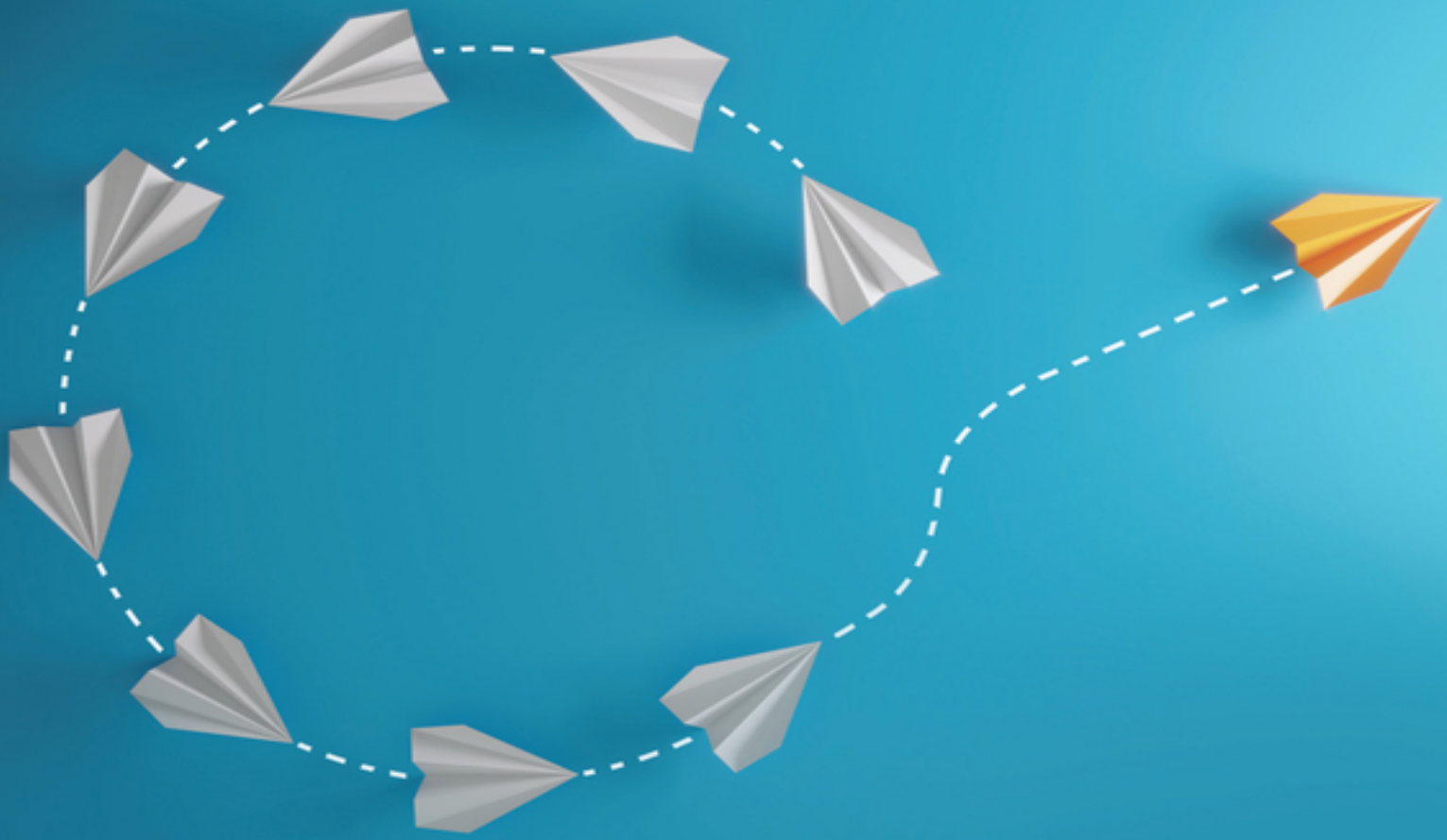
Cliff Berg, Kurt Cagle, Lisa Cooney, Philippa Fewell,
Adrian Lander, Raj Nagappan, Murray Robinson



Agile 2

The Next Iteration of Agile

WILEY

Table of Contents

List of Tables

List of Illustrations

Agile 2

The Next Iteration of Agile

Cliff Berg
Kurt Cagle
Lisa Cooney
Philippa Fewell
Adrian Lander
Raj Nagappan
Murray Robinson

WILEY

Foreword

You can recognize a great book by its ability to make obvious what is wrong with existing worldviews and to add new insights or nuances to change or improve this worldview. I believe *Agile 2* meets these criteria easily. It is an easy read not only for those new to the subject of agility, but also for die-hard professionals who are looking for something “beyond” the basic Agile concepts that are at times dogmatic and often being misused in practice.

In this era of digital disruption and an ever-growing world full of volatility, uncertainty, complexity, and ambiguity, it becomes increasingly important for organizations to become more agile. The Agile Manifesto, originally intended in 2001 to disrupt traditional, not-too-effective software development practices, has inspired many organizations over the past decades to change their ways of working, affecting both work cultures and structures. Its popularity was boosted by a growing workforce consisting of millennials and Generation-Z professionals who demand more autonomy, ownership, and the opportunity to make meaningful impact. Over the past decade, the “Agile movement” has gained increasing momentum and also moved beyond the realm of IT.

A side effect of its success and growth was that all kinds of Agile frameworks, doctrines, and certifications popped up to standardize and monetize the discipline. The original Agile values and principles, being high-level on purpose, gave ample room for various interpretations of the core paradigms. During my career I have worked with many different Agile coaches and consultants, and I was always surprised by how much discussion and fanatic debates arose among them with regard to how to live certain Agile

values or implement specific practices. This tribalism led to confusion among non-Agilists, and this hampers Agile transformations significantly. I thus see a clear need for a comprehensive Agile idea set that is both pragmatic and nuanced by nature. Enter *Agile 2*.

I was happy to find out quickly that the authors do not claim to have written yet another Agile doctrine or “Bible.” Instead, they have written a pragmatic companion guide that will be useful for managers and specialists alike. It is packed with hands-on tactics and practices that can help leaders and specialists in organizations to grow to a next level of agility, while preventing cargo-cult behavior or *avant le lettre* implementations that often do more harm than good.

One of my key drivers for cofounding the DevOps Agile Skills Association (DASA) in 2016 was building a comprehensive view on how to create high-performance IT organizations. The popularity of DASA stems largely from the six DevOps and Agile principles that advocate continuous improvement, customer centricity, autonomous multidisciplinary teams, and product thinking. Following these principles often results in a digital and organizational transformation that typically goes far beyond choosing a standard Agile framework or adding some basic Agile rituals to the mix. To transform successfully to high-performance, organizations need a more mature take on and guidance on what it means to really “be Agile” at scale. Providing this guidance is one of this book's core differentiating features.

Over the past decade I learned firsthand as a consultant, trainer, and senior leader the importance of building the right type of leadership in the organization and creating a culture of continuous learning, experimentation, and innovation. What I like about *Agile 2* is that both the

importance of leadership and learning are advocated strongly. It provides many tangible ideas to reimagine an organization's leadership culture. I wish that I had this book on my nightstand five years ago. It would have helped me greatly in understanding why certain things happened—or did not happen—during the organizational transformations I was leading.

I like the fact that the authors do not intend to reinvent the wheel, but are keen on building on what is already working. Some of the key Agile values and principles are powerful to this day, but application in practice often needs some additional clarity and lots of examples. The authors nicely provide nuance to how to interpret Agile principles and values while referring to many interesting, and more recent, bodies of work. The authors hit the nail with addressing key topics that are haunting many organizations, leaders, and teams, such as how to collaborate, communicate, value both experts and generalists, and commit team capacity. They rightfully argue that how to adopt certain principles or how to interpret certain values depends on your organization's needs and its current level of maturity. Using this book as your Agile guide, you can aim and navigate your transformation in a more tailor-made way, resulting in more business value. I expect this book to be found on many nightstands in the coming years.

Dr. Rik Farenhorst
Senior IT Exec | Trainer | Coach | Speaker | Writer on Creating High-Performance Digital Organizations | Co-founder of DevOps Agile Skills Association (DASA)
Utrecht, The Netherlands
December 2020

Preface

A few people who have become aware of Agile 2 have dismissed it as “more of that Agile stuff,” not realizing that Agile 2 is a departure from the original Agile in attitude, approach, and substance. One of those individuals—a chemical engineer—said that he had discussed Agile at length with an Agile advocate, but still concluded that Agile is not for him. Another—an experienced systems engineer who has testified before Congress regarding aircraft and spacecraft systems reliability—also believes that Agile methods do not provide a robust process for trustworthy systems.

We view ourselves as Agilists, and yet we find widespread doubt about the efficacy and usefulness of Agile in many quarters. One of these is among engineers. These people are not ignorant. They know their job extremely well, yet Agile, as described to them, or as they have experienced it, has not resonated or has not answered critical questions.

The Agile movement also uprooted the product design community to some degree (which we will document in this book), although this is an area in which the Agile community has realized the issue and some are trying to rectify it.

Agile authors largely ignored the role of data: something that is so immensely important, that it is akin to speaking about mountains but missing a vast canyon immediately beside you.

The Agile community also sidestepped the issue of leadership — something that the DevOps community has tried to address. Leadership is so important for any endeavor, that to omit it is, frankly, quite equivocal.

Agile has not resonated among the growing DevOps community. Even though DevOps ideas were developed by people who strongly identified as Agilists, the Agile community at large has remained mostly ignorant of DevOps, which had the effect that DevOps became its own movement. As a result, most Agile coaches today know little about DevOps, and we find that DevOps practitioners often view Agile as superfluous.

You might think that mainstream programmers accept Agile, since they are the ones who use it most directly, but in actuality, there is a lot of doubt about Agile within programming communities in general. That is the biggest irony of all: that Agile, which was created for programmers, has in effect been taken away from them, and no longer serves them.

Agile is mostly accepted within *Agile* communities—comprised of Agile coaches, and managers who have been persuaded of the benefits of Agile. Programmers tend to have mixed feelings about Agile. (We will support that assertion in this book.)

Was Agile described poorly? Is Agile missing things? Did it get some things wrong? Does Agile truly not apply to the needs of the work of any of these people? Since Agile ideas can be applied to most things (in our opinion), we believe that the last explanation is not likely to be the true one.

What we have observed ourselves is that too often, Agilists explain and advocate Agile ideas and methods before asking enough questions. Some of us have seen Agile coaches fired for coming into a setting and insisting on particular practices before actually understanding how the work in that setting is done—a hypocrisy given that Agile coaches so often explain that Agile transformation is a learning journey.

To understand how to apply Agile ideas, one must first understand that domain, how the work is currently done, and why it is done that way. No Agile practice is universal. One size does not fit all, so prescribing before understanding is potentially destructive.

Indeed, the “dogma” of the Agile community helped to launch it, but it has also been its chief failing. Early proponents of Agile insisted that the Agile movement needed to be disruptive—a “call to arms”—and so dogma was called for; but dogmatic insistence also alienates and causes dysfunction when it is not the best advice for the situation.

Agile 2 is not dogmatic. It is not designed to stir up emotion. It is not a call to arms or an attempt to disrupt what we have. As such, it does not try to be disruptive. It does not replace Agile or replace DevOps or replace anything. Agile 2 pivots Agile in some important ways and attempts to fine-tune it. Agile 2 also adds many extremely crucial ideas that have been ignored by much of the Agile community, even though successful Agile practitioners often use those very ideas, and other communities of thought embrace those ideas.

Agile 2 reinforces some DevOps ideas, and some Lean ideas, but Agile 2 does not attempt to duplicate or replace those, and so those sets of ideas are still important in their own right. Agile 2 does not attempt to subsume any existing community of thought. Agile 2 also does not claim to cover all aspects of these topics. Agile 2 claims to only be a set of useful ideas for how to achieve agility in human endeavors and encourages people to include other ideas and fields of thought as well.

Agile 2 is more verbose than the Agile Manifesto. The reason is that we feel that one of the weaknesses of the manifesto was that it over-simplified complex issues. A

simple value maxim cannot describe important trade-offs, and one or two principles cannot address nuanced issues such as what good leadership looks like and which styles of leadership apply best in a given situation. So Agile 2 gives these important topics the space they deserve, from an agility perspective.

One important way that Agile 2 departs from the Agile Manifesto is that Agile 2 provides the foundation of thought from which it was derived. Rather than make bold statements without substantiating them, Agile 2 provides the “problems” and “insights” that arose in discussions about Agile, in the course of the Agile 2 team's retrospective about the state of Agile. It was these problems and insights that led to the Agile 2 principles.

An Agile 2 principle is not intended to be an absolute. This is because there can be no absolutes when it comes to human behavior. An Agile 2 principle is a proposed rule of thumb: true most of the time, but perhaps not in some circumstances. That is why the underlying assumptions and thoughts—the problems and insights—are important for understanding the intention of each principle, meaning what problem it is trying to solve and how.

Someone posted a comment online about Agile 2, saying that if the original Agile Manifesto authors were not involved in the Agile 2 effort, he would not look at it. We believe that all great ideas build upon what has come before, and that even “original” ideas have deep roots. The term *Agile* had been used prior to the creation of the Agile Manifesto, and “agile” methods had been used and circulated for years prior to that. Not only do many Agile methods date back decades, but core ideas in Agile 2 such as Socratic leadership date back millennia.

There is also the matter of dysfunction within the Agile community, which we will discuss at length in the book. The

dogma that is found in some quarters is one form of dysfunction; another is the separation of the community into tribes, for the various frameworks. We will explain why this has been a problem and how it has “frozen” Agile thinking and stifled its evolution.

Many of the thought leaders in the Agile community have a lot invested in current paradigms and practices, and so change is not in their best interest. For these reasons, we felt that we could not rely on the community to fix these problems. The problems come *from* the community—not from the whole community, but from some of the most established and entrenched parts of it.

Why bother then? Why deal with this? It's because Agile is *extremely important*. DevOps cannot replace Agile. While Agile has become mostly about the human side of building things, DevOps has become mostly a collection of technical practices. That is the reality on the ground. But there is more to building things than the technical side: one needs both the human side *and* the technical side.

We therefore realized that addressing Agile's gaps is really important, and that to do it, we needed a diverse team with a wide range of skills, composed of people who are not deeply invested in current paradigms or frameworks. We needed original thinkers. Those criteria led to the Agile 2 team of 15 members, which you can find on the Agile 2 website.

Agile 2 is an attempt to make a solid course correction to Agile, but in an open, additive, inclusive, and nondogmatic or emotional way. We welcome ideas that can supplement Agile 2 and feedback on its principles, in the spirit of inclusiveness and advancing everyone's understanding of these complex issues.

Agile 2 broadens Agile's focus beyond software. The reality is that Agile ideas have been applied for many things besides software, and so the Agile 2 team felt that it made no sense to define Agile 2 only for software.

The reader will notice that many Agile 2 principles are stated in the margin, but not all of them. This book is not a textbook about Agile 2 that covers every aspect of its principles. The purpose of this book is to introduce Agile 2, explain why we need it, and give an overview. You can find more information about Agile 2 on the website: https://agile2.net, which is published under a Creative Commons Attribution license, “CC BY 4.0.”

This book attempts to make the many topics of Agile 2 concrete. We give guidance on how to apply the principles and provide examples. However, we refrain from providing specific steps or templates to follow: we do not want to repeat the mistake of current Agile frameworks in that regard.

Except for our examples, we do not define practices to implement. Practices are important, but that is for another book and for others to propose. This book lays out a conceptual foundation, while using concrete situations as examples, but not for prescription.

A note about the use of the words “agile” and “Agile”: This book uses the word “Agile” when referring to the ideas embraced by the “Agile community,” which is comprised of Agile coaches and others who view the `agilemanifesto.org` document as a guiding source of insight; or in the context of so-called “Agile frameworks” which claim to define practices that are consistent with the philosophies of the Agile community. We use the word “agile” when we intend to convey the generic quality of agility. Agile with a capital “A” and agile with a small “a” are two *different words*.

The name “Agile 2” is not intended to be a version number, as in “2.0,” “2.1,” etc. Rather, it is the name of a reborn Agile. We feel that the principles of agility are timeless, so we do not expect an Agile 3, and so on. Rather, we see Agile 2 as an attempt to reimagine Agile—not from scratch, but by taking Agile ideas and pivoting. We hope that Agile 2 hits closer to the mark!

1
How Did We Get Here?

At a developer conference in 2015, Dave Thomas, one of the authors of the Agile Manifesto, gave a talk titled “Agile Is Dead.”[1] In a 2018 blog post, Ron Jeffries, another Agile Manifesto author, wrote, “Developers should abandon Agile.”[2] In a 2019 article in *Forbes* titled “The End of Agile,” tech author Kurt Cagle wrote, “[Agile] had become a religion.”[3] A post about the article[4] in the programmer forum Slashdot received more than 200 comments from software developers, asserting things like “Agile does not always scale well” and “The definitions of ‘agile’ allow for cargo cult implementations.”

Agile has been a subject of ridicule since its beginning. In the early days, there were many people who did not understand Agile and spoke from ignorance; what has changed is that today the criticism often comes from people who *do* understand Agile methods and have decided that those methods are problematic.

Is Agile actually dead? The statistics say no,[5] yet something is clearly wrong. Agile—which was sold as the solution for software development's ills—has severe problems. What are those problems, how did they happen, and what can be done about them? And is Agile worth saving?

Most of the discussion in this chapter will be about software. That is because Agile began in the software domain. In later chapters, we will broaden the discussion to product development in general, and to other kinds of human endeavor, since many Agile ideas apply to essentially any group effort.

A Culture of Extremes

In 1999 a new book called *Extreme Programming Explained* by Kent Beck sent shock waves through the IT industry. Agile ideas had been circulating and in use prior to this, but Beck's book somehow pierced corporate IT consciousness. It arguably launched the Agile movement, even though the movement was not called “Agile” yet.

The movement's core thesis was that methodical, phase-based projects were too slow and too ineffective for building software—challenging the approach then used by most large organizations and pretty much every government agency.

The book did not launch Extreme Programming, aka XP, which was first defined in 1996,[6] but it was the book that popularized it. Talk about XP could be heard in the halls of every IT shop. It was controversial, but its values strongly resonated: Small teams, working code (rather than documents) as the only real proof of progress, frequent discussions between the customer and the programmers. Out with big, up-front requirements documents that were always incomplete, inconsistent, and incomprehensible; out with big, up-front designs that were usually wrong. Recurring and incremental customer approval instead of contracts that locked everyone in to unvalidated assumptions.

Many of the methods of XP were not new, but they had been outlier methods, and XP put them under a single umbrella. The book strongly asserted that these methods work and are a superior alternative to traditional methods.

It is not that there were no other proposals for how to reshape software development. So-called lightweight methods had been around for a while. Extreme Programming was new in that it threw a grenade into much

current thinking by being so radically different and proposing methods that were so *extreme*—methods such as pair programming (which had been described as early as 1953)[7] and Test-Driven Development (which also had some history prior to XP), which turned many assumptions about programming on their head.

Thus, the movement began as a rejection of the predominant existing paradigms. People knew something was wrong with software development as it was being done. Extreme Programming provided an oppositional alternative. It was not so much that people thought XP was great, but they were sure that current practices were not great. XP received a lot of attention and was a radically different approach. Perhaps the attention was not because XP was so much better or radical, as there had been other ideas circulating such as Rapid Application Development, but perhaps XP got attention mostly because the Internet provided a new medium that made rapid awareness possible.

Then in 2001 a group of IT professionals—all men by the way, with most from the United States and a few from Europe—got together over a weekend and hammered out a set of four "values," which they believed should be the foundation of a new approach to building software. Kent Beck was among them. You can find these four values at AgileManifesto.org . It was largely a rejection of many approaches that had become commonplace, such as detailed plans, passing information by documents, and big all-at-once deliveries.

In the weeks that followed, some of them continued the discussion by email and added 12 principles, which you can also find at the same website.

They called all this the Manifesto for Agile Software Development, and it came to be known colloquially as the

Agile Manifesto or just Agile. This “manifesto” took the popular culture baton from XP and other iterative approaches and launched the Agile movement for real.

Extreme Programming had set the tone for what would become the Agile movement, and the tone was to be extreme. In those days, *extreme* was popular. We had extreme rock climbing, extreme skateboarding, extreme pogo, extreme skiing, and extreme pretty much anything. Extreme was in. People were so tired of the ordinary; everything new had to be extreme. It was a new millennium for crying out loud: everything needed a reset!

And so “extreme” was a necessary aspect of anything new and interesting at that moment in time in the late 1990s—the end of the 20th century.

Since Agile was a rejection of what had become established software development methods, it was inherently a disruptive movement, and in the ethos of the time, it had to be extreme. And so it was that every Agile method that came to be proposed—these are called *practices*—were of necessity extreme. Otherwise, they were not seen to be consistent with the spirit of being entirely new and disruptive.

It was not the Agile Manifesto that set things in that direction. The Agile Manifesto was clearly about balance and moderation. It makes no absolute statements: every value is couched as a trade-off. For example, the first value reads, “[We have come to value] individuals and interactions over processes and tools.”

It does not say, “Forget process and tools—only pay attention to individuals and interactions.” Instead, it says, consider both, but pay special attention to individuals and interactions.

In other words, the Agile Manifesto advocated judgment and consideration of context. In that sense, it is a sophisticated document and cannot be used well by people who do not have the experience needed to apply judgment.

But the tone had already been set by XP: extreme practices received the most attention and applause, because XP practices were all extreme. For example, XP's recommendation of pair programming, in which two people sit together and write code together, sharing a keyboard, was considered by many programmers to be extreme. Or everyone sitting side by side in a single room, with all walls removed and no privacy—that was pretty extreme, as it had been assumed that people needed privacy to focus, and the big programmer complaint of the 1990s, depicted in so many Dilbert cartoons, was that programmers were no longer being given offices and instead were being sat in cubicles that did not afford enough quiet or privacy. And now here comes XP and says, in effect, *You got it all wrong; you need to sit next to each other*. That was an extreme swing of the pendulum.

The Scrum framework, which dates in various forms to the 1980s but became reformulated in 1993 and then popularized through its certification regime during the 2000s,[8] added more ideas that are arguably extreme. For example, Scrum views everyone on a team as an equal player—no one is acknowledged as having more standing than anyone else (“Scrum recognizes no titles for Development Team members”[9]), regardless of their experience. That was pretty extreme, since before Agile, programmers in most (not all) organizations had professional levels, such as programmer, senior programmer, architect, etc.

During the first decade of the Agile movement it seemed that new suggested practices were in a competition to be

more extreme than the others. We saw the introduction of mob programming, in which rather than two programmers working together as in pair programming, the entire team works together—literally—everyone calling out their thoughts in a single room and sharing a single keyboard.[10] Then in 2015 the Agile team room was extended by Facebook to an extreme level when it created its 430,000-square-foot open team room.[11]

We also saw the growth of conference formats pushing popular Agile practices to the extreme—for example, the Lean Coffee format in which there is no agenda or the "unconference" and Open Space formats in which people vote on topics and join ad hoc conversations. These contrast strongly with a traditional conference or discussion group format, which generally has an agenda and scheduled talks, with informal sessions afterward.

But do extremes work?

Certainly they work for *something*. There is always some use for any tool. The question is, are the extreme practices advocated by many among the Agile community actually the most effective method for a wide range of situations that commonly occur in organizations? Another question is, do these practices favor certain ways of working at the expense of others so that certain people benefit but others are at a disadvantage? Or, should a thoughtful approach be used, with moderate approaches being the norm and extreme approaches used sparingly and when a particular form of activity is desired for a specific reason?

Since the Agile movement began through advocacy of extremes and was inherently a disruptive movement, it became evangelistic, and dogmatic elements arose. As a result, if one did not embrace the trending favored set of Agile practices, one was at risk of being labeled an "Agile doubter." That label was brandished readily by many Agile

coaches. And so extremes came to be not just something to consider, but *the* way—and the only way. Agile was now something to accept on faith; as Kurt Cagle had written, it had become a religion.

Divided and Branded

Whenever something new and useful is created, people and organizations jump in to claim it and use it for their own purposes. Any change creates huge opportunities. For example, when Howard Head came out with the oversized Prince tennis racquet in the early 1970s, other manufacturers followed Head's lead and came out with racquets that departed from the standard size. One suddenly saw racquets on the market with very large nets and also ones that were only slightly larger than what was then the standard size.

The ones that were slightly larger became the most popular and came to be seen as the new standard size. These were called “midsize” racquets, although today we just call them racquets.[12]

The inevitable result was that everyone who owned a tennis racquet had to go out and buy a new one; otherwise, they were not adhering to the new “standard.” The sports equipment industry experienced a windfall in sales.

The rise of Agile did the same thing. There were new books, new websites, new consulting practices, and new frameworks that purported to be Agile. Agile quickly became a commodity to sell. It began with an Agile certification industry. Ken Schwaber introduced a two-day certification course for his Scrum framework: the Certified Scrum Master, aka CSM. By sitting in a training room for two days and not even taking a test (they do provide a

simple test now), one could walk away with a certificate claiming to be a “master.”

Essentially the material that could be contained in a small pamphlet was the basis of what Human Resources staff and many hiring managers erroneously interpreted as a “master-level” certification.

Since Schwaber and his partner, Jeff Sutherland, claimed that Scrum was an Agile framework, it was something they could sell under the rising banner of Agile. Organizations that preferred to hire people with certifications made the CSM a requirement. People who had master's degrees from universities were dismayed at the naively perceived equivalence of a master's degree and a Certified Scrum Master certification. Highly qualified people were screened from job applications because they did not have the two-day CSM certification.

Industry groups sprang up: the Agile Alliance, the Scrum Alliance, Scrum.org , ICAgile, SAFe, LeSS, Kanban, and many others. The large consulting companies had a hard time learning about Agile, because Agile's central message of being lean and efficient and not having big contracted “phases” was antithetical to their model. Eventually they figured out how to incorporate it into their offerings, and today they all have substantial Agile practices, claiming to be the experts in Agile.

Thus, there is a lot of money today in the Agile industry, and the Agile community is arguably driven by moneyed interests, with a continuing tide of people seeking certifications in the various frameworks and becoming indoctrinated into them. The phrase *Agile industrial complex*, which might have been coined by Martin Fowler (one of the authors of the Agile Manifesto), has come to be used to refer to the industry as a commentary on the

degree to which it is driven by financial interests rather than by ideas and efficacy.

Controlled by Dogma

Today Agile is big business, and it is highly competitive.

Large consulting companies have traditionally used their partners to fly around and build relationships with their clients' executives, convincing the executives that the partners and consulting firms have strategic insight. That was the case before COVID-19 and will probably resume being the case after COVID-19. Through those relationships, the partners are able to place large numbers of Agile-certified staff on-site, generating a lot of revenue.

Placing staff is their goal—as many as possible. These staff members do not usually have the industry experience that the partners have, but they have a certificate, perhaps a Certified Scrum Master certificate, perhaps a SAFe certificate, or perhaps others. In other words, they sat in a classroom for a few days or weeks, and they probably participated in at least one project that was said to be Agile. Most are not what one would normally expect from a consultant who is advising teams and programs: decades of practical experience at multiple levels of responsibility, great acumen, demonstrated industry thought leadership, and a history of P&L accountability.

Certainly there are many people in large consulting companies who are very qualified, but we have seen many who are not. The point is that one should not treat a large consulting company as if it is inherently more trustworthy than any other with regard to Agile expertise. Have their people achieved business results? The partners will tell you they have and will provide proof, but you know how data can be misused.

Many of the various industry groups do not like each other either. One of the largest Scrum training organizations—one that claims to speak for the Scrum community at large—actually writes into its contract with its trainers that one cannot train for a “competing” training framework such as the Scaled Agile Framework. Thus, the organization does not want its customers to have a choice or to be able to consider other ideas. Limiting what one’s trainers know does not seem to us like a good way to serve one’s customers.

Meanwhile, the organization preaches Agile's value of “customer collaboration over contract negotiation” and professes to being open to all ideas and to being cooperative and collaborative—core Agile attributes. You should not miss the hypocrisy in this.

An Agile training organization's armies are its certified coaches. When the CSM certification was introduced, it had an unintended effect (at least, there is no evidence that it was intentional). Large numbers of people who had no software development experience took the training and became certified. Almost overnight they had a new career: their CSM certification could get them a job as a Scrum Master or an Agile coach.

We then saw the rise of these two new professions (Scrum Master and Agile coach), and they quickly came to be dominated by nontechnical people from every manner of prior profession. Since their background was nontechnical, they emphasized the human side of software teams: team trust, team happiness, and so on, as well as the set of Scrum practices: the sprint planning, the daily scrum (aka daily standup), the sprint review, and the retrospective. The technical focus that XP had was almost entirely lost, and by 2011, 52% of Agile teams reported using Scrum (which

defines no technical practices), compared to only 2% using the highly technical XP.[13]

This is a big problem. In a talk at Agile Australia in 2018, Martin Fowler, one of the authors of the Agile Manifesto, asked the audience, “How many people here are software developers?” Then he said, “A smattering, but actually very much a minority … and that's actually quite tragic.”[14]

Agile coaches and Scrum Masters focused on the nontechnical practices as if they were the be-all and end-all of software development, ignoring critical things such as test coverage, code branching, integration testing, issue management, and all the critical things that every programmer knows are essential. The retrospective, in which a team is supposed to talk about how to improve their work, was facilitated by the Scrum Master, who was usually nontechnical, and so the discussion tended to steer toward the Scrum practices, because team members did not want to bring up issues that the Scrum Master would not understand. Teams then went back to their desks, anxious to get back to work after all of the Scrum-related ceremonies (what the Scrum practices are called), and so important technical issues went undiscussed.

The Agile conversation was essentially taken away from software developers—the people for whom Agile was created. After all, the Manifesto begins with (italics added), “*We* are uncovering better ways of developing software … .”

Agile thought leadership was increasingly controlled by those who wanted to advance an increasingly extreme behavioral agenda. Programmers speak up from time to time, but with trepidation. For example, a poster on Reddit wrote this: