teams around concrete business concepts that matter to your stakeholders who directly serve the market. Let abstract concepts evolve as you apply domain concepts to multiple areas.

There are also some additional ways you can think about carving out domain aggregates into separate bounded contexts; often there are various approaches we can take to identify the subtleties in a domain that justify partitioning a portion of the overall model into a separate bounded context:

- **Regulatory and compliance:** The requirements imposed on a solution can often be housed into their own bounded context, even if those requirements span a number of other, adjacent domains. For instance, requirements that come from "know your customer"–related regulations are often consistent across many banking products and lines of business. The rules, structure, validations, and other business logic related to complying to these regulations could be served up by common microservice infrastructure and injected into various parts of the solution. A know-your-customer team would be responsible for a KYC API and for collaborating and supporting other development teams.
- **Transaction rates:** When transactions occur at dramatically different rates from each other in a system, chances are you have identified a distinct domain boundary that can be decoupled into separate contexts. Customers

and their accounts tend to be acquired, updated, or retired a lot less frequently than customers purchasing new products. For telecom or financial products, we see transactional activities taking place much more frequently than we see products being purchased. Inventory replenishment happens at its own cadence, less often than customer transactions but typically more frequently than changes to products being offered. When we compare how many times data will need to change over time across a solution, it becomes clearer which domains should be more tightly coupled and which areas can be more loosely coupled.

- **Market actors:** The needs of various customer segments, user personas, and so on that make up your market actors may be different enough that you define distinct bounded contexts based on these specific needs. For instance, in the world of banking, the workflow and the applications used to serve global small business customers can often be quite different from the kinds of large-scale processing applications used to serve large commercial business customers. The flows can be quite distinct, the requirements for reversibility and auditability can vary, and so on. Concierge clients may also have a different feature set, risk profile, and so on than other customers, and thus be managed as a separate bounded context.
- **Domain life cycle state:** Our view of a domain can be quite different based on where a domain entity can be in its life cycle. We care about very different things when we are managing a lead versus managing the transaction of a real customer. Having distinct bounded contexts for both leads and customers can often make sense. Likewise, we may care about different aspects of a product when managing its assembly versus marketing the product on our website to consumers. Again, separate bounded contexts for each of these product perspectives can make sense.

LIMITING DOMAIN COMPLEXITY PER TEAM

A natural question to ask is how much domain complexity a team can handle. Skelton and Pais discuss using domain complexity as a guide to determine this number. In my experience, this is a function of both complexity and change being made to the domain, so I would amend their guidance to include both complexity and degree of change, to determine the number of domains a team can work on at a time:

- One complex domain or one domain undergoing a high degree of change
- Two to three simple domains, or one domain going through two or three simple changes
- Avoid a single team owning more than one complicated domain or having to make substantial changes to more than one domain at a time.

Again, these are good rules of thumb, and context matters. The point here is to make sure that we keep cognitive load on teams manageable while at the same time encouraging the teams to grow their skills and knowledge. Looking at the team's flow of work may give you warning signs that the team is overwhelmed, as will asking teams how they feel about the amount of domain complexity that they own.

THE MODERN WORLD OF MICROSERVICES

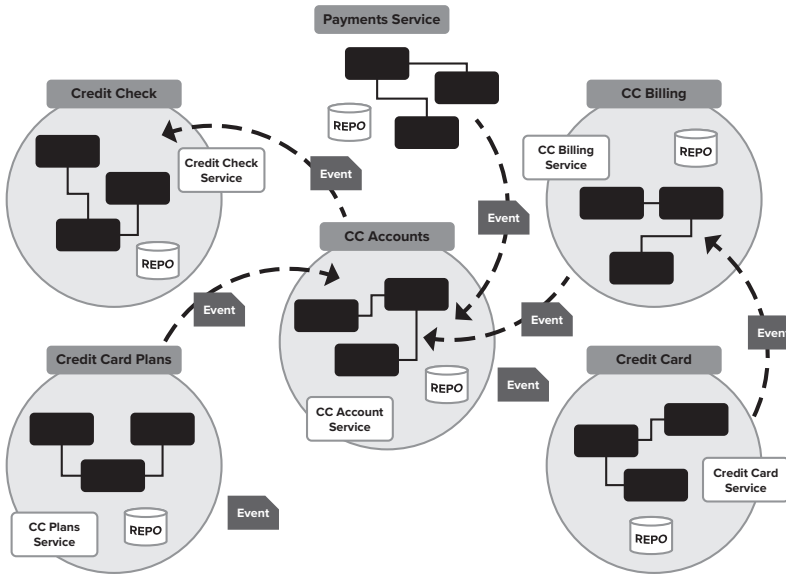
As mentioned previously, the thinking behind structuring teams so they can develop distinct and decoupled bounded contexts has had an influence on the direction and adoption of new architectural paradigms like container-based provisioning, microservices, and No-Sql.

Using modern development and deployment technologies, we can take each one of our bounded contexts and portray it as a cohesive codebase that is represented through a well-formed API and deployed independently from the rest of the system using container-based technology. We deploy a separate container for each bounded context, and each container includes the business logic, data access code, and

even—thanks to No-Sql technologies—the database itself. Some organizations are even experimenting with deploying formatting "naked" UI code, so that application consumers have the option to skin a reference domain–based UI and hook that into their larger front end. We can then maximize decoupling across each bounded context through event-based coordination, perhaps using the latest event-streaming platforms.

This book is in no way a text on software architecture, and there are others who have delved far deeper into how to apply modern software approaches than I have. But I'll admit to an unbridled passion for the subject. And it is worth pointing out that microservices, containerization, event streaming, and No-Sql have been inspired by concepts such as domain aggregates and bounded contexts.

When we guide the use of these more modern delivery technologies with domain-driven design and the Inverse Conway Maneuver, we end up with a system that is decoupled in a way that allows teams to work independently and intelligently on a larger system. We avoid the dangers of the monolith and the dependency magnet. We are able to deliver and operate these distinct parts of our system according to natural seams within the system, which Skelton and Pais call "fracture planes." Each fracture plane can be supported in a way that matches their unique needs. Each fracture plane will have its own rate of change, its performance requirements and usage load, its security needs, or other risk factors. Teams can thus fit their approach to the profile of the fracture plane they are supporting. We get the best of both worlds, a well-reasoned architecture that provides teams with the freedom they need to deliver based on their context. We get to scale with agility.



Context boundaries structure teams as well as the work they do

It is worth being wary of the fact that when many traditional organizations look at how they are approaching their deployment of technologies like cloud and microservices, they do not appear to bring much of this thinking along with it.

Many of those responsible for the adoption of these approaches forget that these technologies are there to enable better self-organization and increase the autonomy of teams. They mandate usage according to detailed standards, gate cloud access through specialist teams, and otherwise use these technologies to exert unnecessary control on teams. Worse, they do not consider the relationship between domain-bounded contexts, microservice topology, and team structure. The result is often a worse mess than the monolith they started with, a fragmented architecture and organizational mess we call the *distributed monolith*. Those leading the adoption of new technology would do well to remember that the successful adoption of these tools without increasing team autonomy is really not much of a success at all.

WHEN WE NEED TO ORGANIZE AROUND TECHNOLOGY INSTEAD

While I have emphasized the importance of taking a business domain–based approach to partitioning systems and defining different bounded contexts, there are situations where bounded contexts should be based on technology considerations. This is often necessary when a solution relies on a number of packaged software and integrating legacy systems. In these cases, teams are forced to work with system representations of various domains, and these representations will vary from system to system, hence the need to define bounded contexts from a systems perspective. Delivery flow will be considerably different across systems, with changes involving older technology being rather slow because of more manual testing, onerous deployments, poor documentation, and aging codebases. Both package products and legacy systems often do not have good support for modern engineering tools and practices like test-driven development and continuous integration and deployment.

Finally, the ecosystem of tools (IDEs, build tools, testing tools, etc.) around such technology tends to behave and feel very different across various proprietary packages and legacy systems.

All of these factors make a compelling case to structure teams around systems rather than domain aggregates. Using this approach, we define bounded contexts for each system, and perhaps for discrete modules within these systems. We can use a bounded context map to identify the way we want teams to collaborate where domain entities are represented in multiple systems and need to work on the integration required to ensure the overall solution works in a consistent way. It is important to note that this approach can suffer when the systems under change are being stood up for the first time or undergoing a very large amount of change. Dependencies across teams can make delivery an incredibly complicated affair. In these cases, it may be better to start with a single team made up of one or two specialists from each system and have them lay out a common understanding of how requirements will cascade across the various systems. Once this

common understanding is established, it then becomes much more feasible to scale out toward system-based teams.

IN SUMMARY

- Scaling organizational agility in many cases means treating the organization and the software that supports it as a single, integrated system.
- Conway's Law, implying that the dysfunction of your software will match the dysfunction of your organizational structure, has been ignored by many an enterprise, resulting in monolithic and fragmented systems that have frozen organizations' ability to deliver value quickly.
- We can use the Inverse Conway Maneuver to deliberately partition our software solution around the communication pathways we want our teams to have, in other words deliberately aligning software architecture to organizational design.
- Domain-driven design asks us to design software solutions that intimately reflect business domain concepts and domain language, according to how domain experts think and speak about the business domain.
- Partitioning systems into domain aggregates and bounded contexts allows us to define parts of the system that change together and that are naturally decoupled from the remainder of the system, as opposed to partitioning the system based on technology internals, which results in system parts that are highly dependent on each other.
- We can structure teams around each bounded context, allowing multiple teams to work in a reasonably autonomous way on a single system.
- A modern software stack using microservices, containerization, No-Sql, event streaming, and so on enables teams

to increase their ability to work within a bounded context while minimizing the impact to other teams working within a separate bounded context.

- When working on legacy systems and package software solutions, the diversity in technology may require us to set up teams and their bounded contexts around these systems instead of domain-based bounded contexts.

Conclusion

OK, you made it to the end. Congratulations on making it this far!

In an attempt to be comprehensive, this book goes over a lot of material in a great amount of detail, using lots of real-world examples.

The key message I would like readers to take from this is that organizing at scale does not have to equate to the factory model of work. We can look to alternative models pioneered by more-modern organizations. But more importantly we can look at the progress, albeit sometimes belated, that many classic old-school organizations have made over the past decade and think about how to accelerate that progress.

If you remember nothing else from this book, remember the following principles:

1. **Organize Around Teams**

Key to designing the new organization is the idea that teams matter more than departments. Cross-functional teams. Self-managing teams. Teams that are comprised of people who are experts in their respective fields yet also able to pinch-hit and swap roles with other team members. Teams that are accountable and empowered to achieve real outcomes. We need to design for teams that are capable and empowered to get the job done with a minimum of interference. A key ingredient to effective teams is the ability of people *to team* with each other. Another is putting the right support structures in place to enable those teams to be successful.

2. **Organize Through Markets**

When you want to scale the concept of teams across your enterprise,

you don't do it by adding in a slew of mandated centralized services. You don't add in layers of mandated governance, coordination, and bureaucracy. You connect teams through markets. First, you place the majority of your people into teams that have direct contact with real customers, accountable to real customer outcomes. Second, you make the use of any enterprise services voluntary for your market-facing teams. In this way you bring the benefit of market mechanics inside your organization and avoid the cruft, senselessness, and even immorality that comes with command-side decision making.

3. **Organize for Change**

For teams to be effective we need the team roster to exhibit stability. But the real world is not always so accommodating. Change happens. Whatever team and organizational structure we come up with will be correct only for a short period of time. The market will change. And we will need to adapt. We want stability, but we also want our teams to be organized in a way that they can achieve market outcomes. And market outcomes change, more and more these days. These are forces that counteract each other. So we need to empower every knowledge worker with the insight and understanding required to form together into teams that create value, to organize in a way that eliminates handoffs. This constant reshifting takes effort and time but is critical if we want to avoid stagnating back into a world where people have to cross organizational boundaries to deliver value.

4. **Organize Around Social and Domain Boundaries**

Teams of five to eight is often the golden rule in most Agile circles. But if we want to scale with agility, we need to consider how to achieve social density at larger scales. We can use *Dunbar's numbers* to group people according to different social outcomes (5, 15, 35, 150). For technology-driven businesses (i.e., almost all businesses), we will take a page from *domain-driven design* and align our organization and solution architecture according to domains that can be delivered and managed by independent, full stack teams. With this approach, leaders influence outcomes using social density, not command and control.

Challenge yourself on which of these you can apply to your team, to your department or line of business, and even to your entire

organization. Ask yourself how far you can move the needle on these principles, and bring others in your organization into the discussion.

Look for areas where you have the space to move toward agility. The good news is that the next generation of workers is getting increasingly vocal about demanding a new way of organizing, and the next generation of leaders is increasingly dissatisfied with the status quo.

The tides of time are on our side, so get out there and be a force for change!

About the Author

Jeff Anderson is the president of Agile by Design, a global services firm that helps companies achieve business agility at scale through a thoughtful, design-driven approach to organizational evolution. Over the past decade, Anderson has led many enterprise-scale agile transformations by providing a mix of coaching, operating-model design, and change-management services.

Anderson frequently blogs about and presents on lean and agile adoption, and he is the author of *The Lean Change Method*, which guides organizational change through the application of lean startup techniques. His mission in life is to help knowledge workers be awesome at what they do.

The way we think about organizations is wrong. **Completely wrong.**

The traditional business—one made up of departments, steered by management, and operated based on a fixed playbook—is based on concepts that are more than a hundred years old. As a result, most companies are built to meet the needs of the industrial age, not our current age of uncertainty.

But it doesn't have to be this way. *Organizing Toward Agility* offers a new path forward: a team-centric, iterative approach to delivering value using customer feedback and continuous experimentation. This book will help you set up and operate new organizing structures, whether you want to make a larger organizational change or move your business forward incrementally.

With his practical tools and real-world examples, agile expert Jeff Anderson builds on decades of experience in enterprise-scale business transformation to help you shift the way you work. A must-read handbook for change agents of all types, *Organizing Toward Agility* will teach you how to design, grow, and operate organizations that can thrive in today's complex world.

Nonfiction * 6" x 9" * 260 pages * Tentative Retail Price: $14.99 (paperback) *Additional Formats: $8.99 (e-book) * Paperback ISBN: 978-1-7780930-0-5 * E-book IBSN: 978-1-7780930-1-2 * Tentative Publication Date: October 2022

FOR MEDIA INQUIRIES, PLEASE CONTACT:

Adria Batt, Girl Friday Productions
adria@girlfridayproductions.com

MARKETING CAMPAIGN:

- Targeted outreach to book reviewers, business interest media, and influencers
- eGalleys distributed through NetGalley
- Goodreads giveaway around publication
- Amazon advertising
- Enhanced content on Amazon product page
- E-book down pricing promotion after publication