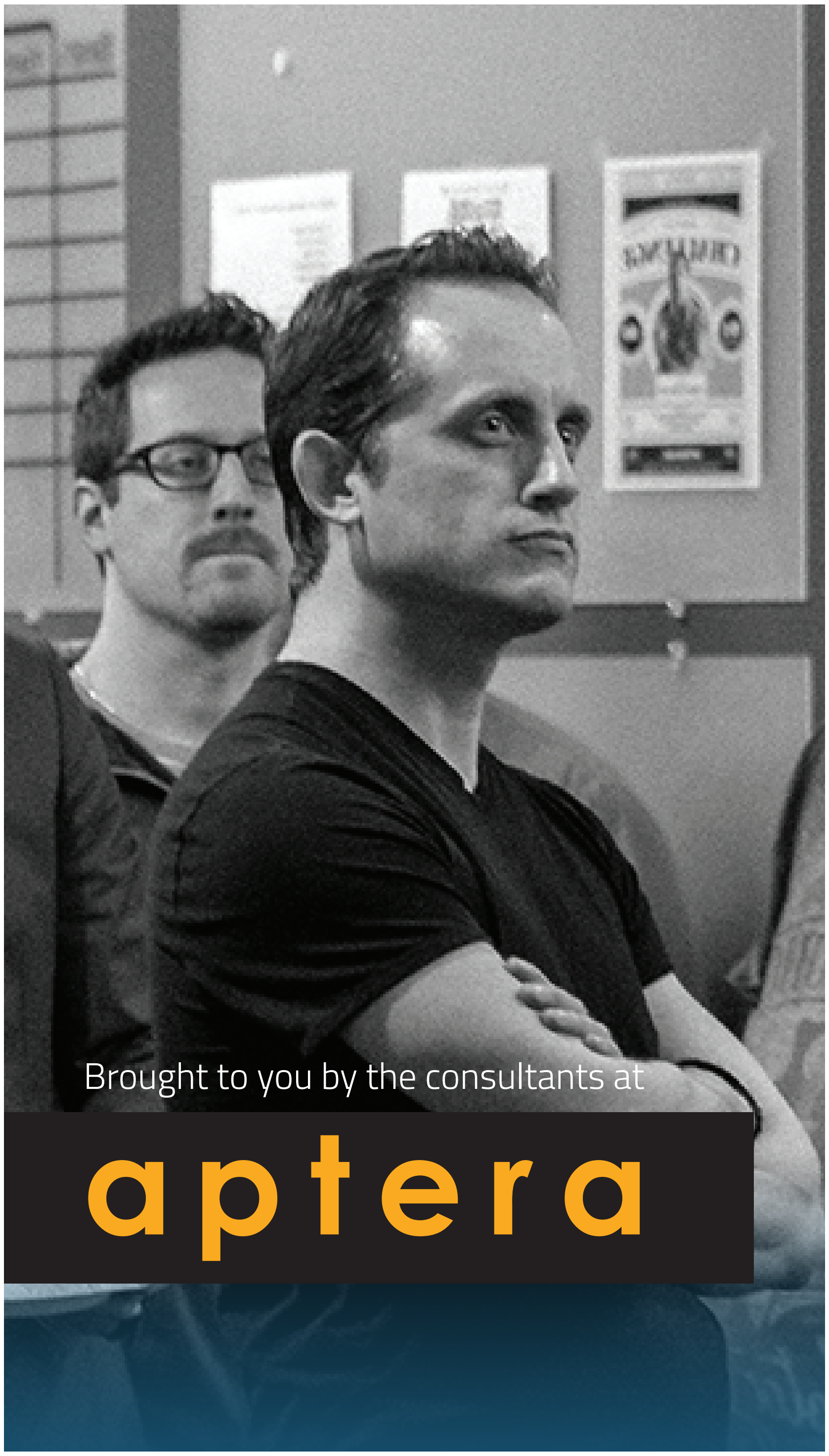
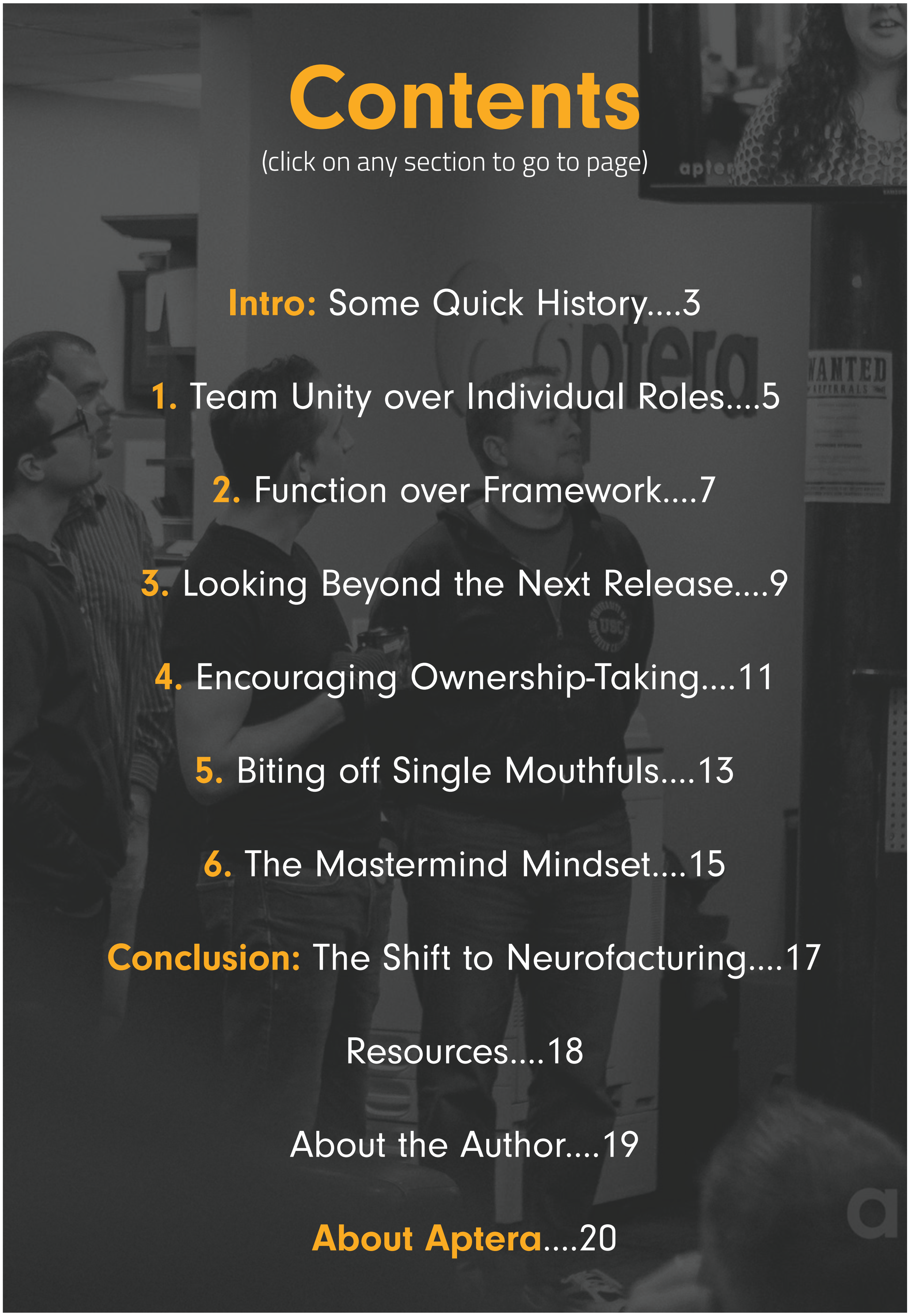


Brought to you by the consultants at

aptera

# 6 Qualities of Successful Scrum Teams



# Contents

(click on any section to go to page)

# Intro

## Some Quick History

Let's start with some quick history:

When Hirotaka Takeuchi and Ikujiro Nonaka studied manufacturers who in the early 1980s were innovating more rapidly and successfully than their competitors, they noticed a critical distinction in how these companies managed projects. Writing for Harvard Business Review in 1986, Takeuchi and Nonaka describe how innovators like Xerox, Honda, and Cannon eschew the "relay race" method of product development in favor of a "rugby" approach, "where a team tries to go the whole distance as a unit, passing the ball back and forth."

When Jeff Sutherland came across Takeuchi and Nonaka's article as part of his research into possible avenues for improving software development processes, he recognized the usefulness of both the principle itself and the image the authors used to illustrate it. Sutherland went on to label the technique he developed "scrum" in homage to their rugby metaphor. And that's how the methodology was introduced to the public by Sutherland (and his coauthor Ken Schwaber) in 1995. Today, scrum is one of the most commonly used of what are known as "agile" approaches to software development.

# Intro

## Some Quick History

*"It turns out many of the most common pitfalls developers run into in their attempts to go agile are the result of holdovers from a traditional manufacturing mindset."*

The transition Takeuchi and Nonaka were observing from a relay-race model of product development to the team-as-a-unit approach is emblematic of the wider shift from the assembly-line mentality of the early 20th Century to the emphasis on "cross-functional" teams that characterizes modern knowledge work (as well as modern manufacturing, as many of the key principles were originally adopted as part of the Toyota Production System).

Now why is this historical overview important in the context of any discussion on the qualities of successful scrum teams?

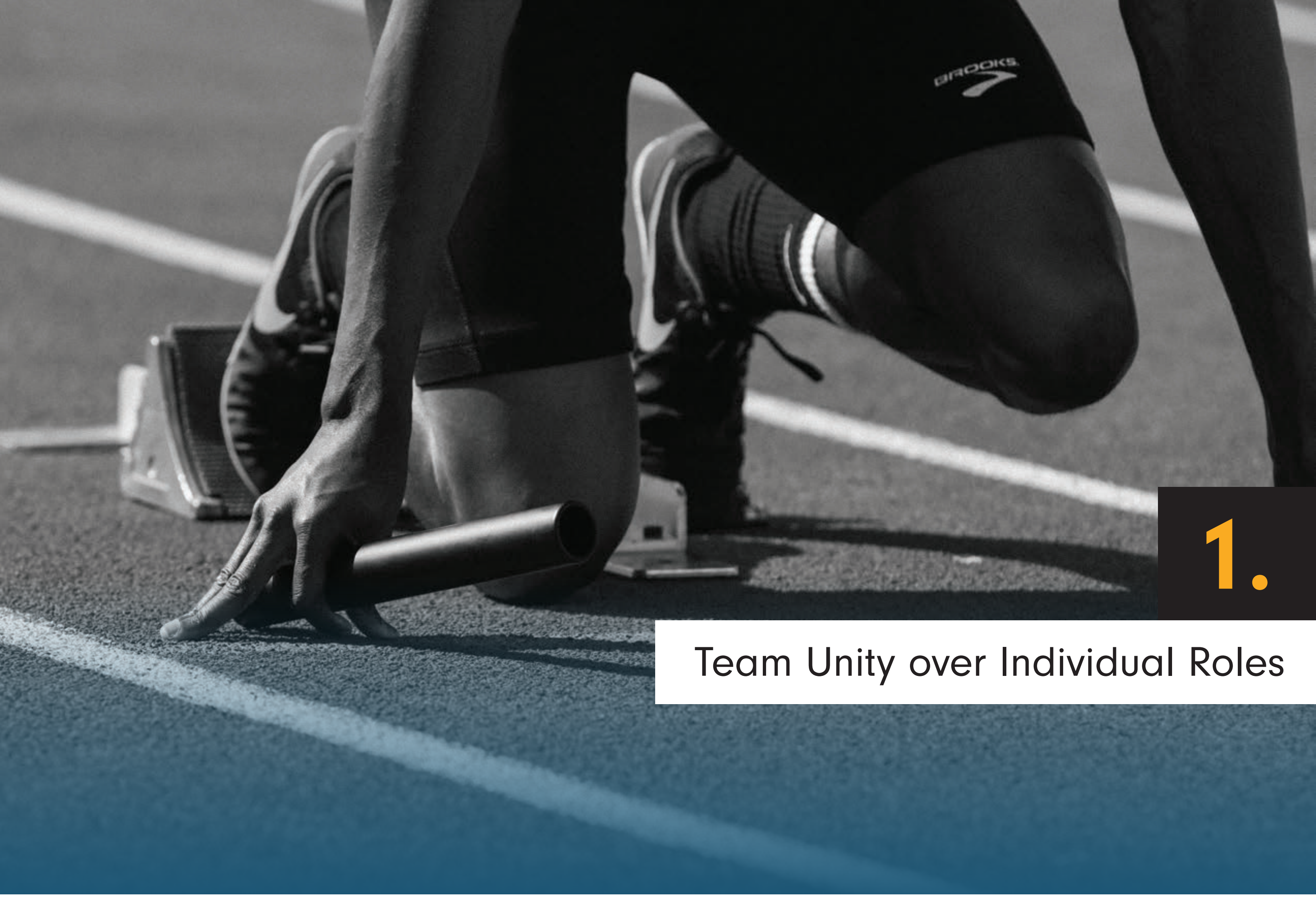
It turns out many of the most common pitfalls developers run into in their attempts to go agile are the result of holdovers from a traditional manufacturing mindset. The assembly-line or relay-race approach to product development calls for top-down planning and the assignment of rigidly defined roles. It emphasizes speed and quantity—i.e. productivity—over creativity and overall impact.

It's a mindset you almost can't help falling back on whenever the pressure gets cranked up.

The members of our software team here at Aptera have enjoyed countless opportunities to see the inner workings of other companies' development practices, both through staff augmentation contracts and as part of our Process Improvement offering. Many (but not all) of these engagements involve efforts to adopt agile principles more effectively, so the team has arrived at a good sense of what commonly goes wrong to undermine these efforts, along with a good sense of what strengths a scrum team needs to be successful.

We've narrowed down the list to just six of the most important qualities you should work on cultivating to help your team achieve its optimal levels of performance.

So let's jump right in.



## 1. Team Unity over Individual Roles

Many development teams read the term "cross-functional" as meaning they should appoint a member to each role as required by the project. And some specialization is of course advantageous, or at any rate unavoidable; you will, for instance, probably want a Quality Assurance expert on your team, who it wouldn't make much sense to have working on something other than testing.

But the farther you go in the direction of thinking and operating as a collection of individuals instead of a single functional unit, the harder it's going to be to achieve the type of group cohesion characteristic of the best agile teams. When you're on a deadline, though, it's tough to be satisfied with the tight grouping of team members slowly advancing the ball down the field. You want someone to break free from the pack and make much faster progress.

The problem that arises from such subdivision of tasks in software development projects is that it encourages a narrowing of focus (and you narrow it even further when you measure productivity at every stage and incentivize increased velocity—more on these below).

## Team Unity over Individual Roles

*“What the most successful teams do instead is involve the QA expert in the project from the beginning”*

What you end up doing is unwittingly setting workers with varying goals against each other, with no one keeping an eye to the big picture. The people you assign to the different roles eventually find themselves working at cross purposes.

The developer is told to write more code in less time, for instance, while the Quality Assurance expert is told to make sure the resulting code has no bugs. So the developer works faster, creating more buggy code, and the QA expert has to spend more time cleaning up the mess.

What the most successful teams do instead is involve the QA expert in the project from the beginning—following the maxim, “Test early, test often.” That way, the testing can be conceived with an eye to the business function the feature will ultimately serve.

Again, you want to get away from the relay-race approach where one team member completes her portion of the task and then passes it on to the next team member. All the team members should instead be putting their heads together and driving the ball down the field.

## 2.

## Function over Framework

When you're first learning about scrum, you have little choice but to start by practicing your application of the framework:

You arrange user stories into a product backlog and take them on in order of priority. You develop features to support the user stories over the course of two-week sprints (or some similar interval) according to your team's capacity. Each day, the team meets for standups to discuss obstacles to progress. At the end of the sprint, you demo the features to the product owner so you can incorporate the resulting feedback into revisions of the backlog. Then you meet again for a sprint retrospective to discuss ways to improve your processes. Then it's time for the next sprint planning meeting—and so on, until the project is complete.

## 2.

## Function over Framework

That's the framework.

What you have to keep in mind is that when Sutherland and Schwaber developed this framework, they were attempting to find the best way to get at a set of principles. It's those principles, not any specific practice or set of practices, that make your project agile.

*"Scrum is your conveyance, not your destination."*

Function over framework, principles over practices—sounds easy, right? Here's the underlying reality many scrum teams overlook: it takes time and practice and discipline to achieve agility. The scrum framework, like any of the other agile methodologies, is properly thought of as a tool to help you make progress in your journey.

Scrum is your conveyance, not your destination.

At first, it's only natural to emphasize strict adherence to the framework, with all its terms and ceremonies and time-boxes. But what tends to happen is when something starts to go wrong—when a deadline looms or when a client is unhappy—the adherence gets even stricter. Before long, you're basically back on an assembly line, just with new terminology for the project stages.

Examples of letting framework overshadow function include:

- Making mad dashes to check off all the items on a list of acceptance criteria before the end of the sprint (discussed below)
- Overburdening team members with meetings that contribute nothing of value
- Treating the meetings, open work spaces, and shared workflow visualizations as an invitation to aggressive transparency and productivity-tracking.

Remember, agility requires self-management and the space for creativity. Under intense pressure, though, many managers succumb to the urge to use scrum as a mask for their efforts at micromanagement. This in turn shifts the intense pressure onto the team members, with predictably counterproductive results.

## 3. Looking Beyond the Next Release

One of the great things about sprints is that they allow you to concentrate on only as many things at one time as you can realistically handle. Nearly everyone you meet will tell you they're pretty good at multitasking, but they're delusional. Nobody is good at multitasking. Scrum works on the principle that your team will work faster and produce higher-quality software if it starts by breaking down large, complex development projects into more manageable user stories and taking them on in manageable chunks.

But then comes the pressure from on high to squeeze as much velocity out of the team as possible. So a measure that's supposed to be used for making estimates is transformed into a performance-tracking metric.

Thus begins the Race to Release.

Instead of taking the time to consider the broader business context or related technical aspects of the feature under development, each team member focuses exclusively on what's expected for the next deliverable.

## Looking Beyond the Next Release

*"...it's easy to make decisions that are good for the short-term but catastrophic for the overall project."*

But isn't this what they're supposed to do?

If the team members are thinking about nothing but the functional requirements for the feature they're developing, then they're not being very agile. The whole reason scrum replaces the traditional requirements list with a backlog of user stories is that your goal is to deliver value to users—to help them meet some goal or overcome some challenge. The goal isn't to check all the items off any list.

An example scenario: developers may be spending a lot of time within each sprint on tasks that are repetitive. They can either continue to perform these tasks manually, or they can take the time to automate them. Do you stop to spin up a unit test, though, or do you press on to achieve the full list of functions for the next release? What about an integration test? You could either continue to do it manually for each sprint—or you could just skip it altogether.

When the clock's ticking and all you care about is completing what's expected of you before the next deadline, it's easy to make decisions that are good for the short-term but catastrophic for the overall project.

Meanwhile, if you don't have someone breathing down your neck and telling you to move faster, you'll realize you can save tons of time over future sprints if you devote some time in this particular sprint to automation. And how much time are you saving by doing tests at every stage, thus avoiding a steady accumulation of technical debt?

The main distinction here isn't between focusing on a few things versus many; you're still only focusing on a few things at a given time—well, those few things and their wider implications. The crux is that you want to focus on quality at each stage rather than speed or velocity, or even productivity.

You're not really saving any time if you're sacrificing quality in your race to the next release; instead, you're digging yourself into a hole you'll eventually have to dig yourself out of.

## 4.

## Encouraging Ownership-Taking

In the best scrum teams, you're just as likely to see a junior developer's idea taken up and implemented as an idea from a manager or a more senior developer. Think of some of your own team's projects: where do the plans and ideas come from? Who's really driving them?

Now consider another point: a team member who's merely doing what she's told is going to define success as fulfilling the given set of instructions.

That's exactly what you're trying to get away from.

Ideally, you want to encourage team members to run with any ideas they're really stoked about (within reason of course). If those ideas are going to fail, they should fail fast—and you'll know because your team took the time to set up a bunch of automated unit tests and integration tests and user acceptance tests, etc.

## 4.

## Encouraging Ownership-Taking

Failed ideas aren't the worst thing in the world. The worst thing in the world—well, in the context of your efforts to be agile anyway—is a bunch of developers mindlessly following instructions.

When everyone's simply following instructions, the only source of ideas is the instruction-giver. However smart this one person may be, the group as a whole will almost always be much smarter. Moreover, the developer working on something every day can be counted on to have more insight into what she's working on than anyone in a managerial role.

It's vital for you to foster a free flow of ideas because, for one, software development is much more about creative problem-solving than it is about meeting a pre-existing set of functional requirements. But it's also because you want each team member, as well as the team as a whole, to be invested in the project's success.

It's the difference between someone saying, "I'm doing everything you ask; if it's not working, that's on you," and someone saying, "This doesn't seem to be working; what can we do to improve it?"

*"Failed ideas aren't the worst thing in the world."*

## 5.

## Biting off Single Mouthfuls

Whenever you ask anyone on our software team what the most common problem is for the scrum teams they've been called in to help, you get remarkably similar answers. "They're just in a constant state of having their hair on fire" is one example that conveys the gist.

Team members are dealing with long lists of bugs, or making glacial progress on a staggeringly long list of features to develop. Deadlines loom. Budgets evaporate. Leadership is in full crisis mode. And the team members themselves range from disengaged to irritated and at each other's throats. There's not a lot of agility to be found in these situations.

But all these problems are the outcomes. You still have to ask what the cause is. One of the culprits is none other than our friend Jeff Sutherland, who mischievously subtitled his book on scrum, "The Art of Doing Twice the Work in Half the Time."

People who already know something about scrum understand what he means—you get much more done, much more quickly, when you concentrate on doing one thing at a time. But everyone else comes away with the wrong impression: that you should be able to increase your team's productivity by a factor of four.

## Biting off Single Mouthfuls

*“Quality is defined as whatever offers the most value to the client.”*

Okay, so it’s not really fair to place the blame solely on Sutherland. There’s not exactly a shortage of people pressuring us to be more productive. Clients want their applications sooner and cheaper. Leadership wants your team to get to the next project. And you yourself like to keep making valuable contributions at a steady clip—leaving plenty of time at the end of the day or week to decompress.

The key point here is that Sutherland is talking about getting more done by working as a team and focusing on a narrower range of concerns until those concerns are fully addressed and then moving on to the next set. But far too few managers ever get past the part about getting more work done.

As scrum teams get more and more experience, they develop an intuitive sense of how much work they can take on successfully at any given time. Beginners, on the other hand, nearly always overestimate their capacity, thinking that because they’re applying the scrum framework they should be able to handle a larger workload (but see chapter 2).

To avoid biting off more than your team can chew, keep in mind that being agile is about doing better work so you don’t end up having to do as much work. Aggressively fixating on velocity or productivity completely defeats the purpose.

The main thing to focus on is quality. And quality is defined as whatever offers the most value to the client.

Hair-on-Fire Syndrome can also be tied directly to the traditional Request for Proposals process. That’s why many of the best development companies have stopped responding to such requests and instead encourage methods better suited to choosing among agile candidate firms.

*(See the Resources page for a link to our e-book:* ***Choosing a Custom Software Development Firm in the Age of Scrum****.)*

# 6. The Mastermind Mindset

Once the team members have gotten to know each other, it's usually somewhat easy for each one to anticipate the thoughts and reactions of the others. It's one thing, though, to be able to answer the question, "What will Sara think of this new automated test?" and another thing entirely to start thinking in terms of how the group as a whole will respond.

The first chapter of this e-book is about the difference between treating your team as a single unit and treating it as a collection of individuals fulfilling their own specified roles. Here we're taking that a step further. Not only are the team members working together—at the same time, on the same features—they are functioning as a whole that's greater than the sum of its parts.

So it's no longer a matter of "You do your part and I'll do mine." It's rather "What part are we taking on next?"

# 6.

## The Mastermind Mindset

*"..isn't the ultimate goal to increase productivity while maintaining some minimum acceptable level of quality?*

The top-performing scrum teams start to operate a little like the human brain; upon close inspection you can see it's made up of individual neurons, but they're all working so closely together that the result is a single, seemingly seamless experience of interacting with the world.

Scrum teams have come up with lots of ways to leverage collaboration and foster greater cohesion:

- Pair Coding
- Mob Coding
- Swarm Coding

It's up to you to choose which methods work best for your team, but the important thing is that the members not be off in their separate spaces, each staring into a different screen. (Though, of course, some individual work at certain points in the project is fine.)

Like the Zen states so many agile advocates are fond of referencing, this quality of being able to come together to form a mastermind is abstract and, as far as we can tell, fleeting. Even the best scrum teams only achieve such levels of cohesion under optimal conditions, and you shouldn't expect to maintain it on an uninterrupted basis. But it's still worth considering because it gives you an idea what you're ultimately trying to accomplish.

But wait... when it really comes down to it, isn't the ultimate goal to increase productivity while maintaining some minimum acceptable level of quality? You have clients, after all, and they're not paying to have your team sit around meditating and mind-melding.

The counterintuitive insight you have to work at wrapping your mind around here is that while increased productivity is still one of the main outcomes you're seeking, focusing on it and holding it up as your chief aim will nearly always guarantee failure in your attempts to bring it about.

# Conclusion

## The Shift to Neurofacturing

The reason so many scrum teams run into these sticking points is that most of us were brought up with the remnants of a culture that rose up around early 20th Century manufacturing. That's where this idea of measuring units of work completed per interval of time first established itself. But, with mass production, you already know exactly what the finished product is supposed to look like, so it's easy to subdivide tasks and assign them to workers in strictly defined roles.

Today's knowledge work, and software development in particular, is not at all suited to this type of top-down, command-and-control management. In the first decades of the 21st Century, we're no longer doing that type of manufacturing—even automakers are using newer, lean approaches. What we're doing is more like neurofacturing, making things with our ideas. And, when it comes to ideas, quality and impact should always trump productivity as your central aims.

**If you'd like to delve deeper into the ramifications of this shift away from applying traditional manufacturing mindsets to software development, check out our e-book: The 4 Things Every Successful Scrum and DevOps Team Understands about Knowledge Work.**

Click Here to Download

**If you are ready to get started,**

contact a consultant and schedule a meeting

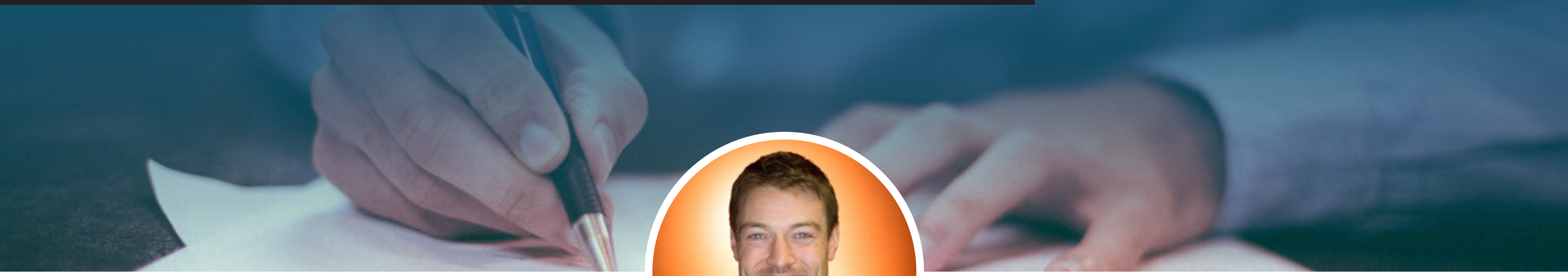
# Resources

*"The Secret History of Agile Innovation," by Darrell K. Rigby, Jeff Sutherland, and Hirotaka Takeuchi, Harvard Business Review, April 20, 2016:*
*https://hbr.org/2016/04/the-secret-history-of-agile-innovation*

*"HBR's Embrace of Agile," by Steve Denning, Forbes, April 21, 2016:*
http://www.forbes.com/sites/stevedenning/2016/04/21/hbrs-embrace-of-agile/#33d28c2527fe

*"Why the Most Successful Leaders Resist Their Instincts when Managing Agile Development Teams," by Dennis Junk, The Mark, Jan. 11, 2017:*
http://blog.apterainc.com/why-the-most-successful-leaders-resist-their-instincts-when-managing-agile-development-teams

# About the Authors



**Dennis Junk:** djunk@apterainc.com

I'm Aptera's Content Strategist. I've been writing about tech and marketing for 5 years and have certifications from HubSpot and The Content Marketing Institute. A big science and literature geek, I taught college rhetoric and composition while I was still busy going to school for way too long, earning bachelor's degrees in anthropology and psychology, along with a master's in British and American literature.

**Jon Fazzaro:** jfazzaro@apterainc.com

Jon took the software thing pro over a decade ago, and has been slinging code with Aptera since 2008. These days, he may or may not be unhealthily consumed with building sustainable software, and with building teams that build sustainable software. Whatever you do, don't follow @jonfazzaro on Twitter.

# About Aptera

Aptera is a results-driven digital marketing and technology firm, specializing in driving growth, increasing revenue, and demonstrating ROI for our client partners. We do this by breaking down the traditional divides between business areas, seamlessly leveraging digital marketing strategies, agile development processes, and advanced IT expertise. We began in 2003 as a homegrown Midwest provider of tech solutions and strategy, but our company has since expanded beyond our proud Fort Wayne roots to now include offices in Indianapolis and Nashville. All along the way, our ability to use business strategy and tech together to make our clients successful not only fueled our own growth, but has also made us an award-winning agency and top-tier Sitefinity, HubSpot and Microsoft partner.

aptera
Results-Driven Digital Marketing & Technology Experts
apterainc.com
Click on the banner to learn more about aptera