CHAPTER 10
The Agile Coach

You've learned about Scrum, XP, Lean, and Kanban. You know what they have in common, and understand what they achieve. If you work with other people to build software, then you've spotted at least a few things—some practices, ideas, attitude changes—that will help your team.

OK! Now, go ahead and do it. Make your team go agile. Right now!

That doesn't seem quite realistic yet, does it? There's a big difference between reading about values, principles, mindsets, and practices in a book and actually changing the way that a team works.

Some teams are able to take a book on Scrum or XP, adopt the practices, and immediately see great results. After reading the first nine chapters of this book, you should recognize why: those teams already have a mindset that's compatible with the values and principles of the Agile manifesto and the methodology. To a compatible team like that, adopting agile feels easy because the individual people on the team don't have to change the way that they think about their work. So if you already have a mindset that's compatible with the agile methodology you're trying to adopt, you're much more likely to have a successful adoption.

But what if you don't already have a mindset that works for Scrum, XP, or another agile methodology? What if you're working in an environment where it's difficult to succeed with the agile values? What if individual contributions on your team are rewarded far more than team effort? What if mistakes are punished harshly? What if you're in an environment that stifles innovation, or where your team has no access to the customers, users, or other people who can help you understand what software you're building? These are all barriers to agile adoption.

This is where an **agile coach** comes in. An agile coach is someone who helps a team adopt agile. He or she will help each person on the team learn a new attitude and

mindset, and get past the mental, emotional, and technical barriers that prevent the team from adopting an agile methodology. Agile coaches work with every person on the team to help him or her understand not just the "how" of the new practices that they're being asked to implement, but also the "why." A coach will help the team overcome the natural dislike—and even fear—of change that happens to anyone who is asked to try something new at work.

We've seen many examples throughout this book of people getting better-than-not-doing-it results: a team adopts the practices of an agile methodology, but the team members only get a marginal improvement because they don't really change the way that they think about the way they work, or their attitude toward working together to building software. In other words, *your team needs an agile mindset to get good results with an agile methodology*. Agile has the values and principles of the Agile Manifesto to help teams get into the right mindset, and each methodology comes with its own values and principles for the same reason. A team gets the best results from an agile adoption when each person gets into a mindset that's compatible with the values and principles of agile, and with the specific methodology that they're adopting.

The goal of the agile coach is to *help the team attain a better, more agile mindset*. A good coach helps the team choose a methodology that best fits their existing mindset, and introduces them to the values, principles, and practices of a methodology in a way that works for them. The coach will help the team adopt its practices, and then use those practices to help the team learn and internalize the values and principles, to slowly change their attitude, and to get into the right mindset so that they go beyond simply getting better-than-not-doing-it results.

In this chapter, you'll learn about agile coaching: how teams learn, how an agile coach can help a team change their mindset so that they can more easily adopt an agile methodology, and how that coach can help you and your team become more agile.

Narrative: a team working on a mobile phone camera app at a company that was bought by a large Internet conglomerate

Catherine – a developer

Timothy – another developer

Dan – their boss

Act III: Just One More Thing (Again?!)...

"Hey, Cathy! Do you have a minute? I've got great news."

Catherine groaned silently as her boss, Dan, called her into his office. She thought to herself, *I'll never get used to that*. She steeled herself for the "great news" as she sat down.

"You've done a fantastic job with this whole agile thing. Really, just fantastic."

Catherine thought back over the last eight months that she and Timothy had spent using Kanban to improve their process. She'd identified several bottlenecks in their process, especially the big one where Dan and other managers would drop new features on them seemingly at random. An amazing thing happened when she added a queue: as they started fighting and jostling for position on the queue, they slowly stopped demanding that she and Tim do more work than they were physically capable of.

In fact, now that she thought about it, this was the first time in months that Dan had called her into his office with "great news"—which usually meant that he had a critical last-minute change that would completely derail her project.

"I'm glad to hear you're happy, Dan," said Catherine. She only meant it half-sarcastically.

Dan continued, "And people in our parent company have noticed. They've been reading about this agile stuff, and now that we've got some success under our belt, they want us to teach other teams how to do it."

Catherine said, "Wait a minute. So you're asking me to... what? Coach them?"

"Exactly," said Dan.

"I have no idea how to be a coach," replied Catherine.

Dan looked her straight in the eye. "This isn't optional, Cathy. Come on, just buck up and do it."

Coaches Understand Why People Don't Always Want to Change

> Most people in your organization are trying to do a good job. They want their peers and supervisors to see that they are good at performing the tasks assigned to them. When someone has developed a level of comfort and familiarity with his job, the last thing he wants is to have someone come along and make him adopt an entirely new way of doing things.
>
> —Andrew Stellman and Jennifer Greene, *Applied Software Project Management*

Agile coaches spend most of their time helping people on teams change the way that they work. This is challenging for both the coach and the team, because only the coach sees the big picture. To the people on the team, they're being asked to adopt new rules about how they work, but they don't necessarily know why they're doing that.

There are many ways that a well-intentioned team trying to adopt a practice can alter it so that it doesn't really work anymore. For example, we saw in Chapter 5 that teams will often turn the Daily Scrum into a daily status meeting. An important goal of the Daily Scrum is to replace command-and-control project management with self-organization; the three questions that each person asks during the meeting are aimed at giving the team control over their own project plan. But many teams that attempt to adopt Scrum end up just using it as a daily meeting for team members to report their individual statuses, in which the Scrum Master acts as a de facto project manager and assigns work to them.

Similarly, some teams add stories to business requirements documents, but otherwise treat those documents as specifications in the same way most waterfall teams do. They're still doing big requirements up front (BRUF) development, just with user stories added. Or instead of doing test-driven development, some teams attempting to adopt XP will just make sure that they have very high code coverage for the tests that they develop after they build the code—which means the tests don't have any impact on the design, because it's complete by the time the tests are written.

In all of these cases, the people on these teams are genuinely trying to adopt agile. But in each of these examples, the people on the team don't really understand how those practices fit into a larger ecosystem of a methodology. So instead of trying to change the way that they work, they *focus on the part of the practice that feels familiar*. And why should we expect any different? A team that has only known command-and-control project management has no experience with self-organization, and no context to help them understand the Daily Scrum beyond what they already know.

And let's give credit where credit is due. Most of the teams adopting agile already build software, and have some success. (Completely dysfunctional teams rarely have a manager who is open-minded enough to let them try agile in the first place.) They're naturally looking to make small, incremental changes, because they don't want to break what's currently working.

This leads to one of the biggest roadblocks for agile adoption: when a team member thinks, "I've seen agile, I've adopted the practices that are familiar to me, my team is working better than before, and that's enough for me." This is what we've been calling better-than-not-doing-it results, and it's what causes many people to dismiss agile by reducing it to specific, marginal improvements that are a letdown from the excitement and "hype."

Why do team members so often insist on adopting only the practices that seem familiar to them, and reject any practice that doesn't immediately translate to something that they're doing today?

Because every new practice is a change, and there's always a chance that any change won't work out. And when changes at the office don't work out, people get fired.

This is something that every agile coach needs to keep in the front of his or her mind. The job of coaching is to help teams to change, and change can cause an outsized—but *entirely rational*—emotional response from the people being asked to change. Why? Because *work is how you pay for your life and feed your family*.

When we're asked to learn and do new things at work, if we don't feel like we can master them quickly and easily, it causes us severe emotional discomfort. There's a basic hunter-gatherer part of our minds that thinks, "Yesterday I knew that I could do my job and bring home food for my family, but today I can't be sure of that anymore." This is an important reason why being expected to learn new things at work can cause anxiety and seemingly irrational reactions (which, in this context, don't really seem all that irrational).

Another reason why people push back against a change at work and gravitate toward what they already understand is that they don't feel like they have time to sit and think through all the reasons behind it. For example, it's very common for teams to adopt agile by simply trying it on a pilot project. This often starts when one person has read a book about agile, and is now leading the whole team through their adoption—while at the same time, dealing with deadlines, bugs, conflicts, changing requirements, and all of the other things that happen during a typical project. This is not an ideal proving ground for a completely new way of thinking about work. People will do the best they can, but if there's something they don't quite get, they'll keep the label "agile" and the names of the methodology and practices, but little else will change about how they work. Milestones in their old project plan are now labeled "Sprint," or maybe someone puts up a task board that never really impacts the work being done. In the end, the team will just do what they did before, because they know that it worked in the past, and there's a deadline.

When a team uses the names of agile practices but doesn't change the way they work, it's not hard to see why the team members quickly become disillusioned with the ideas behind agile. To them, it's a reasonable assumption to think that agile is just another name for whatever it is they're currently doing. They think that they've adopted agile, but they haven't changed the way they work, so they get the same results.

The people on the team will decide that agile doesn't work, without ever realizing that they never actually saw anything that resembles agile in real life. They have this reaction because they're being asked to change how they work without understanding

why, and without someone to help them through the change without compromising the integrity of the new practices and ideas.

This is why teams need agile coaches. An important job of the agile coach is to recognize when people are being asked to do something new, and to help them feel comfortable with the change. The coach needs to give that person context for the change, so they understand *why* (and not just *what*). This helps the team actually change the way they work, rather than simply taking the name of something from an agile methodology and attaching it to a practice that they already do.

For example, a good coach will explain—in language that everyone understands—how the Daily Scrum helps the team self-organize, how test-driven development helps the team think functionally and move toward incremental design, or how user stories help everyone on the team understand the perspective of the people using the software. The coach helps everyone go beyond just adopting new rules, so they start to see where they're going and what these new things will eventually give them.

Coaches Listen for Warning Signs That the Team Is Having Trouble with a Change

If you spend time coaching many teams, you'll hear many of the same sentiments repeated from different people. Here are a few things that team members say that could indicate that they're uncomfortable with a change, and what you can do when you encounter them—which is useful for seeing things from the coaching perspective, even if you're not an agile coach:

"We already build software well. Why are you telling me to do something different?"
: It's hard to argue with success. If you're working with a team that has a history of getting software out the door, they have a right to know why they need to change at all—and it's not enough to tell them that the boss said so, because that will just undermine their morale. As a coach, you need to stay positive about the work that the team did in the past. But don't be shy about pointing out problems that they ran into. The practices in every agile methodology are aimed at fixing problems that teams have; if you can help your team understand why those problems are happening and give them solutions, they're much more likely to accept the change.

"This is far too risky."
: This is a very common response to agile, especially to someone who's used to a command-and-control style of project management. Where are all the buffers? Where's the risk register? Where's all the extra bureaucracy that slows down the project and gives me plenty of CYA? A project plan can be opaque, which can make teams comfortable: they don't need to share details beyond vague milestones, and can build in buffers to reduce and even eliminate variability. When you use status reports as the primary measure of software progress, you can con-

trol the message that you give to the users and customers. To a team that's used to these things, agile can feel very risky. In Chapter 3, we learned about the agile principle that working software is the primary measure of project progress. If something goes wrong on an agile team, everyone knows. Your job as an agile coach is to help the managers, users, and customers to feel comfortable with this undiluted measure of progress—and give the team a safe environment where they're allowed to fail today, as long as they learn from it tomorrow. If this isn't realistic today, then your job is to help everyone, especially the boss, to set a goal to change the team's climate and attitude in the future.

"Pair programming (or test-driven development, or some other practice) just isn't going to work for me."

A developer used to working alone might feel in his gut that pair programming will slow him down. He may even have a legitimate point—if he works on a team (or for a boss) with a mindset in which mistakes are simply unacceptable, then it's reasonable for him to feel uncomfortable having another person watch him code. And anyone who's taught a teenager to drive can relate to the uncomfortable feeling that a senior team member might have thinking about letting a junior team member control the keyboard for their pair. There are a lot of reasons that people can rationally feel uncomfortable with these practices, especially when the team has a mindset that doesn't match the practice. A good agile coach will help the team first choose the practices that are compatible with their mindset. As the team members start working those practices, they'll see the benefits and understand how and why the practices work. With guidance from a good coach, everyone will naturally start to shift toward a more agile mindset.

"Agile doesn't work for a business like ours."

People often say this because they're used to building large and detailed plans or designs before work begins, and they can't imagine working any other way. Command-and-control project managers and bosses get a lot of comfort from having the scope, requirements, and project plan on paper before any work starts. And in the same way, architects and development leads are reassured when the entire design for the system is on paper before the first line of code is written. Now they're being asked to trust the team to make decisions at the last responsible moment—and that means giving up that control. So they'll say something like, "Our business is very complex. People in other companies may be able to jump right into projects, but because of how our particular business works, we need to do all of this planning and design ahead of time." The truth is, every business is complicated, and every project needs analysis and planning. A good agile coach will help the managers, businesspeople, and development leads to see that teams are much better at capturing that complexity when they're allowed to divide the project into smaller pieces, and have the freedom to make decisions at the last responsible moment.

"This is exactly like what we're already doing, just with a different name."

This is one of the most common reasons that people reject agile methodologies. They'll find ways to keep doing what they're doing today, and just choose new names for them that match agile practices they've read or heard about. This can make all of agile seem trivial—and even wrong, if they've decided to give the agile name to a practice that failed for them. In fact, if you do a web search for a term like "agile sucks" you'll find people who have done exactly this, denouncing agile as "hype" without substance because it's just another name for old waterfall practices. They'll even call agile an "obfuscation" of simple, commonsense software development. As a coach, you need to recognize that this isn't done with malice or ill intent; it's just a natural reaction from someone who's never actually seen agile (but thinks he has). Your job, as a coach, is to help your team understand that agile really is different from what they're doing, and that it doesn't suck.

Coaches Understand How People Learn

> One good model for mastering anything (if that's possible) comes from martial arts. A martial arts student progresses through three stages of proficiency called Shu Ha Ri. Shu: Follow the rule. Ha: Break the rule. Ri: Be the rule.
>
> —Lyssa Adkins, *Coaching Agile Teams: A Companion for ScrumMasters*
>
> Kent Beck, author of *Extreme Programming Explained*, described the use of Extreme Programming (XP) using similar levels. Asked about XP and the five levels of the Software Engineering Institute's "Capability Maturity Model," he replied with XP's three levels of maturity:
>
> 1. Do everything as written.
> 2. After having done that, experiment with variations in the rules.
> 3. Eventually, don't care if you are doing XP or not.
>
> —Alistair Cockburn, *Agile Software Development: The Cooperative Game,* 2nd Edition

Back in Chapter 2, you learned about how different people see agile differently. Like each of the blind men who encountered the elephant, different people will initially come to their own answers to the question, "What is agile?"

A programmer might see XP's programming practices like test-driven development and incremental design and decide that agile is about software programming, design, and architecture. A project manager, on the other hand, might decide that agile is about improving project management practices after reading about Scrum's sprints, planning, backlog, and retrospectives.

It's taken us hundreds of pages to explore Scrum, XP, Lean, and Kanban. We've talked about mindsets, values, principles, and practices. We've shown you how to put them all together to help your team deliver more value to your users and customers. We've

helped you find things that you can do today to help your team, and given you tools (like playing "Are you OK with?") to help your team with their mindset. We've given you many tools for learning agile, and understanding Scrum, XP, Lean, and Kanban.

This is *not* how most people adopt agile.

As much as we'd love it if every developer in the world buys and reads our book, the truth is that most people on most teams don't learn agile by reading. They learn it by doing. And it's easier to do something if it's broken into small, straightforward chunks.

In other words, when most people first try to learn agile, what they really want is a simple list of rules that they can follow that will get them to build better software more quickly.

If you're new to agile coaching, this can be frustrating. Imagine that you're trying to teach a team how to effectively adopt Scrum. You know from Chapter 5 that if someone doesn't understand self-organization and collective commitment, they don't "get" Scrum. So you spend time explaining the Scrum values of openness, courage, focus, commitment, and respect, and you talk about how self-organization works. But the team gets frustrated—these seem like very abstract concepts. They just want to talk about task boards and stories. You want them to "get" Scrum; they seem determined to get better-than-not-doing-it results, and you don't seem to be able to do anything about it. What's going on here?

There's an old saying about teaching: *meet them where they are, not where you want them to be*. A good agile coach understands how people learn, and presents them with information, guidance, and examples to help them reach their next stage in learning. Coaches understand that people don't change overnight.

In *Agile Software Development: The Cooperative Game*, Alistair Cockburn talks about a basic idea of learning called **shuhari**, and it's a very valuable tool for a coach trying to figure out where the team is. Shuhari is adopted from martial arts, but it's a good way to think about how people learn about almost anything.

In Shuhari, learning comes in three stages. The first stage, *shu* (守, "obey" or "observe"), describes the mindset of someone who first encounters a new idea or methodology. He wants simple rules that he can follow. This is one reason that people latch onto practices: the team can make a rule about adopting sprints, pair programming, using a task board, etc., and everyone can tell whether or not it's being followed.

Rules are especially important for adults learning new things at work, because unlike kids in school, adults aren't in the habit of learning new things all the time. When you give someone who is learning a new system—like an agile methodology—simple rules to follow, it gives her a place to start. The straightforward practices that are part

of agile methodologies like Scrum, XP, and Kanban work very well for someone in the *shu* stage, because they are simple enough that people can focus on getting good at them. By setting a rule for the team today (like "we have a Daily Scrum meeting at 10:30 every morning where each person answers three questions"), you can help get them on the right track to learn a principle ("we use self-organization to constantly review and adjust our plan").

Your goal as coach, however, is *not to establish an agile orthodoxy*. An agile zealot is someone who has decided that there is exactly one way of doing agile that will work for every team, and who loudly and forcefully pushes that way of doing things on everyone. Agile zealots tend to focus only on the rules that they've learned, and stop at the *shu* level. They believe every problem that they encounter can be solved with a rule that they already know and preach. Agile zealots do not make good coaches because they don't take the time to understand where the people that they're coaching are in their learning.

An agile coach needs to understand more than simple rules, which brings us to the next stage in Shuhari, *ha* (破, "detach" or "break"). This is the stage where people have gotten practice with the rules, and can start to understand and internalize the principles that drive them. Throughout this book you've learned how an effective way to change the mindset of your team is to adopt the practices of a methodology, and through its practices (and lots of team discussion) come to understand its values and principles. When people on a team reach the *ha* stage together, they start to progress from getting better-than-not-doing-it results to getting the real gains in productivity that come when a team truly adopts an agile mindset.

We haven't talked much about the last stage, *ri* (離, "separate"). When you reach the *ri* stage of learning, you are fluent in the ideas, values, and principles. You care less about the methodologies, because you've gone beyond them. When you have a problem that can be fixed with a specific practice, you and your team just do that practice. You don't really care if it's Scrum, XP, Kanban, waterfall—it doesn't matter, and your process doesn't need a name. But it's not like the chaos that you see on a team where nobody has tried agile. A team where everyone has reached *ri* is a smooth, well-functioning team, where everyone just does what's right.

Cockburn points out just how "distressingly Zen" this can sound. In *Agile Software Development: The Cooperative Game*, he talks about how people at the *ri* stage of learning say things that are difficult for a *shu*-level learner to understand:

> "Do whatever works."
>
> "When you are really doing it, you are unaware that you are doing it."
>
> "Use a technique so long as it is doing some good."
>
> To someone at the fluent level of behavior, this is all true. To someone still detaching, it is confusing. To someone looking for a procedure to follow, it is useless.

The first job of a good agile coach is to talk to everyone on the team and figure out each person's current stage of learning. To the coach, a team with people at the *ri* stage of learning is a team that's very easy to work with, because every team member knows how to learn, and is willing to adapt to new practices.

On the other hand, a person at the *shu* stage wants unambiguous rules. For example, do you use stickies or index cards on your task board? You may know as a coach that it doesn't really matter. But someone who has never used a task board has no way of knowing whether this is a trivial or critical decision, so your job is to give him a rule. Later you can help him understand why you chose, say, stickies, and that index cards will do just as good a job. They're a little different—for example, you can write on both sides of an index card, but only one side of a sticky—and you can use the similarities and differences to help the team members learn how they both fulfill the same principles.

Use Shuhari to Help a Team Learn the Values of a Methodology

Put yourself back into the shoes of the frustrated coach trying to teach the team to "get" Scrum. Shuhari can help you understand why the team is frustrated when learning about values, and why they only want to talk about the most tangible practices like task boards and stories. A task board or story is not hard to present as a *shu*-level concept: a straightforward rule that you can apply to a project today. Scrum is full of *shu*-level rules ("meet every day and answer these three questions"); this is one reason that teams find a lot of success in adopting its practices.

Values like openness and commitment, on the other hand, are *ha*-level ides: abstract concepts that govern how you employ rules in a system. Self-organization and collective commitment happen only when everyone on the team has really understood and internalized those values. Shuhari helps us understand why the team needs to first go through the motions of the practices before they can "detach" and start to understand what openness and commitment really mean.

This is why adopting the practices of Scrum is a very effective first step in helping the team to learn its values. A good coach is patient, and waits for opportunities during each sprint to teach the team about the values. Often, these opportunities happen when two people with different perspectives see different aspects of the Scrum "elephant."

For example, let's say that a developer discovers halfway through a sprint that a feature won't be finished, and at the next Daily Scrum he asks the Product Owner to push it to the next sprint. The Product Owner gets angry. She's afraid that a customer or manager will get mad about this, and wants the developer to "do whatever it takes" to get it in—even if it means cutting corners and not finishing all of the code. There's a *shu*-level answer to this problem: the rules of Scrum dictate that the team can only demo the feature at the sprint review if it's "*done* done," and incomplete work *must* be

pushed into the next sprint. An agile coach understands the rules of Scrum, and can use them to resolve the conflict.

But a good agile coach also sees that there's a *ha*-level teaching opportunity here. By recognizing a fractured perspective (like we learned about in Chapter 2) that can be "un-fractured," the coach can help the Product Owner see things from the developer's perspective—that doing poor work will add technical debt, and this will make the entire codebase more difficult to maintain. That technical debt, in turn, will cause future sprints to take longer. So while the Product Owner might end up getting the feature to a customer sooner by ignoring the Scrum rule, in the end she will be worse off because future development will go more slowly. This an important lesson that will help the Product Owner learn more about the Scrum value of *focus*, and how a team that focuses on completing work delivers a higher quality product more quickly.

The coach can also help the developer understand how important the feature is, and how it delivers value to the customers—and how in the future, a better understanding of value can help the developer work more closely with the Product Owner to find a way to break features into smaller ones and deliver more value to the customers sooner. This is an important lesson that will help the developer learn more about the Scrum value of *commitment*, and that a team can be more effective when they really understand what's valuable to the customers and users.

This is an example of how the coach with a deep understanding of a methodology fills an important and valuable role on the team. The coach helps the team understand and adopt the practices and rules of the methodology. And through those rules, the coach can help everyone on the team understand the values of agile, and use those values to help every single person on the team grow into a more agile mindset.

Coaches Understand What Makes a Methodology Work

Conflicts and problems happen on projects every day. A good Scrum Master, for example, spends most of his time resolving issues and problems. Many of these problems won't necessarily present great opportunities to teach the team about Scrum.

So in the example that we just showed you, how did the coach know that there was an opportunity in this particular conflict to teach the team about the Scrum values of *focus* and *commitment*? What was it about this particular problem that the coach recognized?

The coach recognized the opportunity because he understands not just *how* Scrum teams use iteration, but *why* they use it. He knows that a timeboxed iteration can be rendered completely ineffective if the team is allowed to include work that isn't "*done* done." Without that rule, iterations turn into simple project milestones: the scope of each iteration becomes more and more variable, and the team no longer has to deliver working software at the end of each iteration.

This breaks several important values and principles that make agile work. For example, the coach knows that working software is the primary measure of progress, and delivering incomplete software at the end of a sprint gives the customers and users a false measure of that progress. The coach also knows that agile teams value customer collaboration—but he also understands that customer collaboration doesn't mean that the customer is always right. It means that if the team recognizes that there is a real problem with the plan, they can collaborate with the customer to come up with a solution that delivers the most value. He recognizes that if the customer and Product Owner can pressure developers to include software that isn't complete in a sprint review, the team will become more guarded about the true progress of the work, and eventually customer collaboration will give way to contract negotiation between the team and the customer.

The coach knows all of this because he understands the values and principles of agile and Scrum, and knows how those values and principles drive the practices. He recognizes what makes the methodologies "tick." A Scrum coach understands collective commitment and self-organization. An XP coach understands embracing change and incremental design. A Kanban coach understands how improving the team's process requires setting WIP limits and using them to control the flow of work—and that when someone tries to prevent those WIP limits from getting set, the entire methodology unravels.

In *Coaching Agile Teams*, Lyssa Adkins has a suggestion for how coaches can approach problems with the team—we talked about this earlier in the book, but it's worth repeating:

> **Let the team fail:** Certainly, don't stand idly by while the team goes careening off a cliff. But do use the dozens of opportunities that arise each sprint to let them fail. Teams that fail together and recover together are much stronger and faster than ones that are protected. And the team may surprise you. The thing you thought was going to harm them may actually work for them. Watch and wait.

This is excellent advice for any agile coach, especially one who is prone to what Adkins refers to as "command-and-control-ism," or a need to control every aspect of the team's learning and progress. A good coach understands that there are times when the team can and should fail, because that's the most effective and efficient path to a successful project and team.

This is where retrospectives are extremely valuable. We learned about how Scrum and XP teams use retrospectives to look back at the project and find ways to improve. To an agile coach, this is a great opportunity to help the team learn from the failure—and learning is the whole goal of letting them fail. If the team failed because they didn't implement a practice well enough, the coach can help them learn the skills they need to get better at it. But sometimes failure happens because the teams need to improve their mindset—and because an agile coach knows why methodologies and practices work, he or she can use the failure to help the team learn more about the

specific value or principle that would have helped them to make better decisions and avoid failure. This is how teams grow.

But a good coach also understands when a team can't be allowed to fail. Just like there are supporting beams and walls in a house that can't be moved during a renovation, every methodology has its "supporting beams" that cannot be changed without compromising the integrity of the project. A coach needs a *ha*-level understanding of the methodology: he has a deep knowledge of it as a system, and knows why it works. In Chapter 8, we learned about the boss who inadvertently cuts the legs out from under an entire Kanban effort by asking the team to remove the WIP limit. Someone with a *shu*-level understanding might let that WIP limit get changed, feeling like it's better to get at least some of Kanban in place. A good Kanban coach understands that while some things (like the specific columns on a kanban board) can be changed, the WIP limit must be left intact, because it's the lynchpin that holds the entire Kanban methodology together.

One of the more difficult parts of an agile coach's job is to figure out just how much of this to explain to the team, the boss, and the customers. A simple *shu*-level explanation of the rules of a methodology is enough to get the job done, but not enough to help the team get into the right mindset to get better-than-not-doing-it results. Sometimes the team is satisfied with a simple set of rules to follow; other times, they may feel like they've just traded an arbitrary set of waterfall rules for an arbitrary set of agile rules. A *ha*-level understanding will help them see why following these rules will produce better software; but it also may sound "distressingly Zen," as Cockburn put it. The agile coach is there to help the team move toward their *ha*-level of understanding at a pace that they can handle, and without overly abstract explanations that don't help them.

The Principles of Coaching

If there's one thing that we've reinforced throughout this book, it's that agile requires the right mindset, and that teams get to that mindset through values and principles. That's why every agile methodology comes with a set of values, and team members only approach the full potential of an agile methodology when they understand and internalize those values.

So it shouldn't come as a surprise that coaching has its own set of values. John Wooden, coach of the UCLA men's basketball team in the 1960s and 1970s, is widely considered one of the greatest coaches in the history of sports. In his book, *Practical Modern Basketball*, he lays out five basic principles of coaching: **industriousness**, **enthusiasm**, **condition**, **fundamentals**, and **development of team spirit**. They make just as much sense for an agile coach as they do in basketball:

Industriousness

Changing the way a team builds software means working hard at things that they've never done before. Developers have to think about planning and testing, not just coding. Product owners need to understand what the team does, not just throw features over the wall at them. Scrum masters need to learn how to give the team control, while still staying involved with the project. These are all new skills, and all new skills require work.

Enthusiasm

When your heart is in your work, and you're excited about a new way of doing things, it rubs off on everyone around you. And there's a lot to be enthusiastic about: you're solving problems that have given you headaches in the past. When everyone goes into agile with a real enthusiasm toward it, you get more than just better software; you get a team where everyone is more innovative, happier, and excited about working together every day.

Condition

Agile works when every team member is good at what he or she does. There's a real, implicit assumption that everyone has pride of workmanship: they try to build the best software that they can, and work to get better at building it—and that they're genuinely motivated by pride in their work. This is why every member of an agile team continues to work hard at improving their skills, so that they bring their best condition to the project.

Fundamentals

Wooden writes, "the finest system cannot overcome poor execution of the fundamentals. The coach must be certain that he never permits himself to get 'carried away' by a complicated system"—and this is especially relevant for an agile coach. Agile works because its values are simple and straightforward, and its methodologies consist of uncomplicated practices. These are the fundamentals of agile, and a good coach works to keep the team focused on them, and helps the team to keep things simple.

Development of team spirit

We've talked about self-organization, whole team, energized work, and empowering the team: ways that agile teams build each other up and create a collaborative, innovative work environment. On the flipside, an agile coach needs to be on the lookout for team members who are primarily focused on their own personal performance, career goals, résumé, or getting promoted out of the team, and help them change their attitude. Wooden is clear about the effect that sort of person has on team spirit, and what to do about it: "The coach must use every bit of psychology at his command and use every available method to develop a fine team spirit on his squad. Teamwork and unselfishness must be encouraged at every opportunity, and each [team member] must be eager, not just willing, to sacrifice

personal glory for the welfare of the team. Selfishness, envy, egotism, and criticism of each other can crush team spirit and ruin the potential of any team. The coach must be aware of this and constantly alert to prevent such traits by catching them at the source before trouble develops."

These five principles are a solid foundation for the mindset of an agile coach. An important step in becoming a great agile coach is to understand, internalize, and use them, in exactly the same way as you would use the values of the Agile Manifesto and any agile methodology.

Key Points

- An **agile coach** is someone who helps a team adopt agile, and helps each individual person on the team learn a new mindset.
- Effective agile coaches help teams get past their discomfort with new practices by focusing on the part of the practice that feels familiar.
- Coaches recognize the warning signs that teams are uncomfortable with change.
- People learn new systems or mindsets in three stages called **shuhari**.
- Someone who encounters an idea for the first time is in the first stage, **shu**, where she's still learning the rules—she often responds best to specific instructions.
- The **ha** stage is where someone starts to recognize the larger system, and has adapted his mindset to match it.
- In the **ri** stage, people are fluent with the ideas of the system—for example, they know when rules don't have to be followed.
- An effective agile coach knows why a system works, and understands why the rules are there.