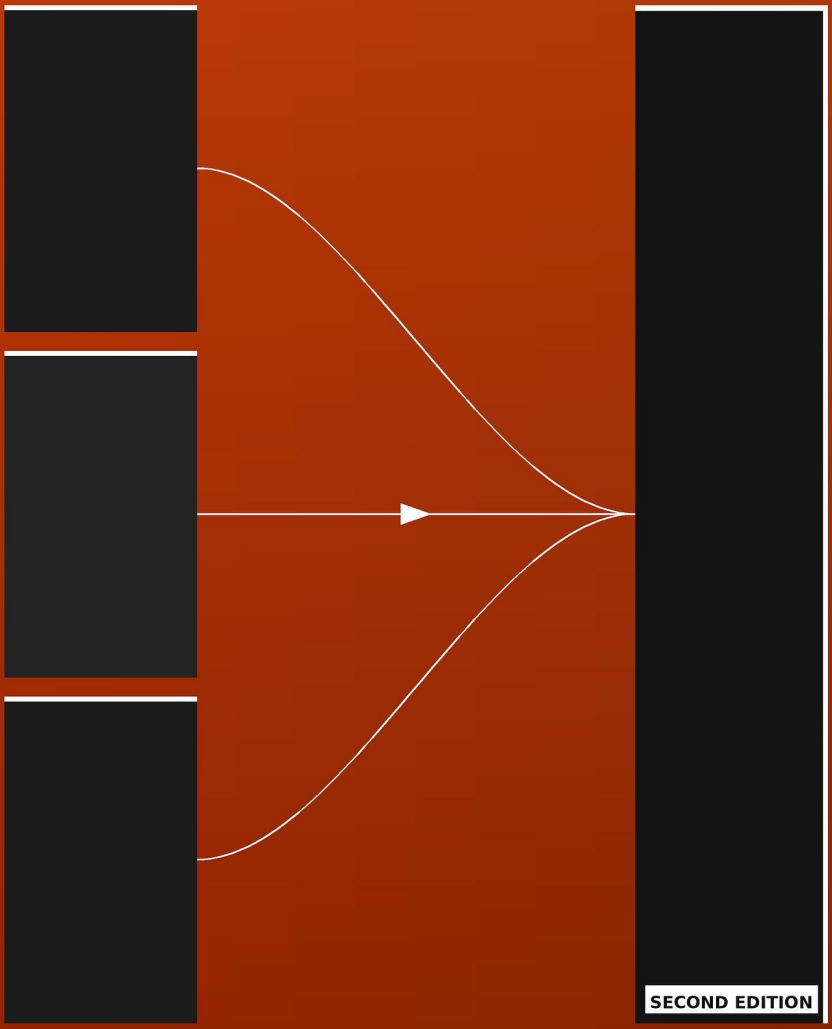
UNITED AGILITY

Uniting Product and Business Agility



A primer for the new. A mirror for the practiced.

DAVID BORZILLO
Better Ways of Working

DAVID BORZILLO

United Agility

Second edition

This book was professionally typeset on Reedsy.
Find out more at reedsy.com

Contents

Introduction

Most agile transformations don't fail because teams can't learn Scrum. They fail because the teams that learn Scrum never truly align with the business around them.

I have spent over a decade inside organizations trying to close that gap. I've been the developer going through a transformation who didn't understand why leadership kept changing direction. I've been the coach brought in to fix teams that were running perfect sprints and delivering nothing anyone cared about. I've seen organizations spend years and significant money implementing agile frameworks, only to find themselves with faster teams pointed in the wrong direction.

That gap — between product teams doing agile and the business they're supposed to serve — is what this book is about.

The word "united" in the title is deliberate. Coordination isn't enough, and alignment on a slide isn't enough. What separates a group of agile teams from a truly agile organization is whether the people doing the work and the people setting the direction have genuinely come together around shared outcomes. When that happens, something changes. Teams stop optimizing for their own velocity and start caring about whether the business is winning. Leaders stop treating agile as a delivery mechanism and start treating it as a way of thinking. The gap closes.

This book is a primer if you're new to agile, and a mirror if you're not. If you've never encountered agile before, you'll find

enough here to understand the landscape, choose a starting point, and avoid the most common mistakes. If you've been practicing agile for years, you'll find questions worth sitting with — about whether your teams are truly united with the business, or just running well in isolation.

A note on scope: this book won't go deep on tools, and it won't give you a step-by-step implementation playbook. What it will give you is a clear picture of what good looks like, why the pieces need to work together, and how to start thinking about the gap in your own organization.

One more thing. This book is itself an agile product: an MVP. It contains enough to be useful, not everything that could be said. My goal is that you finish this, put it down with two or three ideas you want to pursue, and come back to me with what you learn. That feedback will make the next version better. That's how agile is supposed to work.

Let me show you what that looks like before we talk about how to get there.

1

What Agile Looks Like

The meeting started at 9:00 and ended at 9:12.

Twelve minutes. Seven people. No slides, no status report, no manager talking while the team listened. Just a team — some in a conference room, a few faces in small squares on a screen — moving through their work with the quiet efficiency of people who have done this enough times that the structure has become invisible. Whether they were looking at a physical board on the wall or a shared screen, it didn't matter. They were reading their sprint.

One developer flagged a delay on an integration that was blocking two other items. The team didn't spiral into a troubleshooting session. They simply asked: is this something we can resolve ourselves, or does it need to go to leadership? Leadership, agreed the team. It was captured, assigned, and within the hour it was in front of the person who could actually do something about it.

The retrospective at the end of the sprint had the same quality — honest, focused, without blame. What worked? What didn't? What are we going to try differently next time? Not a perfunctory

exercise but a genuine pause to get better.

I have worked with a lot of teams over the years. When I saw this one, I recognized immediately what I was looking at. Not a team doing agile. A team that had become agile.

This book is about how that happens — and how to begin.

That kind of team didn't exist in 1995. Understanding why tells you everything about what agile actually is.

Origin of Agile

The 1990s were a time of heavy procedures with lots of steps to develop products. There was a procedure for everything — a procedure to design, a procedure to develop, a procedure to test. The thinking was that if everything was written down, you could plug anyone into a role and the product would come out perfect. The reality was that it didn't work. Deliveries were always late with lots of problems.

So in 2001, seventeen people who had developed "lightweight" development processes got together at a ski resort in Utah. They had already proved their approaches could be more effective than heavier processes. Together they wrote the Manifesto for Agile Software Development.

While the Manifesto is specific to software, many of its signatories and others have adapted it for other industries — hardware, marketing, medicine, finance, even restaurants. Let's look at it and consider how it applies to your work.

The Four Values

The Manifesto opens with four values, each expressed as a preference rather than an absolute:

Individuals and interactions over processes and tools. This doesn't mean processes are unimportant. It means that where we can, we should turn up the emphasis on people talking to each other rather than following procedures at the expense of connection.

Working software over comprehensive documentation. In the 1990s, heavy processes produced binders full of documentation and very little working product. The second value reorients the team toward delivery. The customer needs value, not aesthetics or audit trails.

Customer collaboration over contract negotiation. Too often teams become order-takers — receiving requirements and delivering output as if they were machines, finding out later they missed the mark. Working together and iterating over time gives a better outcome than ping-ponging requirements back and forth.

Responding to change over following a plan. In the world of complex products, we don't know everything upfront. We need a heading and a direction, but we must be prepared to change course when we learn something new.

The 12 Principles

The Manifesto is accompanied by 12 principles that give the values practical shape. Grouped by theme, they cover four areas:

People

Build projects around motivated individuals and give them the environment and support they need. And maintain a constant pace that sponsors, developers, and users can sustain indefinitely.

These two make a case that agile is as much about how people work as what they produce. You can follow every Scrum event perfectly and still burn your team out if you ignore them.

Delivery

The highest priority is satisfying the customer through early and continuous delivery of valuable software. Deliver working software frequently, with a preference for shorter timescales. Working software is the primary measure of progress. And continuously attend to technical excellence and good design.

Notice what's absent: the word "fast." Agile doesn't ask teams to move recklessly. It asks them to move consistently, and to keep quality high while doing it.

Collaboration

Business people and developers must work together daily throughout the project. The most efficient method of conveying information is face-to-face conversation. And welcome changing requirements, even late in development — agile processes harness change for the customer's competitive advantage.

That last principle still surprises people. Traditional processes treat late changes as failures. Agile treats them as information.

The closer you work with your customer, the earlier you get that information — and the less it costs you.

Improvement

The best architectures, requirements, and designs emerge from self-organizing teams. Simplicity — the art of maximizing the amount of work not done — is essential. And at regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

That last one is the only principle that describes an explicit practice. Every other principle states a value or a belief. This one tells you to stop, look back, and change. It's the principle that makes all the others sustainable — without regular reflection, teams drift from everything the manifesto asks of them.

These principles don't just describe how a team works. They describe how a team stays connected to the people and purpose around it — and that connection is what the rest of this book is about.

2

Frameworks — Scrum, Kanban, and How to Choose

Scrum

The most popular agile framework is Scrum — a term borrowed from rugby, where players lock arms to move the ball down the field together. As the Scrum Guide puts it:

"Scrum requires a Scrum Master to foster an environment where: a Product Owner orders the work for a complex problem into a Product Backlog; the Scrum Team turns a selection of the work into an Increment of value during a Sprint; the Scrum Team and its stakeholders inspect the results and adjust for the next Sprint; then repeat."

Three roles work together: the Scrum Master, who coaches the team; the Product Owner, who maximizes the value of the product; and the Developers, who build it.

A Sprint in Practice

A sprint is a fixed period of one month or less during which the team turns selected backlog items into a working increment of the product. Most teams run two-week sprints. The events below are based on that cadence, but they scale with the length of your sprint — a four-week sprint will have longer planning and review sessions. The one exception is the Daily Scrum, which stays at fifteen minutes regardless.

Day 1 — Sprint Planning. The team comes together to answer two questions: what can we commit to delivering this sprint, and how will we do it? The Product Owner brings the top items from the backlog, already ordered by priority. The team discusses each item, asks questions, and selects what they can realistically complete. By the end of planning, the team has a sprint goal — a single sentence describing what this sprint is trying to achieve — and a sprint backlog of items they've committed to delivering.

Days 2–9 — The work. Each day begins with the Daily Scrum, a fifteen-minute check-in where the team reads the sprint together. Not a status report to a manager. A conversation among the team: what moved yesterday, what's moving today, and is anything blocking progress. Blockers the team can't resolve themselves go immediately to whoever can — the Scrum Master, a leader, a dependency owner. The rest of the day is heads-down work.

Somewhere in the middle of the sprint — most teams do this around day 5 or 6 — the team holds a backlog refinement session. This is where future work gets prepared. The Product Owner brings upcoming backlog items and the team discusses them: is the item clear enough to work on, is it sized, does it need to be broken into smaller pieces? The goal is to make sure that

when sprint planning arrives at the start of the next sprint, the team isn't seeing items for the first time. Nothing slows a team down faster than walking into planning with a backlog full of vague, unrefined items and spending the entire session trying to understand what's being asked rather than committing to what to build.

Refinement isn't a formal Scrum event. It's not prescribed in the Scrum Guide the way planning and retrospectives are. But in practice, teams that skip it consistently struggle with planning. Think of it as maintenance on the backlog. A little investment mid-sprint saves a lot of pain at the start of the next one.

Day 10 — Sprint Review and Retrospective. The sprint closes with two events. The Sprint Review is outward-facing: the team demonstrates what they built to stakeholders and collects feedback. Working product only — no slide decks summarizing what was done. The Retrospective is inward-facing: the team reflects on how they worked together. What helped, what didn't, what one thing will they try differently next sprint. Both events are short. Both are essential.

Then it starts again.

That last line is the part new teams often underestimate. A sprint isn't a project with a beginning and an end. It's one cycle of a continuous loop. The team that ran a mediocre first sprint and reflected honestly on why will outperform the team that ran a perfect first sprint and assumed they had it figured out.

Size, Don't Estimate

If there is one topic that causes more confusion in agile teams than any other, it's story points.

New teams tie themselves in knots over them. What is one

story point? Is it a day? Is it an hour? Can I have half a point? And then, almost inevitably, someone starts assigning points to individual people — tracking how many points each developer completed in a sprint, using it as a performance metric. At that point the tool has been completely inverted from its original purpose.

Story points were never meant to measure individual output. They were a team-level, relative sizing tool — a way to quickly answer two questions: is this item too big to bring into a sprint, and does the team have a shared understanding of what's being asked? That's it. The number itself was never the point.

Here's what's changed: the Scrum Guide no longer prescribes story points at all. Some teams use t-shirt sizes. Some use buckets. And some teams have abandoned estimation altogether — they do this by deliberately breaking all backlog items down to roughly the same size and effort, then simply counting how many stories they complete per sprint. No points, no debate, no conversion to hours. Just a consistent unit of work and a count.

What hasn't changed is the underlying principle: sizing should be quick and reliable. Quick means the team isn't spending half a sprint planning session deconstructing a backlog item into a mini Gantt chart. Reliable means the team converges fast because they share a common understanding of the work.

However your team sizes — points, shirts, buckets, or same-size stories — ask these two questions of every approach: is it fast, and does it create alignment? If yes, you're doing it right. If your sizing method has become a source of anxiety, blame, or individual performance tracking, it's time to change the method.

The goal was never the number. The goal was the conversation.

Kanban

The name Kanban comes from Japanese — "kan" means sign and "ban" means board, so Kanban translates as "signboard." It became popular as part of Toyota's manufacturing system and focuses on visualizing work and managing flow rather than planning in fixed iterations.

According to the Kanban Guide (kanbanguides.org), Kanban has three core practices: defining and visualizing a workflow, actively managing items in that workflow, and improving the workflow over time.

Kanban Metrics

One of Kanban's strengths is that it generates useful data naturally, just by teams doing their work. You need a board, cards moving across it, and four numbers worth paying attention to.

Cycle time is how long it takes a single item to move from start to done. If a marketing team starts working on a campaign brief on Monday and publishes it the following Wednesday, the cycle time is nine days. Tracking cycle time over many items tells you what your team's typical pace looks like.

Throughput is how many items your team completes in a given period. If the same marketing team finishes twelve campaigns in a month, their throughput is twelve. Throughput counts finished work, not work in progress. A team can look very busy and have very low throughput if items are starting but not finishing.

Work in progress — commonly called WIP — is the number of items actively being worked on at any given moment. High WIP is almost always the root cause of slow cycle times. Most Kanban

teams set a WIP limit — a maximum number of items allowed in any column — to force the team to finish before starting something new. It feels counterintuitive at first. Doing less at once leads to delivering more overall.

Work item age is how long a card has been in its current column without moving. A campaign brief sitting in "in review" for eleven days is a warning signal that something is stuck. Regularly scanning for old work items and asking why they haven't moved is one of the most valuable habits a Kanban team can build.

Together these four metrics give a clear picture of flow: how fast things move, how many things finish, how much is in flight, and where things are getting stuck. No time sheets, no individual performance tracking, no complicated reporting. Just the board, the cards, and four honest numbers.

How to Choose — Scrum or Kanban?

The most honest answer to this question is: look at your work before you look at the framework.

Scrum works best when work arrives in chunks that can be planned, the team can commit to a goal for a fixed period, and stakeholders benefit from regular demonstrations of progress. If your team can answer the question "what will we complete in the next two weeks?" with reasonable confidence, Scrum gives you the structure to make and honor that commitment.

Kanban works best when work arrives continuously and unpredictably, and the team's job is to keep things moving rather than plan in advance. Support teams, operations teams, and maintenance work fit this pattern. There is no planning ceremony that assumes you know what's coming — just a board,

a set of priorities, and a commitment to finishing before starting something new.

Many teams find that their work has elements of both. A software team might run Scrum for feature development and use a Kanban board for bug fixes running alongside their sprints. The frameworks aren't competitors. They're tools.

When in doubt, start with Scrum. The structure it provides gives new teams guardrails while they build their agile instincts. Once the team is comfortable with iterative delivery, they'll know naturally whether Scrum, Kanban, or a blend of both is the right fit.

Other Frameworks

Two other frameworks worth knowing: Extreme Programming (extremeprogramming.org), one of the earliest agile approaches for software teams; and Heart of Agile (heartofagile.com), created by Alistair Cockburn — one of the original Manifesto signatories — which focuses on four themes: Collaborate, Deliver, Reflect, and Improve.

Management Teams as Agile Teams

We tend to think of agile teams as delivery teams. But management teams can adopt the same frameworks. A leadership team that meets regularly to review priorities, surface impediments, and reflect on how they're working together is practicing agile — whether they call it that or not. This becomes especially important as organizations grow and the connection between strategy and delivery needs to be deliberate. We'll return to this

in Chapter 5.

3

Building Your Backlog

The Backlog as Backbone

However your teams practice agility, one common artifact ties them together: the backlog. In Scrum it's known as the Product Backlog — the single source of work for the team, an ordered list of everything needed to improve the product.

Think of it as a wish list of what a product or service needs to accomplish for the customer. Unless the team is the entire company, there will be other stakeholders collaborating with the agile team. It helps to distinguish between two types of teams and their backlogs: product teams, who create a product or service for an external or internal customer, and business teams, composed of leadership and supporting functions like finance, operations, and people teams.

User Stories

Whether the product is software, a physical good, a service, or a combination, there needs to be a model representing what the product should do. In agile, the product backlog is built from user stories — short narratives written from the customer's perspective, not the team's.

User stories describe how customers will interact with the product. The team then takes those stories and breaks them down into tasks to execute in an iteration. Stories can also be grouped into larger stories for complex products where it helps to work at a higher level of abstraction before breaking things down.

User Story Maps

User story maps have become a popular technique for building product backlogs, thanks largely to Jeff Patton's book "User Story Mapping." The concept works well when a product requires a flow of actions from one or more users.

Rather than a flat, prioritized list of stories, a user story map narrates the flow of key activities horizontally — then for each activity, breaks down the steps vertically. This two-dimensional grid makes it easy to see which user is performing which action and why. Patton's book is worth reading in full, but watching one of his talks on YouTube will also show you how powerful this technique is for bringing a backlog to life.

Impact Mapping

Impact mapping works well when the flow of user actions isn't yet clear. It uses four columns: start with the overall goal, then identify all the actors needed to achieve that goal. For each actor, specify the impact they need to create to meet the goal. Finally, for each impact, define a deliverable that maps back to that impact, actor, and goal.

The resulting list of deliverables becomes your team's backlog — and because every item traces back to a goal and an actor, the team always knows why they're building what they're building. For more on this technique, visit impactmapping.org.

Collaboration and Liberating Structures

Building a backlog requires genuine collaboration, and genuine collaboration is harder than it sounds. Most meetings have a default shape: one or two people talk, everyone else listens, and the group leaves having confirmed whatever the loudest voice already believed. A backlog built that way reflects one perspective, not the team's collective knowledge.

Liberating Structures, developed by Keith McCandless and Henri Lipmanowicz, is a set of facilitation techniques designed to distribute participation across a group rather than concentrate it at the front of the room. Instead of a few people driving the conversation, everyone contributes, and the group determines what happens next. For backlog creation in particular, activities like 1-2-4-All — where ideas move from individual reflection to pairs to small groups to the whole room — surface input that would never emerge in a standard meeting format.

You don't need to master the whole library to benefit from it.

Start with one activity. Visit liberatingstructures.com and try 1-2-4-All at your next backlog session. The difference in what the room produces will make the case better than any description can.

Vision and Goals

When practicing agile, teams don't know everything upfront. That's by design. But without a shared vision and goal, every decision the team makes in uncertainty becomes a guess. A clear vision gives teams enough direction to move confidently without needing every detail spelled out.

President Kennedy's 1961 address to a joint session of Congress is still one of the best examples of this. He committed the nation to landing a person on the moon and returning them safely to Earth before the decade was out. Enough detail to set the direction. No mention of the Saturn V rocket, the mission architecture, or how long the journey would take. The vision was the destination — the team figured out the path.

Your backlog needs the same thing. What is the vision? What is the nearest waypoint — a goal achievable in the next quarter or year? Is it visible somewhere the whole team can see it regularly? A good vision statement shouldn't need updating while teams are executing. It should be stable enough to make decisions against, and clear enough that the team knows when they've arrived.

A healthy product backlog is necessary. But a backlog that connects to nothing beyond the team is just a list. The next question is what it links to.

4

Business Agility

What Makes a Business Team Different

Every agile transformation eventually hits the same wall. The product teams are running sprints, delivering incrementally, and improving every retrospective. And yet somehow the business isn't moving any faster. Decisions still take weeks. Priorities still shift without warning. Teams still find themselves building things nobody asked for.

The reason is almost always the same: the business team never transformed.

A business team is not a product team. It doesn't build software or deliver a service directly to a customer. It sets direction, allocates resources, removes organizational obstacles, and decides what the company is trying to achieve. In a traditional organization, this team operates on annual planning cycles, quarterly reviews, and hierarchical approval chains. Agile never touched any of that.

Whether your leadership team has never encountered agile or

has watched product teams adopt it for years without joining them, the question is the same: how does the business itself become more agile? The answer starts with two tools — Value Stream Mapping and OKRs — and ends with a backlog.

Value Stream Mapping

A value stream is the sequence of steps required to deliver value to a customer — from the moment a request is made to the moment it's fulfilled. Value Stream Mapping (VSM) is a technique for making that sequence visible.

To run a basic VSM session, gather the people who own each step in the process and map it out together — physically on a whiteboard or virtually on a shared screen. For each step, capture how long it takes to complete and how long work sits waiting before that step begins. That waiting time is where waste lives. In most organizations, the ratio of active work time to total elapsed time is startlingly small. A process that takes three weeks from request to delivery might involve only four hours of actual work.

Once the map is in front of the team, the waste becomes impossible to ignore. Those waiting times, handoff delays, and approval bottlenecks become backlog items — specific problems for the business team to solve iteratively. VSM gives the business team its first backlog. For more on this technique, "Learning to See" by Mike Rother and John Shook is the definitive reference.

OKRs

OKRs — Objectives and Key Results — are a goal-setting tool that shifts the focus from what a team will do to what the business will achieve. An Objective is a qualitative statement of direction: ambitious, memorable, and free of numbers. Key Results are the measurable outcomes that will tell you whether you've achieved it — typically three to five per Objective.

The difference from traditional goal-setting is significant. A traditional goal might be "launch the new customer portal by Q3." An OKR reframes it: the Objective might be "make it effortless for customers to do business with us," with Key Results like "reduce average support ticket resolution time from five days to one" and "increase self-service adoption from 30% to 70%." The team now knows the destination, not just the task.

OKRs work well alongside VSM. Where VSM surfaces process problems, OKRs define the outcomes the business is trying to reach. Together they give a business team both a diagnosis and a direction. Each OKR can be broken down into initiatives and then into backlog items, creating a direct line from the boardroom to the team's sprint. For examples across industries, whatmatters.com is a good starting point.

Connecting Business and Product Backlogs

A product team without a connected business team is a car without a steering wheel. It might move fast. It might even move smoothly. But nobody is confident it's going the right direction.

The connection happens at the backlog level. Business team OKRs get broken into initiatives. Those initiatives become epics

— large bodies of work — that sit at the top of product team backlogs. Each sprint, the product team delivers against those epics. Each quarter, the business team reviews whether the Key Results are moving. If they are, the work continues. If they aren't, the direction changes — not in a crisis, but as a natural part of the cycle.

This is what aligned agility looks like in practice. The business team isn't just a sponsor writing checks. It's an active participant in the same iterative loop the product teams are running — setting direction, inspecting outcomes, and adapting. The backlogs are different. The cadence is different. But the underlying rhythm is the same.

Bridge — Why This Requires Uniting

None of this works if the business team and the product teams are operating in separate worlds.

VSM requires people from across the organization to map a process together — people who often have never been in the same room for that purpose. OKRs require leadership to be transparent about what the company is actually trying to achieve, and humble enough to admit when it isn't working. Connecting backlogs requires trust that product teams will deliver on what they've committed to, and trust that leadership will hold the direction steady long enough for that delivery to matter.

That trust doesn't come from a framework. It comes from people — across levels, across functions, across teams — choosing to work together rather than alongside each other. That's what the next chapter is about.

5

Uniting Teams

The Scaling Problem

A single agile team working well is a beautiful thing. The daily scrum is tight, the backlog is healthy, the retrospectives produce real change. And then the organization asks that team to work with another team. And another. And suddenly the thing that made the team effective — its focus, its rhythm, its internal trust — becomes a liability. The team is so good at looking inward that it stops looking sideways.

This is the scaling problem. It isn't a process failure. It's a success failure. Teams get so good at running their own Scrum or Kanban system that they forget other teams exist — teams they depend on, teams that depend on them. The dependency surfaces late, usually mid-sprint, and the options are all bad: miss the commitment, deliver something that can't be integrated, or build something that directly conflicts with what another team just shipped. The sprint that looked healthy on Monday is in trouble by Thursday, and nobody saw it coming

because everyone was heads down in their own backlog.

Scaling agile is the discipline of solving this problem — deliberately, repeatedly, before it becomes a crisis.

SAFe vs Scrum@Scale

Two frameworks dominate the scaling conversation, and they approach the problem differently.

SAFe — the Scaled Agile Framework — is comprehensive and prescriptive. It defines roles, events, and artifacts at every level of the organization, from individual teams up to the portfolio. The tradeoff is complexity: implementing SAFe is a significant undertaking that requires dedicated training, coaching, and organizational change management. It tends to find its home in large IT and digital technology departments where the need for coordination is high and the appetite for structure matches.

Scrum@Scale, created by Scrum co-creator Jeff Sutherland, takes a different approach. Rather than prescribing a new structure, it scales Scrum itself. A Scrum of Scrums brings together representatives from multiple teams — typically the Scrum Masters — to coordinate dependencies and impediments across teams, mirroring the Daily Scrum at a higher level. A Chief Product Owner network mirrors the Product Owner role across teams. Scrum@Scale tends to work well for product-focused organizations where teams already run Scrum confidently and need a lightweight way to link up.

Neither framework is universally right. The question to ask is: how many teams need to coordinate, and how complex are their dependencies? A handful of teams with manageable overlap can often link up with a simple Scrum of Scrums and a shared backlog. Fifty teams across three product lines probably need

more structure.

Linking Leadership Objectives to Team Backlogs

Frameworks solve the coordination problem between teams. They don't solve the alignment problem between teams and the business. That requires a different kind of connection — one that runs vertically through the organization rather than horizontally across it.

The connection point is the backlog. As discussed in Chapter 4, business team OKRs get broken into initiatives, and those initiatives become epics at the top of product team backlogs. But the link only holds if the people responsible for those objectives are in regular conversation with the people delivering against them.

In practice this means Product Owners and leadership need to get together on a recurring basis — not daily, but consistently. Weekly or fortnightly is enough. The agenda is simple: are the teams delivering what the business needs, are the priorities still correct, and are there decisions only leadership can make that are blocking progress. That last question is the most important. Teams that are waiting on a leadership decision and can't surface it will work around it — and working around it almost always creates waste.

This recurring conversation doesn't need to be a formal ceremony. It can be a standing meeting, a shared board review, or a structured check-in. What it cannot be is optional. The moment it becomes optional it becomes irregular, and the moment it becomes irregular the vertical connection between strategy and execution starts to fray.

Sequencing Your Scaling Journey

The most common mistake organizations make when scaling agile is trying to do too much at once. A transformation that launches twenty teams simultaneously, installs a scaling framework on day one, and expects alignment within a quarter is almost guaranteed to produce chaos — and a lot of cynical people who will tell you agile doesn't work.

The sequencing that has worked in practice looks more like this.

Start with one team that volunteers. Not a team that was assigned to be agile. A team that raised its hand. Motivated individuals, as the manifesto says. Give that team time to actually become agile, not just learn the vocabulary. That means delivering working product at the end of every sprint, running real retrospectives, and building a healthy backlog. This takes longer than most organizations want to wait. It's worth it.

Once that first team is running well, link it deliberately with one other team that has a real dependency. Start a Scrum of Scrums between just those two teams. Identify the shared dependencies explicitly and put them on a visible board both teams can see. Run this for a full quarter before adding a third team.

What you are building in this phase is not process — it's trust. The teams need to experience what it feels like to surface a dependency early and have it resolved, rather than discover it late and absorb the damage. That experience changes how teams think about each other. They stop being obstacles and start being partners.

Add teams gradually, quarter by quarter. Each new team joins a system that is already working, coached by teams that

have already been through it. The scaling framework — SAFe, Scrum@Scale, or something simpler — can be introduced once the teams have enough experience to know what problem it's solving. Imposing a framework before teams feel the coordination pain is like installing a traffic management system before the roads are busy. Nobody understands why it's there.

The United Payoff

There is a version of an organization that most leaders say they want but few have seen up close. Decisions get made at the right level by the people with the right information. Teams deliver consistently and predictably, not in heroic bursts followed by long recoveries. When the market changes, the organization responds within weeks, not quarters. Leadership trusts the teams. The teams trust leadership. And the customer feels all of it.

This is what a united agile organization looks like. Not perfect. Not frictionless. But genuinely aligned — from the business team setting objectives, through the product teams delivering against them, to the customer receiving value at the other end.

Getting there requires everything this book has described. Teams that have internalized the values and principles, not just the ceremonies. Backlogs that trace to real outcomes, not just feature lists. Business teams that have done the work of mapping their processes and setting honest goals. And a scaling approach that grows deliberately, with trust built at every step before the next one is taken.

But more than any of that, it requires people choosing to look up from their own backlog and ask how the team next to them is doing. That question — simple, human, easy to skip — is

the one that makes the difference between a collection of agile teams and a united agile organization.

Ask it early. Ask it often. Don't wait for the framework to tell you to.

6

Epilogue: Your Turn

This book set out to be useful, not exhaustive. If you've made it here, you have enough to start — enough to understand what agile actually is, enough to choose a framework, enough to build a backlog that connects to something real, and enough to see how the pieces need to fit together across teams and business.

That last part is the one most organizations skip. They invest in the teams and forget the connection. They train the Scrum Masters and leave the leaders out. They build great delivery machines and aim them in the wrong direction. The gap between what agile promises and what most organizations experience lives right there — in the space between teams doing agile and an organization being agile.

Closing that gap is the work. Not the framework, not the certification, not the transformation program. The work is people — at every level, across every team — choosing to look up from their own backlog and ask how the whole is doing.

Agile is not the destination. It's how you travel. The destination is an organization that delivers real value to real customers, learns faster than its competitors, and gives its people work

worth doing. Agile, done well and united across the business, is one of the best paths to get there.

You know your organization. You know where the gaps are. You know which teams are running well in isolation and which conversations aren't happening. That knowledge is the starting point.

If this book raised more questions than it answered — about what happens when scaling goes wrong, or why so many transformations stall out after the first year — those questions are the subject of the next book in this series. Sanity at Scale goes deeper into the organizational dynamics that make or break large-scale agile adoption. Details at betterwaysofworking.com.

So — how will you unite your teams?

About the Author

David Borzillo is an agile coach and Registered Scrum Trainer™ based in South Florida. He has been part of agile transformations in various roles over more than a decade, and is the author of three books on agile practice: "United Agility," "Who Killed Agile?" and the upcoming "Sanity at Scale." David operates under the brand Better Ways of Working and is passionate about people coming together to collaborate, deliver, and improve.

LinkedIn: linkedin.com/in/daveborzillo

All links in this book verified June 2026.

You can connect with me on:

http://betterwaysofworking.com