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:

> *"I hate Scrum. There. I said it. Who else is joining me? Scum seems to take away all the joy of being an engineer. working on tasks decided by someone else, under a cadence that never stops. counting story points and 'velocity'. 'control' and priority set by the business - chop/change tasks. lack of career growth—snr/jnr engineers working on similar tasks."*[15]

The Scrum Master and Agile coach roles became so entrenched that organizations now assume that they need those roles. No one questions it. More important, no one questions the skills and knowledge that are needed for those roles to actually be effective.

The career of a Scrum Master or Agile coach relies on the perceived value that they provide. Since the experience of most was not in the technical realm, the natural evolution was to inflate the value of the nontechnical dimension of things. We saw a rise in Agile dogma, in which Agile or Scrum advocates—usually Agile coaches or Scrum Masters—would deride anyone who challenged Scrum or Agile practices. We have mentioned the term *Agile doubter*. These dogmatic people were such a problem that in a 10-year retrospective on the state of Agile, the British Computer Society referred to Scrumdamentalism—a play on the words *Scrum* and *fundamentalism*—as one of the main problems with the state of Agile.[16]

The Agile community suffered from groupthink: the community mostly spoke to itself. One of our editors described the community as insular. During the first decade of Agile, Agilists mostly read books that said "Agile" on the cover, but important ideas from other communities, such as those that study organization change or organizational psychology, did not permeate the Agile community. In the words of *Norwegian Wood*'s author Haruki Murakami, "If you only read the books that everyone else is reading, you can only think what everyone else is thinking."

The result was that thinking in the Agile community became highly constrained: the community would only tolerate ideas that either

modified existing Agile practices to be more extreme or added new nontechnical practices that were also extreme. Anything else was either ignored or dismissed as "not really Agile." So for the entire decade of the 2000s, Agile was confined largely to nontechnical thinking—an irony since Agile was created for software development.

Meanwhile, companies such as Google, Amazon, and Netflix had real problems to solve. They needed to be able to deploy to hundreds of millions of users many times a day, with confidence, and allow teams to make rapid changes and redeploy without delay. They needed agility in their software development and release processes.

So, they invented techniques to help them to achieve these things: techniques such as on-demand test environment creation, test-first integration testing (aka ATDD), containerization, container cluster management, and many other things. Jez Humble was the first who we know to have assembled these ideas into a coherent whole and publish a book about them, which he called *Continuous Delivery*, and the methods he described became known as *continuous delivery*, often abbreviated as CD. Gene Kim later wrote about CD ideas in his book *The Phoenix Project*.

Humble was an advocate of Agile methods and viewed himself as an Agilist, and he spoke about his work at Agile conferences. His talks were well attended, but few in the Agile community at large paid attention: what Humble was advocating was technical, so it was largely ignored by the large and growing cohort of nontechnical Agile coaches, particularly those who had built a career around Scrum. The established spokespeople for Agile—the popular Agile authors of the day—also largely ignored what Humble was talking about, preferring to pile onto the process topics that had been trending, and a few kept tweaking the now dated practices of XP. As a result, CD methods and related ideas became their own movement, which we know today as DevOps—a term that arose shortly before Humble's book was published.

The Agile community saw the rise of DevOps in the early 2010s and became alarmed. Agile was under threat from something new, so they claimed it as their own. Those who still advocated for XP claimed that DevOps was just XP in a more evolved form (not true). Agilists pointed to Humble's Agile origins but did not acknowledge that the majority of Agile coaches paid little attention to CD until DevOps rose to widespread awareness.

To be fair, there are many technical Agile coaches, even though they are a small minority. Those people indeed saw CD as continuation of the march of Agile ideas that began with XP and before. They were a fractional cohort, though.

Today if you ask an Agilist about DevOps, they usually think that DevOps is an evolution or outgrowth of Agile. That is not how it was, but it no longer matters. Agile is a set of ideas. DevOps is a set of ideas. Both sets of ideas, and many others, are extremely valuable in the context of product development.

The Introvert vs. Extrovert Problem

Earlier we posed the question, do these practices favor certain ways of working at the expense of others so that certain people benefit but others are at a disadvantage?

The group that authored the four values of the Agile Manifesto was pretty homogeneous. As we said, they were all English speaking, most were from the United States with a few from England and one from the Netherlands, all were men, and all were very experienced. They did not represent a "typical" team of programmers. It is reasonable to expect, therefore, that they had some collective biases that differ from how a typical team of programmers would feel and think.

Since most were experienced, one might assume that when they wrote that they value "individuals and interactions over processes and tools," they were thinking of themselves as capable individuals—individuals who are highly experienced and who have refined judgment about IT methods. Such individuals arguably need less oversight—less *process*.

Within the principles that some of the group came up with, one of them reads, "The best architectures, requirements, and designs emerge from self-organizing teams."

That one principle has resulted in a strong preference within the Agile community for "self-organization," that is, a team in which there is no designated leader. The authors of the Agile Manifesto were able to come up with four values in the course of a weekend, so they clearly were able to self-organize enough to accomplish that. Would they have been able to continue to be organized over a period of months to create a complex software product?

No one can say. The principle regarding self-organizing teams has been one of the most contentious statements of the Agile Manifesto. An article by one of us in LinkedIn about self-organizing teams received 45,000 reads in the first two weeks—two orders of magnitude higher than the normal read rate for that author.

What happens if you put a group of people together and tell them to "self-organize" to achieve a goal? Well, it depends.

You will read that phrase, "it depends," a lot in this book. We believe it is central to all of the issues pertaining to how groups of people should work together. In the words of Malcolm Gladwell, the author of *Blink*, the only answer that is always right is "it depends," and that is definitely true when dealing with groups of people.

Some of the factors affecting how well a group self-organizes are:

- **Personalities**—some personalities mix well, others less so.
- **Level of knowledge and experience**—do they know the job well enough to get it done without supervision?
- **Urgency**—how important is their goal? Do their lives depend on it? Or at the other extreme, is it something that someone else wants—something the team sees as inconsequential to them—and they are mere mercenaries?
- **Preparation**—have they been trained to work in a certain way so that everyone has predefined roles?
- **Proven history**—have they worked together before and shown that they can?

There are certainly other factors as well.

Some people work well in a group; others have trouble in a group and prefer to have their own task. The Agile community tends to dismiss the value of these "loners" or "lone wolves." Yet lone wolves have created some of the most impactful software. Linux is an example. It was created by Linus Torvalds, who is a self-proclaimed loner and introvert. Today he still oversees the evolution of Linux, but he does so in a room by himself, communicating with the thousands of Linux kernel developers involved entirely by email. Linux powers most of today's computers, from smartphones to most of the servers in the cloud. If you use Amazon, you use Linux. If you use an Android phone, you use Linux.

Introverts tend to shy away from groups. Groups are taxing for them. In contrast, extroverts seek out groups—groups energize them.

It follows then that in a team in which the prescribed form of communication is face-to-face and verbal, introverts would feel a lot more taxed than extroverts, who would actually find the frequent face-to-face discussions energizing.

One of the principles in the Agile Manifesto reads, "The most efficient and effective method of conveying information to and within a development team is face-to-face conversation."

This seems to clearly favor extroverts. Extroverts will enjoy resolving every issue by turning around and talking to others. They would enjoy convening an ad hoc meeting to discuss the issue. According to Noa Herz, a neuroscientist and neuropsychologist, "Group meetings, in which each participant contributes thoughts in a disorganised, dominance-based manner, can put introverts at a disadvantage."[17]

An Agile coach or Scrum Master is supposed to facilitate team meetings, but doing so can be challenging. If a few members are talking rapidly, a facilitator might feel that the members are "on a roll" and be hesitant to interfere.

An introvert—by definition—will find that situation emotionally taxing. Even if asked their opinion, they are not likely to keep the floor long enough to actually convince anyone. It is therefore reasonable to assume that introverts might prefer to first communicate in written form and meet in person only if it is necessary.

An interesting fact about programmers as a group is that they *think* they are extroverts.[18] Yet some psychological studies indicate that programmers actually tend to be introverts.[19] It is a complex issue, though, because other studies produce conflicting results on the question.

Regardless, there is still a problem, because while Agile appears to favor extroverts, the people at whom Agile methods are mostly targeted—software developers—have a high proportion of introverts, as a group, even though the actual percentage is unclear. Even if, say, 60% of programmers are extroverts—something that would be a counterintuitive career choice—forcing extrovert-favoring methods on the other 40% would be unfair and counterproductive.

It is also a problem because by sidelining introverts, teams suffer. According to Herz, "Entrepreneurial teams perform better when leadership is shared between individuals, but only if they have diverse personality traits. Moreover, teams dominated by extraverted members actually perform better under introverted leaders,[20] possibly due to their greater responsiveness to their employees' ideas."

How did the Agile Manifesto come to be written to favor extroverts? Were the authors of the Agile Manifesto extroverts? It is hard to say, but the manifesto's stated preference for face-to-face communication aligns with what one of the authors had been writing shortly before their meeting. Alistair Cockburn had written extensively about how, in his opinion, face-to-face communication is the "most effective" form of communication.[21]

So it is possible that one member of the group influenced the others—a member who seems to fit the profile of an extrovert, based on his high level of involvement in Agile community group initiatives.

Is it really true? Is face-to-face communication always best?

It depends. (There it is again.) We will dig into that question a great deal in Chapter 6, when we talk about how collaboration occurs for simple and complex issues.

What about the people who help Agile teams to define how they work? That is, Agile coaches and Scrum Masters? Do they tend to be introverts or extroverts?

We are not aware of a study on that question, but consider that in seeking such a job, one is looking for a role in which one is coordinating groups of people and encouraging face-to-face communication. It stands to reason that the job, as defined by Scrum and the Agile community, would attract people who like working with groups of people—in other words, extroverts. And it has been our experience that Agile coaches do, by and large, seem to be "people people," although they are as varied as the people in any profession, and no two are the same.

If it is true—if Agile coaches and Scrum Masters lean slightly extroverted—then we have a slightly extrovert-leaning population advising a somewhat introvert-leaning population on how to work. What could go wrong?

Ignoring whether an Agile coach or Scrum Master is an introvert or extrovert or something in between, and following default Agile practices that favor extroversion can be detrimental if those practices are not what are actually preferred by the team members.

Does this matter? Is there actually a problem? What impact might this have?

Let's look at common complaints that we have heard from programmers who work on Agile teams.

- "I can't focus! There are too many distractions and disruptions."
- "Too many meetings!"
- "I don't actually *want* to know what everyone else is working on!"
- "I don't actually *want* to self-organize! I want someone to *get us organized* so that I can code! I just want them to let me *code the way that I want to*."
- "I don't actually *want* to reach out to others for ad hoc discussions! I only do that as a last resort if I am really stuck."
- "Not enough attention is paid to technical issues!"

If these complaints are valid, it is a pretty bad state of affairs, because the true needs of programming teams are not being addressed by the Agile community; in fact, it is worse than that: programmers are being literally coerced, by organizations that adopt Scrum as a mandatory methodology,[22] into working in a way that is not suited to their personalities.

Coaches Should Not Assume

What has continued to amaze us is that if one asks an Agile coach, "Is Agile working?" the answer will usually be an emphatic yes followed by gushing about how productive teams are and how Agile has revolutionized software development. Yet if one asks the typical programmer—the very people who Agile was created for—they will have a different range of responses. In fact, it seems to us that most programmers have not had a very positive experience with Agile.

Studies at the University of Florida and the University of Notre Dame indicate that introverted team members tend to view their extroverted co-workers more negatively, whereas the extroverted co-workers do not seem to have that negative bias.[23] In other words, the introverts silently judged the extroverts negatively.

We think something like that might be happening among Agile teams. Agile coaches tend to be people-oriented, and therefore more extroverted than programmers on average, so they don't see a problem; programmers tend to be more introverted than average.[24] The Agile coaches think things are great. The programmers? Well, not so much.

The University of Florida and Notre Dame experiments might also indicate that extroverts tend to be relatively insensitive to how introverts are experiencing things, because if they were aware, they would probably not be so favorable in their own ratings of their introverted colleagues. That conclusion is also supported by a Yale study that indicated the introverts are generally more sensitive to the feelings of others.[25]

This is why coaches should not assume that what *they* would prefer is what their team will prefer. Ask! And pay attention to what is being written and said, even if it goes against Agile dogma. For example, in her book *Carved In Sand*, Cathryn Ramin writes the following on page 33:

> *". . . an endless stream of interruptions has become the norm. Researchers at the University of California, Irvine attempted to quantify the number of distractions and interruptions that occur among IT workers in a medium-sized office. They predicted that something would interfere with concentration every fifteen minutes, but on average, interruptions occurred every three minutes, and only two-thirds of the interrupted work was resumed on the same day."*

If this sounds like your organization, do you think that team members can focus? Ask them!

What about use of tools like Slack and Microsoft Teams? Are they too distracting? Consider this article in Nir & Far, "If Tech Is So Distracting, How Do Slack Employees Stay So Focused?":

> *"Slack management leads by example to encourage employees to take time to disconnect. In an interview with OpenView Labs,[26] Bill Macaitis, who served as Slack's chief revenue officer and chief marketing officer, states, "You need to have uninterrupted work time . . . . This is why—whether I'm dealing with Slack or email—I always block off time to go in and check messages and then return to uninterrupted work."[27]*

Perhaps do not assume that the team should be texting each other all day in these tools. Perhaps encourage them to disable desktop notifications and to not install the tool on their phones.

Maybe some people are bothered by these tools and by ambient noise, but others are not. Ask! Consider this post in Slashdot, "Why Office Noise Bothers Some People More Than Others":

> *"According to a 2015 survey of the most annoying office noises by Avanta Serviced Office Group, conversations were rated the most vexing, closely followed by coughing, sneezing and sniffing, loud phone voices, ringing phones and whistling. Why do we find it so hard to be around these everyday noises? . . . As the researchers suspected, all the students performed better in silence. But they also found that . . . the more extroverted they were, the less they were affected by noise."*[28]

Maybe some team members can work just fine in a team room, but others cannot. Do not assume that a team vote is the answer. Maybe a range of approaches should be considered. Ask!

Noise and pop-up alerts are not the only problem. Some people also find local movement distracting. That is why Panasonic has developed "horse blinders" for humans, specifically for use in open office layouts (Figure 1.1).[29]

Figure 1.1: Blinders to help people focus in "team rooms"

Source: (Photo from Japan Trends: www.japantrends.com/wear-space-panasonic-wearable-concentration/)

Are frequent "stopping by" discussions disruptive? For some people, yes. According to a study at George Mason University, being interrupted for only 60 seconds while concentrating can completely wipe one's short-term memory.[30]

Perhaps do not advise your team to just "stop by" each other's desk at any time. Perhaps people should have an "I'm thinking" sign on their desk, indicating that they do not want to be disturbed.

What about all the Agile ceremonies? Many developers think it is too much.[31] Perhaps instead of assuming that the team needs those, ask! According to an article in `Doist.com`,

> *"This trend toward near-constant communication means that the average knowledge worker must organize their workday around multiple meetings, with the time in between spent doing their work half-distractedly with one eye on email and Slack."*[32]

Talk to teams. Find out how they feel that they work best. Do not assume. Do not blindly follow someone else's methods.

Now What?

We can see that while there are some really powerful and important ideas behind the Agile movement, things are not quite right. What should we do?

Agile 2 is not intended to be a new set of ideas. The ideas of Agile 2 are all well-established, but they are not Agile doctrine. Many members of the Agile community have been using these ideas on an individual basis but usually describe these as implied nuances, or "bringing in ideas from other domains." These ideas pertain to leadership, to effective collaboration, to empowering people to work in a way that is best for them individually, and to applying context and judgment instead of just adopting an extreme practice. Agile 2 is about bringing these nuances explicitly into the Agile idea set and making them part of the Agile envelope.

In the next chapter, we will take a more comprehensive look at some problems of Agile to establish a better understanding of what areas of Agile need improvement. In later chapters, we will then look at models

for effective leadership, collaboration, product design, transformation, continuous delivery, and other ideas that are part of Agile 2, and we will conclude with examples of some specific problem domains through an Agile 2 lens.

Notes

1. www.youtube.com/watch?v=a-BOSpxYJ9M
2. ronjeffries.com/articles/018-01ff/abandon-1/
3. www.forbes.com/sites/cognitiveworld/2019/08/23/the-end-of-agile/
4. developers.slashdot.org/story/19/08/24/1748246/is-agile-becoming-less-and-less-relevant
5. goremotely.net/blog/agile-adoption/
6. www.extremeprogramming.org/
7. www.oreilly.com/library/view/making-software/9780596808310/ch17s01.html
8. en.wikipedia.org/wiki/Scrum_%28software_development%29#History
9. www.scrumguides.org/scrum-guide.html#team-dev
10. www.agilealliance.org/glossary/mob-programming/
11. www.fastcompany.com/3044389/a-first-look-at-facebooks-new-mothership-designed-by-frank-gehry
12. The larger racquets were not welcomed by many tennis enthusiasts, because the larger racquets encouraged sloppy strokes, and they also produced a faster serve, shifting the focus of the game to one's serve.
13. explore.digital.ai/state-of-agile/6th-annual-state-of-agile-report
14. www.youtube.com/watch?v=G_y2pNj0zZg&feature=youtu.be
15. www.reddit.com/r/devops/comments/i4cbwu/i_hate_scrum/
16. www.slideshare.net/jcasal1/20120419-agile-business conferencepptx, slide 11.
17. psyche.co/ideas/introverts-are-excluded-unfairly-in-an-extraverts-world
18. www.infoq.com/news/2013/02/Introverted-Intuitive-Logical/
19. ir.lib.uwo.ca/cgi/viewcontent.cgi?article=1005&context=electricalpub
20. psycnet.apa.org/record/2011-15936-006
21. www.researchgate.net/figure/14-Effectiveness-of-different-modes-of-communication_fig8_27296733

22. The Scrum authors refer to it as a *framework*, but Scrum perfectly fits the definition of *methodology* as a "body of methods, rules, and postulates employed by a discipline" (Merriam-Webster). A methodology can be extended, just as a framework can.
23. www.eurekalert.org/pub_releases/2014-12/osu-ics121614.php
24. ir.lib.uwo.ca/cgi/viewcontent.cgi?article=1005&context=electricalpub
25. www.inc.com/glenn-leibowitz/yale-psychologists-introverts-are-better-than-extroverts-at-performing-this-essential-leadership-skill.html
26. expand.openviewpartners.com/former-slack-cmo-bill-macaitis-on-how-slack-uses-slack-868ffb495b71
27. www.nirandfar.com/slack-use/
28. slashdot.org/story/363608
29. techcrunch.com/2018/10/17/open-offices-have-driven-panasonic-to-make-horse-blinders-for-humans/
30. www.dailymail.co.uk/sciencetech/article-2697466/Do-not-disturb-Being-interrupted-just-60-seconds-concentrating-completely-wipe-short-term-memory.html
31. medium.com/serious-scrum/time-spent-in-scrum-meetings-75e38b08d8
32. blog.doist.com/asynchronous-communication/