Choose Your WoW!

A Disciplined Agile Delivery Handbook for Optimizing Your Way of Working

Scott W. Ambler

Mark Lines

Version: 1.1

CONTENTS

FOREWORD

PREFACE

SECTION 1: DISCIPLINED AGILE DELIVERY IN A NUTSHELL

1 CHOOSING YOUR WOW!

2 BEING DISCIPLINED

3 DISCIPLINED AGILE DELIVERY (DAD) IN A NUTSHELL

4 ROLES, RIGHTS, AND RESPONSIBILITIES

5 PROCESS GOALS

6 CHOOSING THE RIGHT LIFE CYCLE

SECTION 2: SUCCESSFULLY INITIATING YOUR TEAM

7 FORM TEAM

8 ALIGN WITH ENTERPRISE DIRECTION

9 EXPLORE SCOPE

10 IDENTIFY ARCHITECTURE STRATEGY

11 PLAN THE RELEASE

12 DEVELOP TEST STRATEGY

13 DEVELOP COMMON VISION

14 SECURE FUNDING

SECTION 3: PRODUCING BUSINESS VALUE

15 PROVE ARCHITECTURE EARLY

16 ADDRESS CHANGING STAKEHOLDER NEEDS

17 Produce a Potentially Consumable Solution

18 Improve Quality

19 Accelerate Value Delivery

SECTION 4: RELEASING INTO PRODUCTION

20 Ensure Production Readiness

21 Deploy the Solution

SECTION 5: SUSTAINING AND ENHANCING YOUR TEAM

22 Grow Team Members

23 Coordinate Activities

24 Evolve Way of Working (WoW)

25 Address Risk

26 Leverage and Enhance Existing Infrastructure

27 Govern Delivery Team

SECTION 6: PARTING THOUGHTS AND BACK MATTER

28 Disciplined Success

Appendix – Disciplined Agile Certification

References and Additional Resources

Acronyms and Abbreviations

About the Authors

SECTION 1: DISCIPLINED AGILE DELIVERY (DAD) IN A NUTSHELL

This section is organized into the following chapters:

- **Chapter 1: Choosing Your WoW!** Overview of how to apply this book.
- **Chapter 2: Being Disciplined.** Values, principles, and philosophies for disciplined agilists.
- **Chapter 3: Disciplined Agile Delivery in a Nutshell.** An overview of DAD.
- **Chapter 4: Roles, Rights, and Responsibilities.** Individuals and interactions.
- **Chapter 5: The Process Goals.** How to focus on process outcomes rather than conform to process prescriptions.
- **Chapter 6: Choosing the Right Life Cycle.** How teams can work in unique ways, yet still be governed consistently.

1 CHOOSING YOUR WOW!

A man's pride can be his downfall, and he needs to learn when to turn to others for support and guidance.—Bear Grylls

Key Points in This Chapter

- Disciplined Agile Delivery (DAD) teams have the autonomy to choose their way of working (WoW).
- You need to both "be agile" and know how to "do agile."
- Software development is complicated; there's no easy answer for how to do it.
- Disciplined Agile (DA) provides the scaffolding—a tool kit of agnostic advice—to choose your WoW.
- Other people have faced, and overcome, similar challenges to yours. DA enables you to leverage their learnings.
- You can use this book to guide how to initially choose your WoW and then evolve it over time.
- The real goal is to effectively achieve desired organizational outcomes, not to be/do agile.
- Better decisions lead to better outcomes.

Welcome to *Choose Your WoW!*, the book about how agile software development teams, or more accurately agile/lean solution delivery teams, can choose their way of working (WoW). This chapter describes some fundamental concepts around why choosing your WoW is important, fundamental strategies for how to do so, and how this book can help you to become effective at it.

Why Should Teams Choose Their WoW?

Agile teams are commonly told to own their process, to choose their WoW. This is very good advice for several reasons:

- **Context counts.** People and teams will work differently depending on the context of their situation. Every person is unique, every team is unique, and every team finds itself in a unique situation. A team of five people will work differently than a team of 20, than a team of 50. A team in a life-critical regulatory situation will work

differently than a team in a nonregulatory situation. Our team will work differently than your team because we're different people with our own unique skill sets, preferences, and backgrounds.

- **Choice is good.** To be effective, a team must be able to choose the practices and strategies to address the situation that they face. The implication is that they need to know what these choices are, what the trade-offs are of each, and when (not) to apply each one. In other words, they either need to have a deep background in software process, something that few people have, or have a good guide to help them make these process-related choices. Luckily, this book is a very good guide.
- **We should optimize flow.** We want to be effective in the way that we work, and ideally to delight our customers/stakeholders in doing so. To do this, we need to optimize the workflow within our team and in how we collaborate with other teams across the organization.
- **We want to be awesome.** Who wouldn't want to be awesome at what they do? Who wouldn't want to work on an awesome team or for an awesome organization? A significant part of being awesome is to enable teams to choose their WoW and to allow them to constantly experiment to identify even better ways they can work.

In short, we believe that it's time to take back agile. Martin Fowler recently coined the term "agile industrial complex" to refer to the observation that many teams are following a "faux agile" strategy, sometimes called "agile in name only" (AINO). This is often the result of organizations adopting a prescriptive framework, such as SAFe, and then forcing teams to adopt it regardless of whether it actually makes sense to do so (and it rarely does). Or forcing teams to follow an organizational standard application of Scrum. Yet canonical agile is very clear; it's individuals and interactions over processes and tools—teams should be allowed, and better yet, supported, to choose and then evolve their WoW.

You Need to "Be Agile" *and* Know How to "Do Agile"

Scott's daughter, Olivia, is 10 years old. She and her friends are some of the most agile people we've ever met. They're respectful (as much as 10-year-old children can be), they're open-minded, they're collaborative, they're eager to learn, and they're always experimenting. They clearly embrace an agile mindset, yet if we were to ask them to develop software it would be a disaster. Why? Because they don't have the skills. They could gain these skills in time, but right now they just don't know what they're doing

when it comes to software development. We've also seen teams made up of millennials who collaborate very naturally and have the skills to develop solutions, although perhaps are not yet sufficiently experienced to understand the enterprise-class implications of their work. And, of course, we've seen teams of developers with decades of IT experience but very little experience doing so collaboratively. None of these situations are ideal. Our point is that it's absolutely critical to have an agile mindset, to "be agile," but you also need to have the requisite skills to "do agile" and the experience to "do enterprise agile." An important aspect of this book is that it comprehensively addresses the potential skills required by agile/lean teams to succeed.

The real goal is to effectively achieve desired organizational outcomes, not to be/do agile. What good is it to be working in an agile manner if you're producing the wrong thing, or producing something you already have, or are producing something that doesn't fit into the overall direction of your organization? Our real focus must be on achieving the outcomes that will make our organization successful, and becoming more effective in our WoW will help us to do that.

Accept That There's No Easy Answer

Software development, or more accurately solution delivery, is complex. You need to be able to initiate a team, produce a solution that meets the needs of your stakeholders, and then successfully release it to them. You need to know how to explore their needs, architect and design a solution, develop that solution, validate it, and deploy it. This must be done within the context of your organization, using a collection of technologies that are evolving, and for a wide variety of business needs. And you're doing this with teams of people with different backgrounds, different preferences, different experiences, different career goals, and they may report to a different group or even a different organization than you do.

We believe in embracing this complexity because it's the only way to be effective, and better yet, to be awesome. When we ignore important aspects of our WoW, say architecture for example, we tend to make painful mistakes in that area. When we denigrate aspects of our WoW, such as governance, perhaps because we've had bad experiences in the past with not-so-agile governance, then we risk people outside of our team taking responsibility for that aspect and inflicting their non-agile practices upon us. In this way, rather than enabling our agility, they act as impediments.

We Can Benefit From the Learnings of Others

A common mistake that teams make is that they believe that just because they face a unique situation that they need to figure out their WoW from scratch. Nothing could be further from the truth. When you develop a new application, do you develop a new language, a new compiler, new code libraries, and so on, from scratch? Of course not, you adopt the existing things that are out there, combine them in a unique way, and then modify them as needed. Development teams, regardless of technology, utilize proven frameworks and libraries to improve productivity and quality. It should be the same thing with process. As you can see in this book, there are hundreds, if not thousands, of practices and strategies out there that have been proven in practice by thousands of teams before you. You don't need to start from scratch, but instead can develop your WoW by combining existing practices and strategies and then modifying them appropriately to address the situation at hand. DA provides the tool kit to guide you through this in a streamlined and accessible manner. Since our first book on DAD [AmblerLines2012], we have received feedback that while it is seen as an extremely rich collection of strategies and practices, practitioners sometimes struggle to understand how to reference the strategies and apply them. One of the goals of this book is to make DAD more accessible so that you can easily find what you need to customize your WoW.

One thing that you'll notice throughout the book is that we provide a lot of references. We do this for three reasons: First, to give credit where credit is due. Second, to let you know where you can go for further details. Third, to enable us to focus on summarizing the various techniques and to put them into context, rather than going into the details of every single one. The goal is to make you aware of what techniques are available, and the trade-offs of each based on context. You can then find other detailed information on how to apply a technique elsewhere. For example, we will identify and compare test-driven development (TDD) to test-after development as potential techniques to experiment with, and then you can do further research into your chosen option. Here is our approach to references:

- **[W].** This indicates that there is a Wikipedia page for the concept at wikipedia.org. Wikipedia is an online, open-content, collaborative encyclopedia that allows anyone to alter its content. With the absence of peer review and validation of content, PMI cannot ensure that the information

available is complete, accurate, reliable, or corresponds with the current state of knowledge in the relevant fields. Having said that, many of these pages could use some work. Wikipedia pages cited in this book can be located in the Additional Resources section at the end of this book. Our hope is that readers such as yourself will step up and help to evolve these pages so as to share our expertise with the rest of the world.

- **[MeaningfulName].** There is a corresponding entry in the references at the back of the book. This is an indication that either we couldn't find an appropriate Wikipedia page or that we had a detailed source on the subject already. Either way, we'd really like to see Wikipedia pages developed for these topics, so please consider starting one if you're knowledgeable about that topic. Also feel free to reach out to us as we'd be happy to donate appropriate material to help seed the effort.
- **[W, MeaningfulNames].** This indicates Wikipedia has a good page, plus there are a few more resources that we recommend. Please consider updating the Wikipedia page though.
- **No reference.** When a technique is a practice, such as TDD, we can often find a solid reference for it. When the technique is a strategy, such as testless programming, then it's difficult to find a reference for it. So please consider writing a blog about that strategy that we could refer to in the future.

DA Knowledge Makes You a Far More Valuable Team Member

We have heard from many DA organizations—and they permit us to quote them—that team members who have invested in learning DA (and proving it through challenging certifications) become more valuable contributors. The reason to us is quite clear. Understanding a larger library of proven strategies means that teams will make better decisions and “fail fast” less, and rather “learn and succeed earlier.” A lack of collective self-awareness of the available options is a common source of teams struggling to meet their agility expectations—and that is exactly what happens when you adopt prescriptive methods/frameworks that don't provide you with choices. Every team member, especially consultants, are expected to bring a tool kit of ideas to customize the team's process as part of self-organization. A larger tool kit and commonly understood terminology is a good thing.

The Disciplined Agile (DA) Tool Kit Provides Accessible Guidance

One thing that we have learned over time is that some people, while they understand the concepts of DA by either reading the books or attending a workshop, struggle with how to actually apply DA. DA is an extremely rich body of knowledge that is presented in an accessible manner.

The good news is that the content of this book is organized by the goals, and that by using the goal-driven approach, it is easy to find the guidance that you need for the situation at hand. Here's how you can apply this tool kit in your daily work to be more effective in achieving your desired outcomes:

- Contextualized process reference
- Guided continuous improvement (GCI)
- Process-tailoring workshops
- Enhanced retrospectives
- Enhanced coaching

Contextualized Process Reference

As we described earlier, this book is meant to be a reference. You will find it handy to keep this book nearby to quickly reference available strategies when you face particular challenges. This book presents you with process choices and more importantly puts those choices into context. DA provides three levels of scaffolding to do this:

1. **Life cycles.** At the highest level of WoW guidance are life cycles, the closest that DAD gets to methodology. DAD supports six different life cycles, as you can see in Figure 1.1, to provide teams with the flexibility of choosing an approach that makes the most sense for them. Chapter 6 explores the life cycles, and how to choose between them, in greater detail. It also describes how teams can still be governed consistently even though they're working in different ways.
2. **Process goals.** Figure 1.2 presents the goal diagram for the Improve Quality process goal, which is described in detail in Chapter 18, and Figure 1.3 overviews the notation of goal diagrams. DAD is described as a collection of 21 process goals, or process outcomes, if you like. Each goal is described as a collection of decision points, issues that your team needs to determine whether

they need to address, and if so, how they will do so. Potential practices/strategies for addressing a decision point, which can be combined in many cases, are presented as lists. Goal diagrams are similar conceptually to mind maps, albeit with the extension of the arrow to represent relative effectiveness of options in some cases. Goal diagrams are, in effect, straightforward guides to help a team to choose the best strategies that they are capable of doing right now given their skills, culture, and situation. Chapter 5 explores the goal-driven approach in greater detail.

3. **Practices/strategies.** At the most granular level of WoW guidance are practices and strategies, depicted on goal diagrams in the lists on the right-hand side. Sections 2–4 of this book explore each process goal in detail, one per chapter. Each of these chapters overviews the process goal and key concepts behind the goal, describes each decision point for the goal, and then overviews each practice/strategy and the trade-offs associated with them in an agnostic manner.

An important implication of goal diagrams, such as the one in Figure 1.2, is that you need less process expertise to identify potential practices/strategies to try out. What you do need is an understanding of the fundamentals of DAD, the focus of Section 1 of this book, and familiarity with the goal diagrams so that you can quickly locate potential options. You do not need to memorize all of your available options because you can look them up, and you don't need to have deep knowledge of each option because they're overviewed and put into context in the individual goal chapters. Rather, you can use this book to refer to DA when you need guidance to solve particular challenges that you face.

Improvement Occurs at Many Levels

Process improvement, or WoW evolution, occurs across your organization. Organizations are a collection of interacting teams and groups, each of which evolves continuously. As teams evolve their WoWs, they motivate changes in the teams they interact with. Because of this constant process evolution, hopefully for the better, and because people are unique, it becomes unpredictable how people are going to work

together or what the results of that work will be. In short, your organization is a complex adaptive system (CAS) [W]. This concept is overviewed in Figure 1.4, which depicts teams, organization areas (such as divisions, lines of business, or value streams), and enterprise teams. Figure 1.4 is a simplification—there are far more interactions between teams and across organizational boundaries, and in large enterprises, an organizational area may have its own "enterprise" groups, such as enterprise architecture or finance—the diagram is complicated enough as it is. There are several interesting implications for choosing your WoW:

1. **Every team will have a different WoW.** We really can't say this enough.
2. **We will evolve our WoW to reflect learnings whenever we work with other teams.** Not only do we accomplish whatever outcome we set to achieve by working with another team, we very often learn new techniques from them or new ways of collaborating with them (that they may have picked up from working with other teams).
3. **We can purposefully choose to learn from other teams.** There are many strategies that we can choose to adopt within our organization to share learnings across teams, including practitioner presentations, communities of practice (CoPs)/guilds, coaching, and many others. Team-level strategies are captured in the Evolve WoW process goal (Chapter 24) and organizational-level strategies in the Continuous Improvement process blade[1] [AmblerLines2017]. In short, the DA tool kit is a generative resource that you can apply in agnostically choosing your WoW.
4. **We can benefit from organizational transformation/improvement efforts.** Improvement can, and should, happen at the team level. It can also happen at the organizational-area level (e.g., we can work to optimize flow between the teams within an area). Improvement also needs to occur outside of DAD teams (e.g., we can help the enterprise architecture, finance, and people management groups to collaborate with the rest of the organization more effectively).

As Figure 1.5 depicts, the Disciplined Agile (DA) tool kit is organized into four levels:

1. **Foundation.** The foundation layer provides the conceptual underpinnings of the DA tool kit.
2. **Disciplined DevOps.** DevOps is the streamlining of solution development and operations, and Disciplined DevOps is an enterprise-class approach to DevOps.

This layer includes Disciplined Agile Delivery (DAD), the focus of this book, plus other enterprise aspects of DevOps.

3. **Value Stream.** The value stream layer is based on Al Shalloway's FLEX. It's not enough to be innovative in ideas if these ideas can't be realized in the marketplace or in the company. FLEX is the glue that ties an organization's strategies in that it visualizes what an effective value stream looks like, enabling you to make decisions for improving each part of the organization within the context of the whole.
4. **Disciplined Agile Enterprise (DAE).** The DAE layer focuses on the rest of the enterprise activities that support your organization's value streams.

Teams, regardless of what level they operate at, can and should choose their WoW. Our focus in this book is on DAD teams, although at times we will delve into cross-team and organizational issues where appropriate.

Guided Continuous Improvement (GCI)

Many teams start their agile journey by adopting agile methods such as Scrum [W], Extreme Programming (XP) [W], or Dynamic Systems Development Method (DSDM)-Atern [W]. Large teams dealing with "scale" (we'll discuss what scaling really means in Chapter 2) may choose to adopt SAFe® [W], LeSS [W], or Nexus® [Nexus] to name a few. These methods/frameworks each address a specific class of problem(s) that agile teams face, and from our point of view, they're rather prescriptive in that they don't provide you with many choices. Sometimes, particularly when frameworks are applied to contexts where they aren't an ideal fit, teams often find that they need to invest significant time "descaling" them to remove techniques that don't apply to their situation, then add back in other techniques that do. Having said that, when frameworks are applied in the appropriate context, they can work quite well in practice. When you successfully adopt one of these prescriptive methods/frameworks, your team productivity tends to follow the curve shown in Figure 1.6. At first, there is a drop in productivity because the team is learning a new way of working, it's investing time in training, and people are often learning new techniques. In time, productivity rises, going above what it originally was, but eventually plateaus as the team falls into its new WoW. Things have gotten better, but without concerted effort to improve, you discover that team productivity plateaus.

Some of the feedback that we get about Figure 1.6 is that this can't be, that Scrum promises that you can do twice the work in half the time [Sutherland]. Sadly, this claim of four times productivity improvement doesn't seem to hold water in practice. A recent study covering 155 organizations, 1,500 waterfall, and 1,500 agile teams found actual productivity increases of agile teams, mostly following Scrum, to be closer to 7–12 % [Reifer]. At scale, where the majority of organizations have adopted SAFe, the improvement goes down to 3–5 %.

There are many ways that a team can adopt to help them improve their WoW, strategies that are captured by the Evolve WoW process goal described in Chapter 24. Many people recommend an experimental approach to improvement, and we've found guided experiments to be even more effective. The agile community provides a lots of advice around retrospectives, a working session where a team reflects on how they get better, and the lean community gives great advice for how to act on the reflections [Kerth]. Figure 1.7 summarizes W. Edward Deming's plan-do-study-act (PDSA) improvement loop [W], sometimes called a kaizen loop. This was Deming's first approach to continuous improvement, which he later evolved to plan do check act (PDCA), which became popular within the business community in the 1990s and the agile community in the early 2000s. But what many people don't realize is that after experimenting with PDCA for several years, Deming realized that it wasn't as effective as PDSA and went back to it. The primary difference being that the “study” activity motivated people to measure and think more deeply about whether a change worked well for them in practice. So we've decided to respect Deming's wishes and recommend PDSA rather than PDCA, as we found critical thinking such as this results in improvements that stick. Some people gravitate toward U.S. Air Force Colonel John Boyd's OODA (Observe Orient Decide Act) loop to guide their continuous improvement efforts—as always, our advice is to do what works for you [W]. Regardless of which improvement loop you adopt, remember that your team can, and perhaps should, run multiple experiments in parallel, particularly when the potential improvements are on different areas of your process and therefore won't affect each other (if they effect each other, it makes it difficult to determine the effectiveness of each experiment).

The basic idea with the PDSA/PDCA/OODA continuous improvement loop strategy is that you improve your WoW as a series of small changes, a strategy the lean community calls kaizen, which is Japanese for improvement. In Figure 1.9, you see the

workflow for running an experiment. The first step is to identify a potential improvement, such as a new practice or strategy, that you want to experiment with to see how well it works for you in the context of your situation. The effectiveness of a potential improvement is determined by measuring against clear outcomes, perhaps identified via a goal question metric (GQM) or an objectives and key results (OKRs) strategy as described in Chapter 27. Measuring the effectiveness of applying the new WoW is called validated learning [W]. It's important to note that Figure 1.8 provides a detailed description of a single pass through a team's continuous improvement loop.

The value of DA is that it can guide you through this identification step by helping you to agnostically identify a new practice/strategy that is likely to address the challenge you're hoping to address. By doing so, you increase your chance of identifying a potential improvement that works for you, thereby speeding up your efforts to improve your WoW—we call this guided continuous improvement (GCI). In short, at this level, the DA tool kit enables you to become a high-performing team quicker. In the original DAD book, we described a strategy called "measured improvement" that worked in a very similar manner.

A similar strategy that we've found very effective in practice is Lean Change[2] [LeanChange1, LeanChange2], particularly at the organizational level. The Lean Change management cycle, overviewed in Figure 1.9, applies ideas from Lean Startup [Ries] in that you have insights (hypothesis), identify potential options to address your insights, and then run experiments in the form of minimum viable changes (MVCs). These MVCs are introduced, allowed to run for a while, and then the results are measured to determine how effective they are in practice. Teams then can choose to stick with the changes that work well for them in the situation that they face, and abandon changes that don't work well. Where GGI enables teams to become high performing, Lean Change enables high-performing organizations.

The improvement curve for (unguided) continuous improvement strategies is shown in Figure 1.10 as a dashed line. You can see that there is still a bit of a productivity dip at first as teams learn how to identify MVCs and then run the experiments, but this is small and short lived. The full line depicts the curve for GCI in context; teams are more likely to identify options that will work for them, resulting in a higher rate of positive experiments and thereby a faster rate of improvement. In short, better decisions lead to better outcomes.

Of course, neither of the lines in Figure 1.10 are perfectly smooth. A team will have ups and downs, with some failed experiments (downs) where they learn what doesn't work in their situation and some successful experiences (ups) where they discover a technique that improves their effectiveness as a team. The full line, representing GCI, will be smoother than the dashed line because teams will have a higher percentage of ups.

The good news is that these two strategies, adopting a prescriptive method/framework and then improving your WoW through GCI, can be combined, as shown in Figure 1.11. We are constantly running into teams that have adopted a prescriptive agile method, very often Scrum or SAFe, that have plateaued because they've run into one or more issues not directly addressed by their chosen framework/method. Because the method doesn't address the problem(s) they face, and because they don't have expertise in that area, they tend to flounder. Ivar Jacobson has coined the term "they're stuck in method prison" [Prison]. By applying a continuous improvement strategy, or better yet, GCI, their process improvement efforts soon get back on track. Furthermore, because the underlying business situation that you face is constantly shifting, it tells you that you cannot sit on your "process laurels," but instead must adjust your WoW to reflect the evolving situation.

To be clear, GCI at the team level tends to be a simplified version of what you would do at the organizational level. At the team level, teams may choose to maintain an improvement backlog of things they hope to improve. At the organizational area or enterprise levels, we may have a group of people guiding a large transformation or improvement effort that is focused on enabling teams to choose their WoWs and to address larger, organizational issues that teams cannot easily address on their own.

Process-Tailoring Workshops

Another common strategy to apply DA to choose your WoW is a process-tailoring workshop [Tailoring]. In a process-tailoring workshop, a coach or team lead walks the team through important aspects of DAD and the team discusses how they're going to work together. This typically includes choosing a life cycle, walking through the process goals one at a time and addressing the decision points of each one, and discussing roles and responsibilities.

A process-tailoring workshop, or several short workshops, can be run at any time.

As shown in Figure 1.12, they are typically performed when a team is initially formed to determine how they will streamline their initiation efforts (what we call Inception, described in detail in Section 2), and just before Construction begins to agree on how that effort will be approached. Any process decisions made in process-tailoring workshops are not carved in stone but instead evolve over time as the team learns. You always want to be learning and improving your process as you go, and in fact, most agile teams will regularly reflect on how to do so via holding retrospectives. In short, the purpose of process-tailoring workshops is to get your team going in the right direction, whereas the purpose of retrospectives is to identify potential adjustments to that process.

Process-Tailoring Workshops in a Large Financial Institution By Daniel Gagnon

In my experience in running dozens of process-tailoring workshops over several years, with teams of every shape, size, and experience level and in different organizations [Gagnon], interestingly, the most recurring comment is that the workshops "revealed all kinds of options we didn't even realize were options!" Although almost always a bit of a hard sell at the outset, I have yet to work with a team that is unable to quickly grasp and appreciate the value of these activities.

Here are my lessons learned:

1. A team lead, architecture owner, or senior developer can actually stand in for most of the developers in the early stages.
2. Tools help. We developed a simple spreadsheet to capture WoW choices.
3. Teams can make immediate WoW decisions and identify future, more "mature" aspirational choices that they set as improvement goals.
4. We defined a small handful of enterprise-level choices to promote consistency across teams, including some "infrastructure as code" choices.
5. Teams don't have to start from a blank slate, but instead can start with the choices made by a similar team and then tailor it from there.

Here's an important note on determining participation: Ultimately, the teams themselves are the best arbiters of who should attend the sessions at varying stages of advancement. The support will become easier and easier to obtain as the benefits of allowing teams to choose their WoW become apparent.

Daniel Gagnon has coached the adoption of Disciplined Agile in two large Canadian financial institutions and is now a senior agile coach with Levio in Quebec.

A valid question to ask is what does the timeline look like for evolving the WoW within a team? Jonathan Smart, who guided the transformation at Barclays, recommends Dan North's visualize, stabilize, and optimize timeline as depicted in Figure 1.13. You start by visualizing your existing WoW and then identifying a new potential WoW that the team believes will work for them (this is what the initial tailoring is all about). Then the team needs to apply that new WoW and learn how to make it work in their context. This stabilization phase could take several weeks or months, and once the team has stabilized its WoW then it is in a position to evolve it via a GCI strategy.

The good news is that with effective facilitation, you can keep process-tailoring workshops streamlined. To do this, we suggest that you:

- Schedule several short sessions (you may not need all of them).
- Have a clear agenda (set expectations).
- Invite the entire team (it's their process).
- Have an experienced facilitator (this can get contentious).
- Arrange a flexible work space (this enables collaboration).

A process-tailoring workshop is likely to address several important aspects surrounding our way of working (WoW):

- Determine the rights and responsibilities of team members, which is discussed in detail in Chapter 4.
- How do we intend to organize/structure the team?
- What life cycle will the team follow? See Chapter 6 for more on this.
- What practices/strategies will we follow?
- Do we have a definition of ready (DoR) [Rubin], and if so what is it?
- Do we have a definition of done (DoD) [Rubin], and if so what is it?
- What tools will we use?

Process-tailoring workshops require an investment in time, but they're an effective way to ensure that team members are well aligned in how they intend to work together. Having said that, you want to keep these workshops as streamlined as possible as they can easily take on a life of their own—the aim is to get going in the right "process direction." You can always evolve your WoW later as you learn what

works and what doesn't work for you. Finally, you still need to involve some people who are experienced with agile delivery. DA provides a straightforward tool kit for choosing and evolving your WoW, but you still need the skills and knowledge to apply this tool kit effectively.

While DA provides a library or tool kit of great ideas, in your organization you may wish to apply some limits to the degree of self-organization your teams can apply. In DAD, we recommend self-organization within appropriate governance. As such, what we have seen with organizations that adopt DA is that they sometimes help steer the choices so that teams self-organize within commonly understood organizational "guard rails."

Enhance Retrospectives Through Guided Improvement Options

A retrospective is a technique that teams use to reflect on how effective they are and hopefully to identify potential process improvements to experiment with [W, Kerth]. As you would guess, DA can be used to help identify improvements that would have a good chance of working for you. As an example, perhaps you are having a discussion regarding excessive requirements churn due to ambiguous user stories and acceptance criteria. The observation may be that you need additional requirements models to clarify the requirements. But which models to choose? Referring to the Explore Scope process goal, described in Chapter 9, you could choose to create a domain diagram to clarify the relationships between entities, or perhaps a low-fidelity user interface (UI) prototype to clarify user experience (UX). We have observed that by using DA as a reference, teams are exposed to strategies and practices that they hadn't even heard of before.

Enhance Coaching by Extending the Coach's Process Tool Kit

DA is particularly valuable for agile coaches. First of all, an understanding of DA means that you have a larger tool kit of strategies that you can bring to bear to help solve your team's problems. Second, we often see coaches refer to DA to explain that some of the things that the teams or the organization itself sees as "best practices" are actually very poor choices, and that there are better alternatives to consider. Third, coaches use DA to help fill in the gaps in their own experience and knowledge.

Documenting Your WoW

Sigh, we wish we could say that you don't need to document your WoW. But the reality is that you very often do, and for one or more very good reasons:

1. **Regulatory.** Your team works in a regulatory environment where by law you need to capture your process—your WoW—somehow.
2. **It's too complicated to remember.** There are a lot of moving parts in your WoW. Consider the goal diagram of Figure 1.2. Your team will choose to adopt several of the strategies called out in it, and that's only one of 21 goals. As we said earlier, solution delivery is complex. We've done our best in DA to reduce this complexity so as to help you to choose your WoW, but we can't remove it completely.
3. **It provides comfort.** Many people are uncomfortable with the idea of not having a "defined process" to follow, particularly when they are new to that process. They like to have something to refer to from time to time to aid their learning. As they become more experienced in the team's WoW, they will refer to the documentation less until finally they never use it at all.

Because few people like to read process material, we suggest you keep it as straightforward as possible. Follow agile documentation [AgileDocumentation] practices, such as keeping it concise and working closely with the audience (in this case, the team itself) to ensure it meets their actual needs. Here are some options for capturing your WoW:

- Use a simple spreadsheet to capture goal diagram choices [Resources].
- Create an A3 (single sheet) overview of the process.
- Put up posters on the wall.
- Capture the process concisely in a wiki.

As we show in the Evolve WoW process goal (Chapter 24), there are several strategies that you can choose from to capture your WoW. A common approach is for a team to develop and commit to a working agreement. Working agreements will describe the roles and responsibilities that people will take on the team, the general rights and responsibilities of team members, and very often the team's process (their WoW). As shown in Figure 1.14, we like to distinguish between two important aspects of a team working agreement—the internal portion of it that describes how the team will work together and the external portion of it that describes how others should

interact with the team. The external portion of a team's working agreement in some ways is a service-level agreement (SLA), or application programming interface (API), for the team. It may include a schedule of common meetings that others may attend (for example, daily coordination meetings and upcoming demos), an indication of how to access the team's automated dashboard, how to contact the team, and what the purpose of the team is. The team's working agreement, both the internal and external aspects of it, will, of course, be affected by the organization environment and culture in which it operates.

In Summary

We've worked through several critical concepts in this chapter:

- Disciplined Agile Delivery (DAD) teams choose their way of working (WoW).
- You need to both "be agile" and know how to "do agile."
- Solution delivery is complicated; there's no easy answer for how to do it.
- Disciplined Agile (DA) provides the agnostic scaffolding to support a team in choosing their WoW to deliver software-based solutions.
- Other people have faced, and overcome, similar challenges to yours. DA enables you to leverage their learnings.
- You can use this book to guide how to initially choose your WoW and then evolve it over time.
- A guided continuous improvement (GCI) approach will help your teams to break out of "method prison" and thereby improve their effectiveness.
- The real goal is to effectively achieve desired organizational outcomes, not to be/do agile.
- Better decisions lead to better outcomes.

---

[1] A process blade addresses a cohesive process area—such as reuse engineering, finance, or procurement—in other layers of Disciplined Agile.

[2] In Chapter 7 of *An Executive's Guide to Disciplined Agile*, we show how to apply Lean Change at the organizational level.

2 BEING DISCIPLINED

Better decisions lead to better outcomes.

Key Points in This Chapter

- The Agile Manifesto is a great starting point, but it isn't sufficient.
- Lean principles are critical to success for agile solution delivery teams in the enterprise.
- The DA mindset is based on seven principles, seven promises, and eight guidelines.
- There are several "hashtag rebellions" that we can learn from.

What does it mean to be disciplined? To be disciplined is to do the things that we know are good for us, things that usually require hard work and perseverance. It requires discipline to regularly delight our customers. It takes discipline for teams to become awesome. It requires discipline for leaders to ensure that their people have a safe environment to work in. It takes discipline to recognize that we need to tailor our way of working (WoW) for the context that we face, and to evolve our WoW as the situation evolves. It takes discipline to recognize that we are part of a larger organization, that we should do what's best for the enterprise and not just what's convenient for us. It requires discipline to evolve and optimize our overall workflow, and it requires discipline to realize that we have many choices regarding how we work and organize ourselves, so we should choose accordingly.

The Manifesto for Agile Software Development

In 2001 the publication of the *Manifesto for Agile Software Development* [Manifesto], or Agile Manifesto for short, started the agile movement. The manifesto captures four values supported by 12 principles, which are listed below. It was created by a group of 17 people with deep experience in software development. Their goal was to describe what they had found to work in practice rather than describe what they hoped would work in theory. Although it sounds like an obvious thing to do now, back then this was arguably a radical departure from the approach taken by many thought leaders in the software engineering community.

The *Manifesto for Agile Software Development:*

> We are uncovering better ways of developing software by doing it and helping others do it. Through this work, we have come to value:
>
> 1. **Individuals and interactions** over processes and tools
> 2. **Working software** over comprehensive documentation
> 3. **Customer collaboration** over contract negotiation
> 4. **Responding to change** over following a plan
>
> That is, while there is value in the items on the right, we value the items on the left more.

There are 12 principles behind the Agile Manifesto that provide further guidance to practitioners. These principles are:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.

11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

The publication of the *Manifesto for Agile Software Development* has proven to be a milestone for the software development world and, as we've seen in recent years, for the business community as well. But time has had its toll, and the manifesto is showing its age in several ways:

1. **It is limited to software development.** The manifesto purposefully focused on software development, not other aspects of IT and certainly not other aspects of our overall enterprise. Many of the concepts can be modified to fit these environments, and they have over the years. Thus, the manifesto provides valuable insights that we can evolve and should be evolved and extended for a broader scope than was originally intended.
2. **The software development world has moved on.** The manifesto was crafted to reflect the environment in the 1990s, and some of the principles are out of date. For instance, the third principle suggests that we should deliver software from every few weeks to a couple of months. At the time, it was an accomplishment to have a demonstrable increment of a solution even every month. In modern times, however, the bar is significantly higher, with agile-proficient companies delivering functionality many times a day in part because the manifesto helped us to get on a better path.
3. **We've learned a lot since then.** Long before agile, organizations were adopting lean ways of thinking and working. Since 2001, agile and lean strategies have not only thrived on their own but they've been successfully commingled. As we will soon see, this commingling is an inherent aspect of the DA mindset. DevOps, the merging of software development and IT operations life cycles, has clearly evolved as a result of this commingling. There are few organizations that haven't adopted, or are at least in the process of adopting, DevOps ways of working—which Chapter 1 showed are an integral part of the DA tool kit. Our point is that it's about more than just agile.

Lean Software Development

The DA mindset is based on a combination of agile and lean thinking. An important starting point for understanding lean thinking is *The Lean Mindset* by Mary and Tom Poppendieck. In this book, they show how the seven principles of lean manufacturing can be applied to optimize the entire value stream. There is great value in this, but we must also remember that most of us are not manufacturing cars—or anything else for that matter. There are several types of work that lean applies to: manufacturing, services, physical-world product development, and (virtual) software development, among others. While we like the groundbreaking work of the Poppendieks, we prefer to look at the principles to see how they can apply anywhere [Poppendieck]. These principles are:

1. **Eliminate waste.** Lean-thinking advocates regard any activity that does not directly add value to the finished product as waste [WomackJones]. The three biggest sources of waste in our work are the addition of unrequired features, project churn, and crossing organizational boundaries (particularly between stakeholders and development teams). To reduce waste, it is critical that teams be allowed to self-organize and operate in a manner that reflects the work they're trying to accomplish. In product development work (the physical or virtual world), we spend considerable time discovering what is of value. Doing this is not waste. We've seen many folks have endless debates on what waste is because of this. We propose that a critical waste to eliminate is the waste of time due to delays in workflow. On reflection, it can be verified that most waste is reflected, even caused by, delays in workflow. We build unrequired features because we build too-large batches and have delays in feedback as to whether they are needed (or we're not writing our acceptance tests, which delays understanding what we need). Project churn (in particular, errors) is almost always due to getting out of sync without realizing we are. Crossing organizational boundaries is almost always an action that incurs delays as one part of the organization waits for the other.
2. **Build quality in.** Our process should not allow defects to occur in the first place, but when this isn't possible, we should work in such a way that we do a bit of work, validate it, fix any issues that we find, and then iterate. Inspecting after the fact and queuing up defects to be fixed at some time in the future isn't as effective. Agile practices that build quality into our process include test-driven development (TDD) and nonsolo development practices, such as pair

programming, mob programming, and modeling with others (mob modeling). All of these techniques are described later in this book.

3. **Create knowledge.** Planning is useful, but learning is essential. We want to promote strategies, such as working iteratively, that help teams discover what stakeholders really want and act on that knowledge. It's also important for team members to regularly reflect on what they're doing and then act to improve their approach through experimentation.
4. **Defer commitment.** It's not necessary to start solution development by defining a complete specification, and in fact that appears to be a questionable strategy at best. We can support the business effectively through flexible architectures that are change tolerant and by scheduling irreversible decisions for when we have more information and our decisions will be better—the last possible moment. Frequently, deferring commitment until the last responsible moment requires the ability to closely couple end-to-end business scenarios to capabilities developed in multiple applications by multiple teams. In fact, a strategy of deferring commitments to projects is a way of keeping our options open [Denning]. Software offers some additional mechanisms for deferring commitment. Through the use of emergent design, automated testing, and patterns thinking, essential decisions can often be deferred with virtually no cost. In many ways, agile software development is based on the concept that incremental delivery takes little extra implementation time while enabling developers to save mountains of effort that would otherwise be built on creating features that were not useful.
5. **Deliver quickly.** It is possible to deliver high-quality solutions quickly. By limiting the work of a team to within its capacity, we can establish a reliable and repeatable flow of work. An effective organization doesn't demand teams do more than they are capable of, but instead asks them to self-organize and determine what outcomes they can accomplish. Constraining teams to delivering potentially shippable solutions on a regular basis motivates them to stay focused on continuously adding value.
6. **Respect people.** The Poppendiecks also observe that sustainable advantage is gained from engaged, thinking people. The implication is that we need a lean approach to governance (see Govern Delivery Team in Chapter 27) that focuses

on motivating and enabling teams—not on controlling them.

7. **Optimize the whole.** If we want to be effective at a solution, we must look at the bigger picture. We need to understand the high-level business processes that a value stream supports—processes that often cross multiple systems and multiple teams. We need to manage programs of interrelated efforts, so we can deliver a complete product/service to our stakeholders. Measurements should address how well we're delivering business value, and the team should be focused on delivering valuable outcomes to its stakeholders.

The Disciplined Agile Mindset

The Disciplined Agile mindset is summarized in Figure 2.1 and is described as a collection of principles, promises, and guidelines. We like to say that we believe in these seven principles, so we promise to one another that we will work in a disciplined manner and follow a collection of guidelines that enable us to be effective.

We Believe in These Principles

Let's begin with the seven principles behind the Disciplined Agile (DA) tool kit. These ideas aren't new; there is a plethora of sources from which these ideas have emerged, including Alistair Cockburn's work around Heart of Agile [CockburnHeart], Joshua Kerievsky's Modern Agile [Kerievsky], and, of course, the *Agile Manifesto for Software Development* described earlier. In fact, the DA tool kit has always been a hybrid of great strategies from the very beginning, with the focus being on how all of these strategies fit together in practice. While we have a strong belief in a scientific approach and what works, we're agnostic as to how we get there. The DA mindset starts with seven fundamental principles:

- Delight customers
- Be awesome
- Context counts
- Be pragmatic
- Choice is good
- Optimize flow
- Organize around products/services

- Enterprise awareness

Principle: Delight Customers

Customers are delighted when our products and services not only fulfill their needs and expectations, but surpass them. Consider the last time you checked into a hotel. If you're lucky there was no line, your room was available, and there was nothing wrong with it when you got there. You were likely satisfied with the service, but that's about it. Now imagine that you were greeted by name by the concierge when you arrived, that your favorite snack was waiting for you in the room, and that you received a complimentary upgrade to a room with a magnificent view—all without asking. This would be more than satisfying and would very likely delight you. Although the upgrade won't happen every time you check in, it's a nice touch when it does and you're likely to stick with that hotel chain because they treat you so well.

Successful organizations offer great products and services that delight their customers. Systems design tells us to build with the customer in mind, to work with them closely, and to build in small increments and then seek feedback, so that we better understand what will actually delight them. As disciplined agilists, we embrace change because we know that our stakeholders will see new possibilities as they learn what they truly want as the solution evolves. We also strive to discover what our customers want and to care for our customers. It's much easier to take care of an existing customer than it is to get a new one.

Jeff Gothelf and Josh Seiden say it best in *Sense & Respond:* "If you can make a product easier to use, reduce the time it takes a customer to complete a task, or provide the right information at the exact moment, you win" [SenseRespond].

Principle: Be Awesome

Who doesn't want to be awesome? Who doesn't want to be part of an awesome team doing awesome things while working for an awesome organization? We all want these things. Recently, Joshua Kerievsky has popularized the concept that modern agile teams make people awesome, and, of course, it isn't much of a leap that we want awesome teams and awesome organizations, too. Similarly, Mary and Tom Poppendieck observe that sustainable advantage is gained from engaged, thinking people, as does Richard Sheridan in *Joy Inc.* [Sheridan]. Helping people to be awesome is important because, as Richard Branson of the Virgin Group says, "Take care of your

employees and they'll take care of your business."

There are several things that we, as individuals, can do to be awesome. First and foremost, act in such a way that we earn the respect and trust of our colleagues: Be reliable, be honest, be open, be ethical, and treat them with respect. Second, willingly collaborate with others. Share information with them when asked, even if it is a work in progress. Offer help when it's needed and, just as important, reach out for help yourself. Third, be an active learner. We should seek to master our craft, always being on the lookout for opportunities to experiment and learn. Go beyond our specialty and learn about the broader software process and business environment. By becoming a T-skilled, "generalizing specialist," we will be able to better appreciate where others are coming from and thereby interact with them more effectively [Agile Modeling]. Fourth, seek to never let the team down. Yes, it will happen sometimes, and good teams understand and forgive that. Fifth, Simon Powers [Powers] points out that we need to be willing to improve and manage our emotional responses to difficult situations. Innovation requires diversity, and by their very nature, diverse opinions may cause emotional reactions. We must all work on making our workplace psychologically safe.

Awesome teams also choose to build quality in from the very beginning. Lean tells us to fix any quality issues and the way we worked that caused them. Instead of debating which bugs we can skip over for later, we want to learn how to avoid them completely. As we're working toward this, we work in such a way that we do a bit of work, validate it, fix any issues that we find, and then iterate. The Agile Manifesto is clear that continuous attention to technical excellence and good design enhances agility [Manifesto].

Senior leadership within our organization can enable staff to be awesome individuals working on awesome teams by providing them with the authority and resources required for them to do their jobs, by building a safe culture and environment (see next principle), and by motivating them to excel. People are motivated by being provided with the autonomy to do their work, having opportunities to master their craft, and to do something that has purpose [Pink]. What would you rather have, staff who are motivated or demotivated?[3]

Principle: Context Counts

Every person is unique, with their own set of skills, preferences for work style, career

goals, and learning styles. Every team is unique not only because it is composed of unique people, but also because it faces a unique situation. Our organization is also unique, even when there are other organizations that operate in the same marketplace that we do. For example, automobile manufacturers such as Ford, Audi, and Tesla all build the same category of product, yet it isn't much of a stretch to claim that they are very different companies. These observations—that people, teams, and organizations are all unique—lead us to a critical idea that our process and organization structure must be tailored for the situation that we currently face. In other words, context counts.

Figure 2.2, adapted from the Software Development Context Framework (SDCF) [SDCF], shows that there are several context factors that affect how a team chooses its WoW. The factors are organized into two categories: factors which have a significant impact on our choice of life cycle (more on this in Chapter 6), and factors that motivate our choice of practices/strategies. The practice/strategy selection factors are a superset of the life cycle-selection factors. For example, a team of eight people working in a common team room on a very complex domain problem in a life-critical regulatory situation will organize themselves differently, and will choose to follow different practices, than a team of 50 people spread out across a corporate campus on a complex problem in a nonregulatory situation. Although these two teams could be working for the same company, they could choose to work in very different ways.

There are several interesting implications of Figure 2.2. First, the further to the right on each selection factor, the greater the risk faced by a team. For example, it's much riskier to outsource than it is to build our own internal team. A team with a lower set of skills is a riskier proposition than a highly skilled team. A large team is a much riskier proposition than a small team. A life-critical regulatory situation is much riskier than a financial-critical situation, which in turn is riskier than facing no regulations at all. Second, because teams in different situations will need to choose to work in a manner that is appropriate for the situation that they face, to help them tailor their approach effectively, we need to give them choices. Third, anyone interacting with multiple teams needs to be flexible enough to work with each of those teams appropriately. For example, we will govern that small, colocated, life-critical team differently than the medium-sized team spread across the campus. Similarly, an enterprise architect who is supporting both teams will collaborate differently with

each.

Scrum provides what used to be solid guidance for delivering value in an agile manner, but it is officially described by only a 19-page booklet [ScrumGuide]. Disciplined Agile recognizes that enterprise complexities require far more guidance, and thus provides a comprehensive reference tool kit for adapting our agile approach for our unique context in a straightforward manner. Being able to adapt our approach for our context with a variety of choices rather than standardizing on one method or framework is a good thing and we explore this further below.

Principle: Be Pragmatic

Many agilists are quite fanatical about following specific methods strictly. In fact, we have met many who say that to "do agile right," we need to have 5–9 people in a room, with the business (product owner) present at all times. The team should not be disturbed by people outside the team and should be 100 % dedicated to the project. However, in many established enterprises, such ideal conditions rarely exist. The reality is that we have to deal with many suboptimal situations, such as distributed teams, large team sizes, outsourcing, multiple team coordination, and part-time availability of stakeholders.

DA recognizes these realities, and rather than saying "we can't be agile" in these situations, we instead say: "Let's be pragmatic and aim to be as effective as we can be." Instead of prescribing "best practices," DA provides strategies for maximizing the benefits of agile despite certain necessary compromises being made. As such, DA is pragmatic, not purist in its guidance. DA provides guardrails to help us make better process choices, not strict rules that may not even be applicable given the context that we face.

Principle: Choice Is Good

Let's assume that our organization has multiple teams working in a range of situations, which in fact is the norm for all but the smallest of companies. How do we define a process that applies to each and every situation that covers the range of issues faced by each team? How do we keep it up to date as each team learns and evolves their approach? The answer is that we can't; documenting such a process is exponentially expensive. But does that mean we need to inflict the same, prescriptive process on everyone? When we do that, we'll inflict process dissonance on our teams, decreasing

their ability to be effective and increasing the chance that they invest resources in making it look as if they're following the process when in reality they're not. Or, does this mean that we just have a "process free-for-all" and tell all our teams to figure it out on their own? Although this can work, it tends to be very expensive and time-consuming in practice. Even with coaching, each team is forced to invent or discover the practices and strategies that have been around for years, sometimes decades.

Developing new products, services, and software is a complex endeavor. That means we can never know for sure what's going to happen. There are many layers of activities going on at the same time and it's hard to see how each relates to the others. Systems are holistic and not understandable just by looking at their components. Instead, we must look at how the components of the system interact with each other. Consider a car, for example. While cars have components, the car itself is also about how the car's components interact with each other. For example, putting a bigger engine in a car might make the car unstable if the frame can't support it, or even dangerous if the brakes are no longer sufficient.

When making improvements to how we work, we must consider the following:

- How people interact with each other;
- How work being done in one part of the system affects the work in others;
- How people learn; and
- How people in the system interact with people outside of the system.

These interactions are unique to a particular organization. The principle of "context counts" means we must make intelligent choices based on the situation we are in. But how? We first recognize that we're not trying to figure out the best way to do things up front, but rather create a series of steps, each either making improvements on what we're doing or by learning something that will increase the likelihood of improvement the next time.

Each step in this series is presented as a hypothesis; that is, a conjecture that it will be an improvement if we can accomplish it. If we get improvement, we're happy and can go on to the next step. If we don't, we should ask why we didn't. Our efforts should lead to either improvement or learning, which then sets up the next improvement action. We can think of this as a scientific approach as we're trying actions and validating them. The cause may be that we took the wrong action, people didn't accept it, or it was beyond our capability.

Here's an example. Let's say that we see our people are multitasking a lot. Multitasking is usually caused by people working on too many things that they are not able to finish quickly. This causes them to go from one task to another and injects delays in their workflow as well as anyone depending upon them. How to stop this multitasking depends on the cause or causes of it. These are often clear or can be readily discerned. Even if we're not sure, trying something based on what's worked in similar situations in the past often achieves good results or learning. The salient aspect of Disciplined Agile is that we use practices that are germane to our situation, and in order to do that we need to know what practices exist that we could choose from.

Different contexts require different strategies. Teams need to be able to own their own process and to experiment to discover what works in practice for them given the situation that they face. As we learned in Chapter 1, DAD provides six life cycles for teams to choose from and 21 process goals that guide us toward choosing the right practices/strategies for our team given the situation that we face. Yes, it seems a bit complicated at first, but this approach proves to be a straightforward strategy to help address the complexities faced by solution delivery teams. Think of DAD, and DA in general, as the scaffolding that supports our efforts in choosing and evolving our WoW.

This choice-driven strategy is a middle way. At one extreme, we have prescriptive methods, which have their place, such as Scrum, Extreme Programming (XP), and SAFe®, which tell us one way to do things. Regardless of what the detractors of these methods claim, these methods/frameworks do in fact work quite well in some situations, and as long as we find ourselves in that situation, they'll work well for use. However, if we're not in the situation where a certain method fits, then it will likely do more harm than good. At the other extreme are creating our own methods by looking at our challenges, creating new practices based on principles, and trying them as experiments and learning as we go. This is how methods[4] that tell us to experiment and learn as we go developed their approach. This works well in practice, but can be very expensive, time-consuming, and can lead to significant inconsistencies between teams, which hampers our overall organizational process. Spotify® had the luxury of evolving their process within the context of a product company, common architecture, no technical debt, and a culture that they could grow rather than change—not to mention several in-house experts. DA sits between these two extremes; by taking this process-goal-driven approach, it provides process commonality between teams that is required at the organizational level, yet provides teams with flexible and

straightforward guidance that is required to tailor and evolve their internal processes to address the context of the situation that they face. Teams can choose—from known strategies—the likely options to then experiment with, increasing the chance that they find something that works for them in practice. At a minimum, it at least makes it clear that they have choices, that there is more than the one way described by the prescriptive methods.

People are often surprised when we suggest that mainstream methods such as Scrum and Extreme Programming (XP) are prescriptive, but they are indeed. Scrum mandates a daily standup meeting (a Scrum), no longer than 15 minutes, to which all team members must attend; that teams must have a retrospective at the end of each iteration (sprint); and that team size should not be more than nine people. Extreme Programming prescribes pair programming (two people sharing one keyboard) and test-driven development (TDD); granted, both of these are great practices in the right context. We are not suggesting that prescription is a bad thing, we're merely stating that it does exist.

In order to provide people with choices from which they can choose their way of working (WoW), DA has gathered strategies and put them into context from a wide array of sources. An important side effect of doing so is that it quickly forced us to take an agnostic approach. In DA, we've combined strategies from methods, frameworks, bodies of knowledge, books, our practical experiences helping organizations to improve, and many other sources. These sources use different terminology, sometimes overlap with each other, have different scopes, are based on different mindsets, and quite frankly often contradict each other. Chapter 3 goes into greater detail about how DA is a hybrid tool kit that provides agnostic process advice. As described earlier, leadership should encourage experimentation early in the interest of learning and improving as quickly as possible. However, we would suggest that by referencing the proven strategies in Disciplined Agile, we will make better choices for our context, speeding up process improvement through failing less. Better choices lead to better outcomes, earlier.

Principle: Optimize Flow

Although agile sprang from lean thinking in many ways, the principles of flow look to be transcending both. Don Reinertsen, in *Principles of Product Development Flow: 2nd Edition.* [Reinertsen], provides more direct actions we can take to accelerate value

realization. Looking at the flow of value enables teams to collaborate in a way as to effectively implement our organization's value streams. Although each team may be but one part of the value stream, they can see how they might align with others to maximize the realization of value.

The implication is that as an organization we need to optimize our overall workflow. DA supports strategies from agile, lean, and flow to do so:

1. **Optimize the whole.** DA teams work in an "enterprise-aware" manner. They realize that their team is one of many teams within their organization and, as a result, they should work in such a way as to do what is best for the overall organization and not just what is convenient for them. More importantly, they strive to streamline the overall process, to optimize the whole as the lean canon advises us to do. This includes finding ways to reduce the overall cycle time—the total time from the beginning to the end of the process to provide value to a customer [Reinertsen].
2. **Measure what counts.** Reinertsen's exhortation, "If you only quantify one thing, quantify the cost of delay," provides an across-the-organization view of what to optimize. "Cost of delay" is the cost to a business in value when a product is delayed. As an organization or as a value stream within an organization, and even at the team level, we will have outcomes that we want to achieve. Some of these outcomes will be customer focused and some will be improvement focused (often stemming from improving customer-focused outcomes). Our measures should be to assist in improving outcomes or in improving our ability to deliver better outcomes.
3. **Deliver small batches of work continuously at a sustainable pace.** Small batches of work not only enable us to get feedback faster, they enable us to not build things of lesser value, which often get thrown into a project. Dr. Goldratt, creator of Theory of Constraints (ToC), once remarked, "Often reducing batch size is all it takes to bring a system back into control" [Goldratt]. By delivering consumable solutions frequently, we can adjust what's really needed and avoid building things that aren't. By "consumable," we mean that it is usable, desirable, and functional (it fulfills its stakeholder's needs). "Solution" refers to something that may include software, hardware, changes to a business process, changes to the organizational structure of the people using the solution, and of

course any supporting documentation.

4. **Attend to delays by managing queues.** By attending to queues (work waiting to be done), we can identify bottlenecks and remove them using concepts from lean, Theory of Constraints, and Kanban. This eliminates delays in workflow that create extra work.
5. **Improve continuously.** Optimizing flow requires continuous learning and improvement. The process goal Evolve WoW (Chapter 24) captures strategies to improve our team's work environment, our process, and our tooling infrastructure over time. Choosing our way of working is done on a continuous basis. This learning is not just how we work but what we are working on. Probably the most significant impact of Eric Ries' work in Lean Startup is the popularization of the experimentation mindset—the application of fundamental concepts of the scientific method to business. This mindset can be applied to process improvement following a guided continuous improvement (GCI) strategy that we described in Chapter 1. Validating our learnings is one of the guidelines of the DA mindset. Improve continuously is also one of the promises that disciplined agilists make to one another (see below).
6. **Prefer long-lived dedicated product teams.** A very common trend in the agile community is the movement away from project teams to cross-functional product teams. This leads us to the next principle: Organize Around Products/Services.

Principle: Organize Around Products/Services

There are several reasons why it is critical to organize around the products and services, or more simply offerings, that we provide to our customers. What we mean by this is that we don't organize around job function, such as having a sales group, a business analysis group, a data analytics group, a vendor management group, a project management group, and so on. The problem with doing so is the overhead and time required to manage the work across these disparate teams and aligning the differing priorities of these teams. Instead, we build dedicated teams focused on delivering an offering for one or more customers. These teams will be cross-functional in that they include people with sales skills, business analysis skills, management skills, and so on.

Organizing around products/services enables us to identify and optimize the flows

that count, which are value streams. We will find that a collection of related offerings will define a value stream that we provide to our customers, and this value stream will be implemented by the collection of teams for those offerings. The value stream layer of the DA tool kit, captured by the DA FLEX life cycle, was described in Chapter 1.

Organizing around products/services enables us to be laser-focused on delighting customers. Stephen Denning calls this the Law of the Customer, that everyone needs to be passionate about and focused on adding value to their customers [Denning]. Ideally, these are external customers, the people or organizations that our organization exists to serve. But sometimes these are also internal customers as well, other groups or people whom we are collaborating with so as to enable them to serve their customers more effectively.

Within a value stream, the industry has found that dedicated cross-functional product teams that stay together over time are the most effective in practice [Kersten]. Having said that, there will always be project-based work as well. Chapter 6 shows that DA supports life cycles that are suited for project teams as well as dedicated product teams. Always remember, choice is good.

Principle: Enterprise Awareness

When people are enterprise aware, they are motivated to consider the overall needs of their organization, to ensure that what they're doing contributes positively to the goals of the organization and not just to the suboptimal goals of their team. This is an example of the lean principle of optimizing the whole. In this case, “the whole” is the organization, or at least the value stream, over local optimization at the team level.

Enterprise awareness positively changes people's behaviors in several important ways. First, they're more likely to work closely with enterprise professionals to seek their guidance. These people—such as enterprise architects, product managers, finance professionals, auditors, and senior executives—are responsible for our organization's business and technical strategies and for evolving our organization's overall vision. Second, enterprise-aware people are more likely to leverage and evolve existing assets within our organization, collaborating with the people responsible for those assets (such as data, code, and proven patterns or techniques) to do so. Third, they're more likely to adopt and follow common guidance, tailoring it where need be, thereby increasing overall consistency and quality. Fourth, they're more likely to share their learnings across teams, thereby speeding up our organization's overall improvement

efforts. In fact, one of the process blades of DA, Continuous Improvement, is focused on helping people to share learnings. Fifth, enterprise-aware people are more likely to be willing to work in a transparent manner although they expect reciprocity from others.

There is the potential for negative consequences as well. Some people believe that enterprise awareness demands absolute consistency and process adherence by teams, not realizing that context counts and that every team needs to make their own process decisions (within bounds or what's commonly called "guard rails"). Enterprise awareness can lead some people into a state of "analysis paralysis," where they are unable to make a decision because they're overwhelmed by the complexity of the organization.

We Promise To

Because disciplined agilists believe in the principles of DA, they promise to adopt behaviors that enable them to work both within their team and with others more effectively. These promises are designed to be synergistic in practice, and they have positive feedback cycles between them. The promises of the DA mindset are:

1. Create psychological safety and embrace diversity.
2. Accelerate value realization.
3. Collaborate proactively.
4. Make all work and workflow visible.
5. Improve predictability.
6. Keep workloads within capacity.
7. Improve continuously.

Promise: Create Psychological Safety and Embrace Diversity

Psychological safety means being able to show and employ oneself without fear of negative consequences of status, career, or self-worth—we should be comfortable being ourselves in our work setting. A 2015 study at Google found that successful teams provide psychological safety for team members, that team members are able to depend on one another, there is structure and clarity around roles and responsibilities, and people are doing work that is both meaningful and impactful to them [Google].

Psychological safety goes hand-in-hand with diversity, which is the recognition that everyone is unique and can add value in different ways. The dimensions of personal uniqueness include, but are not limited to, race, ethnicity, gender, sexual orientation, agile, physical abilities, socioeconomic status, religious beliefs, political beliefs, and other ideological beliefs. Diversity is critical to a team's success because it enables greater innovation. The more diverse our team, the better our ideas will be, the better our work will be, and the more we'll learn from each other.

There are several strategies that enable us to nurture psychological safety and diversity within a team:

1. **Be respectful.** Everyone is different, with different experiences and different preferences. None of us is the smartest person in the room. Respect what other people know that we don't and recognize that they have a different and important point of view.
2. **Be humble.** In many ways, this is key to having a learning mindset and to being respectful.
3. **Be ethical and trustworthy.** People will feel safer working and interacting with us if they trust us. Trust is built over time through a series of actions and can be broken instantly by one action.
4. **Make it safe to fail.** There is a catchy phrase in the agile world called "fail fast." We prefer Al Shalloway's advice, "Make it safe to fail so you can learn fast." The idea is to not hesitate to try something, even if it may fail. But the focus should be on learning safely and quickly. Note that "safely" refers both to psychological safety and the safety of our work. As we learned in Chapter 1, the aim of guided continuous improvement (GCI) is to try out new ways of working (WoW) with the expectation that they will work for us, while being prepared to learn from our experiment if it fails.

Promise: Accelerate Value Realization

An important question to ask is: What is value? Customer value, something that benefits the end customer who consumes the product/service that our team helps to provide, is what agilists typically focus on. This is clearly important, but in Disciplined Agile, we're very clear that teams have a range of stakeholders, including external end customers. So, shouldn't we provide value to them as well?

Mark Schwartz, in *The Art of Business Value*, distinguishes between two types of value: customer value and business value [Schwartz]. Business value addresses the issue that some things are of benefit to our organization and perhaps only indirectly to our customers. For example, investing in enterprise architecture, in reusable infrastructure, and in sharing innovations across our organization offer the potential to improve consistency, quality, reliability, and reduce cost over the long term. These things have great value to our organization but may have little direct impact on customer value. Yet, working in an enterprise-aware manner such as this is clearly a very smart thing to do.

There are several ways that we can accelerate value realization:

1. **Work on small, high-value items.** By working on the most valuable thing right now, we increase the overall return on investment (ROI) of our efforts. By working on small things and releasing them quickly, we reduce the overall cost of delay and our feedback cycle by getting our work into the hands of stakeholders quickly. This is a very common strategy in the agile community and is arguably a fundamental of agile.
2. **Reuse existing assets.** Our organization very likely has a lot of great stuff that we can take advantage of, such as existing tools, systems, sources of data, standards, and many other assets. But we need to choose to look for them, we need to be supported in getting access to them and in learning about them, and we may need to do a bit of work to improve upon the assets to make them fit our situation. One of the guidelines of the DA mindset, described later in this chapter, is to leverage and enhance organizational assets.
3. **Collaborate with other teams.** An easy way to accelerate value realization is to work with others to get the job done. Remember the old saying: Many hands make light work.

Promise: Collaborate Proactively

Disciplined agilists strive to add value to the whole, not just to their individual work or to the team's work. The implication is that we want to collaborate both within our team and with others outside our team, and we also want to be proactive doing so. Waiting to be asked is passive, observing that someone needs help and then volunteering to do so is proactive. We have observed that are three important

opportunities for proactive collaboration:

1. **Within our team.** We should always be focused on being awesome, on working with and helping out our fellow team members. So if we see that someone is overloaded with work or is struggling to work through something, don't just wait to be asked but instead volunteer to help out.
2. **With our stakeholders.** Awesome teams have a very good working relationship with their stakeholders, collaborating with them to ensure that what they do is what the stakeholders actually need.
3. **Across organizational boundaries.** In Chapter 1, we discussed how an organization is a complex adaptive system (CAS) of teams interacting with other teams.

Promise: Make All Work and Workflow Visible

Disciplined Agile teams—and individual team members—make all their work and how they are working visible to others.[5] This is often referred to as "radical transparency" and the idea is that we should be open and honest with others. Not everyone is comfortable with this. Organizations with traditional methods have a lot of watermelon projects—green on the outside and red on the inside—by which we mean that they claim to be doing well even though they're really in trouble. Transparency is critical for both supporting effective governance, a topic covered in greater detail in Chapter 27, and for enabling collaboration as people are able to see what others are currently working on.

Disciplined agile teams will often make their work visible at both the individual level as well as the team level. It is critical to focus on our work in process, which is more than the work in progress. Work in progress is what we are currently working on. Work in process is our work in progress plus any work that is queued up waiting for us to get to it. Disciplined agilists focus on work in process as a result.

Disciplined agile teams make their workflow visible, and thus have explicit workflow policies, so that everyone knows how everyone else is working. This supports collaboration because people have agreements as to how they are going to work together. It also supports process improvement because it enables us to understand what is actually happening and thereby increases the chance that we can detect where we have potential issues. It is important that we are both agnostic and pragmatic in the

way that we work, as we want to do the best that we can in the context that we face.

Promise: Improve Predictability

Disciplined agile teams strive to improve their predictability to enable them to collaborate and self-organize more effectively, and thereby to increase the chance that they will fulfill any commitments that they make to their stakeholders. Many of the earlier promises we have made work toward improving predictability. To see how to improve predictability, it is often useful to see what causes unpredictability, such as technical debt and overloaded team members, and to then attack those challenges.

Common strategies to improve predictability include:

- **Pay down technical debt.** Technical debt refers to the implied cost of future refactoring or rework to improve the quality of an asset to make it easy to maintain and extend. When we have significant technical debt, it becomes difficult to predict how much effort work will be—working with high-quality assets is much easier than working with low-quality assets. Because most technical debt is hidden (we don't really know what invokes that source code we're just about to change or we don't know what's really behind that wall we're about to pull down as we renovate our kitchen), it often presents us with unpredictable surprises when we get into the work. Paying down technical debt, described by the process goal Improve Quality (Chapter 18), is an important strategy for increasing the predictability of our work.
- **Respect work-in-process (WIP) limits.** When people are working close to or at their maximum capacity then it becomes difficult to predict how long something will take to accomplish. Those two days' worth of work might take me three months to accomplish because I either let it sit in my work queue for three months or I do a bit of the work at a time over a three-month period. Worse yet, the more loaded someone becomes, the more their feedback cycles will increase in length, generating even more work for them (see below) and thus increasing their workload further. So we want to keep workloads within capacity, another one of our promises.
- **Adopt a test-first approach.** With a test-first approach, we think through how we will test something before we build it. This has the advantage that our tests both specify as well as validate our work, thereby doing double duty, which

will very likely motivate us to create a higher quality work product. It also increases our predictability because we will have a better understanding of what we're working on before actually working on it. There are several common practices that take a test-first approach, including acceptance test-driven development (ATDD) where we capture detailed requirements via working acceptance tests, and test-driven development (TDD) where our design is captured as working developer tests. These techniques are described in greater detail in Chapters 9 and 17, respectively.

- **Reduce feedback cycles.** A feedback cycle is the amount of time between doing something and getting feedback about it. For example, if we write a memo and then send it to someone to see what they think, and it then takes four days for them to get back to us, the feedback cycle is four days long. But, if we work collaboratively and write the memo together, a technique called pairing, then the feedback cycle is on the order of seconds because they can see what we type and discuss it as we're typing. Short feedback cycles enable us to act quickly to improve the quality of our work, thereby improving our predictability and increasing the chance that we will delight our customers. Long feedback cycles are problematic because the longer it takes to get feedback, the greater the chance that any problems we have in our work will be built upon, thereby increasing the cost of addressing any problems because now we need to fix the original problem and anything that extends it. Long feedback cycles also increase the chance that the requirement for the work will evolve, either because something changed in the environment or because someone simply changed their mind about what they want. In both cases, the longer feedback cycle results in more work for us to do and thereby increases our workload (as discussed earlier).

Promise: Keep Workloads Within Capacity

Going beyond capacity is problematic from both a personal and a productivity point of view. At the personal level, overloading a person or team will often increase the frustration of the people involved. Although it may motivate some people to work harder in the short term, it will cause burnout in the long term, and it may even motivate people to give up and leave because the situation seems hopeless to them. From a productivity point of view, overloading causes multitasking, which increases

overall overhead. We can keep workloads within capacity by:

- **Working on small batches.** Having small batches of work enables us to focus on getting the small batch done and then move on to the next small batch.
- **Having properly formed teams.** Teams that are cross-functional and sufficiently staffed increase our ability to keep workload within capacity because it reduces dependencies on others. The more dependencies we have, the less predictable our work becomes and therefore is harder to organize. Chapter 7 describes how to form teams effectively.
- **Take a flow perspective.** By looking at the overall workflow we are part of, we can identify where we are over capacity by looking for bottlenecks where work is queuing up. We can then adjust our WoW to alleviate the bottleneck, perhaps by shifting people from one activity to another where we need more capacity, or improving our approach to the activity where we have the bottleneck. Our aim, of course, is to optimize flow across the entire value stream that we are part of, not to just locally optimize our own workflow.
- **Use a pull system.** One of the advantages of pulling work when we are ready is that we can manage our own workload level.

Promise: Improve Continuously

The really successful organizations—Apple, Amazon, eBay, Facebook, Google, and more—got that way through continuous improvement. They realized that to remain competitive, they needed to constantly look for ways to improve their processes, the outcomes that they were delivering to their customers, and their organizational structures. This is why these organizations adopt a kaizen-based approach of improving via small changes. In Chapter 1, we learned that we can do even better than that by taking a guided continuous improvement (GCI) approach that leverages the knowledge base contained within the DA tool kit.

Continuous improvement requires us to have agreement on what we're improving. We've observed that teams that focus on improving on the way that they fulfill the promises described here, including improving on the way that they improve, tend to improve faster than those that don't. Our team clearly benefits by increasing safety and diversity, improving collaboration, improving predictability, and keeping their workload within capacity. Our organization also benefits from these things, as well as

when we improve upon the other promises.

We Follow These Guidelines

To fulfill the promises that disciplined agilists make, they will choose to follow a collection of guidelines that make them more effective in the way that they work. The guidelines of the DA mindset are:

1. Validate our learnings.
2. Apply design thinking.
3. Attend to relationships through the value stream.
4. Create effective environments that foster joy.
5. Change culture by improving the system.
6. Create semi-autonomous, self-organizing teams.
7. Adopt measures to improve outcomes.
8. Leverage and enhance organizational assets.

Guideline: Validate Our Learnings

The only way to become awesome is to experiment with, and then adopt where appropriate, a new WoW. In the GCI workflow, after we experiment with a new way of working, we assess how well it worked, an approach called validated learning. Hopefully, we discover that the new WoW works for us in our context, but we may also discover that it doesn't. Either way, we've validated what we've learned. Being willing and able to experiment is critical to our process-improvement efforts. Remember Mark Twain's aphorism: “It ain't what you don't know that gets you into trouble. It's what you know for sure that just ain't so.”

Validated learning isn't just for process improvement, we should also apply this strategy to the product/service (offering) that we are providing to our customers. We can build in thin slices, make changes available to our stakeholders, and then assess how well that change works in practice. We can do this through demoing our offering to our stakeholders or, better yet, releasing our changes to actual end users and measuring whether they benefited from these changes.

Guideline: Apply Design Thinking

Delighting customers requires us to recognize that our work is to create operational value streams for our customers that are designed with them in mind. This requires design thinking on our part. Design thinking means to be empathetic to the customer, to first try to understand their environment and needs before developing a solution. Design thinking represents a fundamental shift from building systems from our perspective to creatively solving customer problems and, better yet, fulfilling needs they didn't even know they had.

Design thinking is an exploratory approach that should be used to iteratively explore a problem space and identify potential solutions for it. Design thinking has its roots in user-centered design as well as usage-centered design, both of which influenced Agile Modeling, one of many methods that the DA tool kit adopts practices from. In Chapter 6, we will learn that DA includes the Exploratory life cycle, which is specifically used for exploring a new problem space.

Guideline: Attend to Relationships Through the Value Stream

One of greatest strengths of the Agile Manifesto is its first value: Individuals and interactions over processes and tools. Another strength is the focus on teams in the principles behind the manifesto. However, the unfortunate side effect of this takes the focus away from the interactions between people on different teams or even in different organizations. Our experience, and we believe this is what the authors of the manifesto meant, is that the interactions between the people doing the work are what is key, regardless of whether or not they are part of the team. So, if a product manager needs to work closely with our organization's data analytics team to gain a better understanding of what is going on in the marketplace, and our strategy team to help put those observations into context, then we want to ensure that these interactions are effective. We need to proactively collaborate between these teams to support the overall work at hand.

Caring for and maintaining healthy interactive processes is important for the people involved and should be supported and enabled by our organizational leadership. In fact, there is a leadership strategy called middle-up-down management [Nonaka], where management looks “up” the value stream to identify what is needed, enables the team to fulfill that need, and works with the teams downstream to coordinate work effectively. The overall goal is to coordinate locally in a manner that supports optimizing the overall workflow.

Guideline: Create Effective Environments That Foster Joy

To paraphrase the Agile Manifesto, awesome teams are built around motivated individuals who are given the environment and support required to fulfill their objectives. Part of being awesome is having fun and being joyful. We want working in our company to be a great experience, so we can attract and keep the best people. Done right, work is play.

We can make our work more joyful by creating an environment that allows us to work together well. A key strategy to achieve this is to allow teams to be self-organizing—to let them choose and evolve their own WoW, organizational structure, and working environments. Teams must do so in an enterprise-aware manner—meaning we need to collaborate with other teams, and there are organizational procedures and standards we must follow and constraints on what we can do. The job of leadership is to provide a good environment for teams to start in and then to support and enable teams to improve as they learn over time.

Guideline: Change Culture by Improving the System

Peter Drucker is famous for saying that "culture eats strategy for breakfast." This is something that the agile community has taken to heart, and this philosophy is clearly reflected in the people-oriented nature of the Agile Manifesto. While culture is important, and culture change is a critical component of any organization's agile transformation, the unfortunate reality is that we can't change it directly. This is because culture is a reflection of the management system in place, so to change our culture, we need to evolve our overall system.

From a systems point of view, the system is both the sum of its components plus how they interact with each other [Meadows]. In the case of an organization, the components are the teams/groups within it and the tools and other assets, both digital and physical, that they work with. The interactions are the collaborations of the people involved, which are driven by the roles and responsibilities that they take on and their WoW. To improve a system, we need to evolve both its components and the interactions between those components in lock step.

To improve the components of our organizational system, we need to evolve our team structures and the tools/assets that we use to do our work. The next DA mindset guideline, create semi-autonomous, self-organizing teams, addresses the team side of

this. In Chapter 18, we describe options for improving the quality of our infrastructure, which tends to be a long-term endeavor requiring significant investment. To improve the interactions between components, which is the focus of this book, we need to evolve the roles and responsibilities of the people working on our teams and enable them to evolve their WoW.

To summarize, if we improve the system, then culture change will follow. To ensure that culture change is positive, we need to take a validated learning approach to these improvements.

Guideline: Create Semi-Autonomous, Self-Organizing Teams

Organizations are complex adaptive systems (CASs) made up of a network of teams or, if you will, a team of teams. Although mainstream agile implores us to create "whole teams" that have all of the skills and resources required to achieve the outcomes that they've been tasked with, the reality is that no team is an island unto itself. Autonomous teams would be ideal but there are always dependencies on other teams upstream that we are part of, as well as downstream from us. And, of course, there are dependencies between offerings (products or services) that necessitate the teams responsible for them to collaborate. This network-of-teams organizational structure is being recommended by Stephen Denning in his Law of the Network [Denning], Mik Kersten in his recommendation to shift from project to product teams [Kersten], John Kotter in Accelerate [Kotter], Stanley McChrystal in his team-of-teams strategy [MCSF], and many others.

Teams will proactively collaborate with other teams on a regular basis, one of the promises of the DA mindset. Awesome teams are as whole as possible—they are cross-functional; have the skills, resources, and authority required to be successful; and team members themselves tend to be cross-functional generalizing specialists. Furthermore, they are organized around the products/services offered by the value stream they are part of. Interestingly, when we have teams dedicated to business stakeholders, budgeting becomes much simpler because we just need to budget for the people aligned with each product/service.

Creating semi-autonomous teams is great start, but self-organization within the context of the value stream is also something to attend to. Teams will be self-organizing, but they must do so within the context of the overall workflow that they are part of. Remember the principles, Optimize Flow and Enterprise Awareness, in that

teams must strive to do what's right for the overall organization, not just what is convenient for them. When other teams also work in such a way, we are all much better for it.

Guideline: Adopt Measures to Improve Outcomes

When it comes to measurement, context counts. What are we hoping to improve? Quality? Time to market? Staff morale? Customer satisfaction? Combinations thereof? Every person, team, and organization has their own improvement priorities, and their own ways of working, so they will have their own set of measures that they gather to provide insight into how they're doing and, more importantly, how to proceed. And these measures evolve over time as their situation and priorities evolve. The implication is that our measurement strategy must be flexible and fit for purpose, and it will vary across teams. The Govern Delivery Team process goal (Chapter 27) provides several strategies, including goal question metric (GQM) [W] and objectives and key results (OKRs) [W], that promote context-driven metrics.

Metrics should be used by a team to provide insights into how they work and provide visibility to senior leadership to govern the team effectively. When done right, metrics will lead to better decisions which in turn will lead to better outcomes. When done wrong, our measurement strategy will increase the bureaucracy faced by the team, will be a drag on their productivity, and will provide inaccurate information to whomever is trying to govern the team. Here are several heuristics, described in detail in Chapter 27, to consider when deciding on the approach to measuring our team:

- Start with outcomes.
- Measure what is directly related to delivering value.
- There is no “one way” to measure; teams need fit-for-purpose metrics.
- Every metric has strengths and weaknesses.
- Use metrics to motivate, not to compare.
- We get what we measure.
- Teams use metrics to self-organize.
- Measure outcomes at the team level.
- Each team needs a unique set of metrics.
- Measure to improve; we need to measure our pain so we can see our gain.

- Have common metric categories across teams, not common metrics.
- Trust but verify.
- Don't manage to the metrics.
- Automate wherever possible so as to make the metrics ungameable.
- Prefer trends over scalars.
- Prefer leading over trailing metrics.
- Prefer pull over push.

Guideline: Leverage and Enhance Organizational Assets

Our organization has many assets—information systems, information sources, tools, templates, procedures, learnings, and other things—that our team could adopt to improve our effectiveness. We may not only choose to adopt these assets, we may also find that we can improve them to make them better for us as well as other teams who also choose to work with these assets. This guideline is important for several reasons:

1. **A lot of good work has occurred before us.** There is a wide range of assets within our organization that our team can leverage. Sometimes we will discover that we need to first evolve the existing asset so that it meets our needs, which often proves faster and less expensive than building it from scratch.
2. **A lot of good work continues around us.** Our organization is a network of semi-autonomous, self-organizing teams. We can work with and learn from these teams, proactively collaborating with them, thereby accelerating value realization. The enterprise architecture team can help point us in the right direction and we can help them learn how well their strategies work when applied in practice. Stephen Denning stresses the need for the business operations side of our organization, such as vendor management, finance, and people management, to support the teams executing the value streams of our organization [Denning]. We must work and learn together in an enterprise-aware manner if we are to delight our customers.
3. **We can reduce overall technical debt.** The unfortunate reality is that many organizations struggle under significant technical debt loads, as we discussed earlier. By choosing to reuse existing assets, and investing in paying down some of the technical debt that we run into when doing so, we'll slowly dig our way

out of the technical debt trap that we find ourselves in.

4. **We can provide greater value quicker.** Increased reuse enables us to focus on implementing new functionality to delight our customers instead of just reinventing what we're already offering them. By paying down technical debt, we increase the underlying quality of the infrastructure upon which we're building, enabling us to deliver new functionality faster over time.
5. **We can support others.** Just like our team collaborates with and learns from other teams, so do those other teams collaborate and learn from us. At the organizational level, we can enhance this through the creation of centers of excellence (CoEs) and communities of practice (CoPs) to capture and share learnings across the organization. CoEs and CoPs are two of many strategies described in Chapter 24.

#JoinTheRebellions!

Agile itself is a rebellion against traditional strategies, which for the most part were based on theory, most of which has been shown to be false. But like all rebellions, the agile thinking of the 1990s has become stale. Predictably, a new generation of rabble rousers has come along with their ideas and, in some cases, movements.

Woody Zuill and Neil Killick started what we call the "hashtag rebellions" with their #NoEstimates movement. Since then, #NoProjects [NoProjects], along with other movements described in Table 2.1, have appeared. We believe there are some very interesting and practical strategies coming out of these movements, many of which are captured in DA.

Are these hashtag rebellions good or bad? We think both. Our premise is that it depends because #ContextCounts. We also feel that it's unfortunate that these hashtags are negative in the sense that they're against something rather than for something, but we also recognize that they have been very effective in drawing attention to significant problems in the software process space. Most importantly, they represent a key agile philosophy to question the status quo, to always ask if that's really the way it needs to be.

Table 2.1: Common hashtag rebellions and their visions.

Hashtag	The Vision

#NoEstimates	Estimates are a source of waste because they don't add real value for stakeholders; they're rarely accurate to begin with, and when we deploy regularly, people stop asking for them anyway. See the process goals Plan the Release in Chapter 11 and Accelerate Value Delivery in Chapter 19 for options.
#NoFrameworks	This is pushback against the agile scaling frameworks that experienced agilists find too restrictive and ineffective. More accurately, this should be #NoPrescriptiveFrameworks, but that's just too long to tweet. While DA is arguably a framework (being a collection of good options to consider experimenting with), it is very different than the prescriptive scaling frameworks that many organizations are struggling to succeed with. Instead, we call DA a tool kit.
#NoProjects	This is based on the observation that it is better to flow constant value delivery to our stakeholders, rather than batch up blobs of value that may or may not be worthwhile. It's important to note that this move away from project management in the agile community is not a move away from management, but instead from the inherent risks and overhead of projects.
#NoTemplates	Following a template blindly is wrong, as the applicability may be wrong. But selecting templates that suit context can both accelerate delivery and improve quality. See the process goal Accelerate Value Delivery in Chapter 19.

And a Few More Great Philosophies

Here are a few philosophies that we've seen work well in practice for disciplined agilists:

1. **If it's hard, do it more often.** You believe system integration testing (SIT) is hard? Instead of pushing it to the end of the life cycle, like traditionalists do, find a way to do it every single iteration. Then find a way to do it every single day. Doing hard things more often forces us to find ways, often through automation, to make them easy.
2. **If it's scary, do it more often.** We're afraid to evolve a certain piece of code? We're afraid to get feedback from stakeholders because they may change their minds? Then let's do it more often and find ways to overcome what we fear. Find ways to avoid the negative outcomes, or to turn them positive. Fix that code. Make it easier to evolve our solution. Help those stakeholders understand the implications of the decisions they're making.

3. **Keep asking why.** To truly understand something, we need to ask why it happened, why it works that way, or why it's important to others. Then ask why again, and again, and again. Toyota calls this practice 5 whys analysis [Liker], but don't treat five as a magic number. We keep asking why until we get to the root cause.
4. **Learn something every day.** Disciplined agilists strive to learn something every day. Perhaps it's something about the domain they're working in. Perhaps it's something about the technologies, or something about their tools. Perhaps it's a new practice, or a new way to perform a practice. There are a lot of learning opportunities before us. Take them.

In Summary

How can we summarize the Disciplined Agile mindset? Simon Powers sums up the mindset in terms of three core beliefs [Powers]. These beliefs are:

1. **The complexity belief.** Many of the problems that we face are complex adaptive problems, meaning by trying to solve these problems we change the nature of the problem itself.
2. **The people belief.** Individuals are both independent from and dependent on their teams and organizations. Human beings are interdependent. Given the right environment (safety, respect, diversity, and inclusion) and a motivating purpose, it is possible for trust and self-organization to arise. For this to happen, it is necessary to treat everyone with unconditional positive regard.
3. **The proactive belief.** Proactivity is found in the relentless pursuit of improvement.

We find these beliefs compelling. In many ways, they summarize the fundamental motivations behind why we need to choose our WoW. Because we face a unique context, we need to tailor our WoW, and in doing so, we change the situation that we face that also requires us to learn and evolve our WoW. The people belief motivates us to find a WoW that enables us to work together effectively and safely, and the proactive belief reflects the idea that we should continuously learn and improve.

Mindset Is Only the Beginning

The Disciplined Agile mindset provides a solid foundation from which our organization

can become agile, but it is only a foundation. Our fear is that too many inexperienced coaches are dumbing down agile, hoping to focus on the concepts overviewed in this chapter. It's a good start, but it doesn't get the job done in practice. It isn't sufficient to "be agile," we also need to know how to "do agile." It's wonderful when someone wants to work in a collaborative, respectful manner, but if they don't actually know how to do the work, they're not going to get much done. Software development, and more importantly solution delivery, is complex—we need to know what we're doing.

---

[3] If you think happy employees are expensive, wait until you try unhappy ones!

[4] Spotify, like other methods, is a great source of potential ideas that we've mined in DA. We've particularly found their experimental approach to process improvement, which we've evolved into guided experiments (Chapter 1), to be useful. Unfortunately, many organizations try to adopt the Spotify method verbatim, which is exactly what the Spotify people tell us not to do. The Spotify method was great for them in their context several years ago. They are clear that if we are copying what they did then, that is not Spotify now. Our context, even if we happen to be a Swedish online music company, is different.

[5] This, of course, may be constrained by the need to maintain secrecy, resulting either from competitive or regulatory concerns.

3 DISCIPLINED AGILE DELIVERY (DAD) IN A NUTSHELL

Discipline is doing what you know needs to be done, even if you don't want to do it.—Unknown

Key Points in This Chapter

- DAD is the delivery portion of the Disciplined Agile (DA) tool kit—it is not just another methodology.
- If you are using Scrum, XP, or Kanban, you are already using variations of a subset of DAD.
- DAD provides six life cycles to choose from, it doesn't prescribe a single way of working—choice is good.
- DAD addresses key enterprise concerns.
- DAD does the process heavy lifting so that you don't have to.
- DAD shows how agile development works from beginning to end.
- DAD provides a flexible foundation from which to tactically scale mainstream methods.
- It is easy to get started with DAD.
- You can start with your existing WoW and then apply DAD to improve it gradually. You don't need to make a risky "big bang" change.

Many organizations start their agile journey by adopting Scrum because it describes a good strategy for leading agile software teams. However, Scrum is a very small part of what is required to deliver sophisticated solutions to your stakeholders. Invariably, teams need to look to other methods to fill in the process gaps that Scrum purposely ignores, and Scrum is very clear about this. When looking at other methods, there is considerable overlap and conflicting terminology that can be confusing to practitioners as well as outside stakeholders. Worse yet, people don't always know where to look for advice or even know what issues they need to consider.

To address these challenges, Disciplined Agile Delivery (DAD) provides a more cohesive approach to agile solution delivery. DAD is a people-first, learning-oriented, hybrid agile approach to IT solution delivery. These are the critical aspects of DAD:

1. **People first.** People, and the way we work together, are the primary determinant of success for a solution delivery team. DAD supports a robust set of roles, rights, and responsibilities that you can tailor to meet the needs of

your situation.

2. **Hybrid.** DAD is a hybrid tool kit that puts great ideas from Scrum, SAFe, Spotify, Agile Modeling (AM), Extreme Programming (XP), Unified Process (UP), Kanban, Lean Software Development, and several other methods into context.
3. **Full-delivery life cycle.** DAD addresses the full-delivery life cycle, from team initiation all the way to delivering a solution to your end users.
4. **Support for multiple life cycles.** DAD supports agile, lean, continuous delivery, exploratory, and large-team versions of the life cycle. DAD doesn't prescribe a single life cycle because it recognizes that one process approach does not fit all. Chapter 6 explores life cycles in greater detail, providing advice for selecting the right one to start with and then how to evolve from one to another over time.
5. **Complete.** DAD shows how development, modeling, architecture, management, requirements/outcomes, documentation, governance, and other strategies fit together in a streamlined whole. DAD does the "process heavy lifting" that other methods leave up to you.
6. **Context-sensitive.** DAD promotes what we call a goal-driven or outcome-driven approach. In doing so, DAD provides contextual advice regarding viable alternatives and their trade-offs, enabling you to tailor DAD to effectively address the situation in which you find yourself. By describing what works, what doesn't work, and more importantly why, DAD helps you to increase your chance of adopting strategies that will work for you and do so in a streamlined manner. Remember the DA principle: Context Counts.
7. **Consumable solutions over working software.** Potentially shippable software is a good start, but what we really need are consumable solutions that delight our customers.
8. **Self-organization with appropriate governance.** Agile and lean teams are self-organizing, which means that the people who do the work are the ones who plan and estimate it. But that doesn't mean they can do whatever they want. They must still work in an enterprise-aware manner that reflects the priorities of their organization, and to do that they will need to be governed appropriately by senior leadership. The Govern Delivery Team process goal of

Chapter 27 describes options for doing exactly that.

This chapter provides a brief overview of DAD, with the details coming in later chapters.

What's New With DAD?

For existing DAD practitioners, there are several exciting changes that you'll see in this book compared to *Disciplined Agile Delivery* [AmblerLines2012]. We've made these changes based on our work at dozens of organizations worldwide and, more importantly, from the input we've received from a myriad of practitioners. These changes are:

1. **The process goals have been refactored.** Over the past six years, we've renamed several goals, introduced a new goal, and combined two pairs of goals. We believe it will make the goals more understandable.
2. **Every goal has been updated.** We've learned a lot over the last six years, a lot of great techniques have appeared, and we've applied older techniques in new situations. We've been posting updates to the goals online at DisciplinedAgileDelivery.com and in our courseware, but this is the first time we've captured all of the updates in print.
3. **All of the goals are captured visually.** This is the first book to capture all of DAD's goal diagrams. We introduced the goal diagrams after the 2012 book came out, although we have published some of them in our short book, *Introduction to Disciplined Agile Delivery*, 2nd edition [LinesAmbler2018], and *An Executive Guide to Disciplined Agile* [AmblerLines2017].
4. **New and updated life cycles.** We've explicitly introduced the Program life cycle (we had described it in terms of team structure before) and the Exploratory life cycle. We've also introduced both agile and lean versions of what we used to call the Continuous Delivery life cycle.
5. **Advice for applying the tool kit in practice.** A big difference you'll see in this book is much more advice for how to apply DA in practice. This advice reflects an additional six years of working with organizations around the world to adopt Disciplined Agile strategies.

People First: Roles, Rights, and Responsibilities

Figure 3.1 shows the potential roles that people will fill on DAD teams, and Chapter 4 describes them in detail. The roles are organized into two categories: primary roles that we find are critical to the success of any agile team and supporting roles that appear as needed.

The primary roles are:

- **Team lead.** This person leads the team, helping the team to be successful. This is similar to the scrum master role in Scrum [ScrumGuide].
- **Product owner (PO).** A product owner is responsible for working with stakeholders to identify the work to be done, prioritize that work, help the team to understand the stakeholders' needs, and help the team interact effectively with stakeholders [ScrumGuide].
- **Architecture owner (AO).** An architecture owner guides the team through architecture and design decisions, working closely with the team lead and product owner when doing so [AgileModeling].
- **Team member.** Team members work together to produce the solution. Ideally, team members are generalizing specialists, or working on becoming so, who are often referred to as cross-skilled people. A generalizing specialist is someone with one or more specialities (such as testing, analysis, programming, etc.) and a broad knowledge of solution delivery and the domain they are working in [GenSpec].
- **Stakeholder.** A stakeholder is someone who will be affected by the work of the team, including but not limited to end users, support engineers, operations staff, financial people, auditors, enterprise architects, and senior leadership. Some agile methods call this role customer.

The supporting roles are:

- **Specialist.** Although most team members will be generalizing specialists, or at least striving to be so, we sometimes have specialists on teams when called for. User experience (UX) and security experts are specialists who may be on a team when there is significant user interface (UI) development or security concerns respectively. Sometimes business analysts are needed to support product owners in dealing with a complex domain or geographically distributed stakeholders. Furthermore, roles from other parts of the DA tool

kit such as enterprise architects, portfolio managers, reuse engineers, operations engineers, and others are considered specialists from a DAD point of view.

- **Independent tester.** Although the majority of testing, if not all of it, should be performed by the team, there can be a need for an independent test team at scale. Common scenarios requiring independent testers include: regulatory compliance that requires that some testing occur outside of the team, and a large program (a team of teams) working on a complex solution that has significant integration challenges.
- **Domain expert.** A domain expert, sometimes called a subject matter expert (SME), is someone with deep knowledge in a given domain or problem space. They often work with the team or product owners to share their knowledge and experience.
- **Technical expert.** This is someone with deep technical expertise who works with the team for a short time to help them overcome a specific technical challenge. For example, an operational database administrator (DBA) may work with the team to help them set up, configure, and learn the fundamentals of a database.
- **Integrator.** Also called a system integrator, they will often support independent testers who need to perform system integration testing (SIT) of a complex solution or collection of solutions.

Everyone on agile teams has rights and responsibilities. Everyone. For example, everyone has the right to be given respect, but they also have the responsibility to give respect to others. Furthermore, each role on an agile team has specific additional responsibilities that they must fulfill. Rights and responsibilities are also covered in detail in Chapter 4.

A Hybrid of Great Ideas

We like to say that DAD does the heavy process lifting so that you don't have to. What we mean by that is that is we've mined the various methods, frameworks, and other sources to identify potential practices and strategies that your team may want to experiment with and adopt. We put these techniques into context, exploring fundamental concepts such as what are the advantages and disadvantages of the

technique, when would you apply the technique, when wouldn't you apply the technique, and to what extent would you apply it? Answers to these questions are critical when a team is choosing its WoW.

Figure 3.2 indicates some of the methodologies and frameworks that we've mined for techniques. For example, XP is the source of technical practices such as test-driven development (TDD), refactoring, and pair programming to name a few. Scrum is the source of strategies such as product backlogs, sprint/iteration planning, daily coordination meetings, and more. Agile Modeling gives us model storming, initial architecture envisioning, continuous documentation, and active stakeholder participation. Where these methods go into detail about these individual techniques, the focus of DAD, and DA in general, is to put them into context and to help you choose the right strategy at the right time.

Choice Is Good: Process Goals

DAD includes a collection of 21 process goals, or process outcomes if you like, as Figure 3.3 shows. Each goal is described as a collection of decision points, issues that your team needs to determine whether they need to address and, if so, how they will do so. Potential practices/strategies for addressing a decision point, which can be combined in many cases, are presented as lists. Goal diagrams, an example is shown in Figure 3.4, are similar conceptually to mind maps, albeit with the extension of the arrow to represent the relative effectiveness of options in some cases. Goal diagrams are, in effect, guides to help a team to choose the best strategies that they are capable of doing right now given their skills, culture, and situation. Chapter 5 explores DAD's goal-driven approach and Sections 2–5 describe each goal in detail.

Choice Is Good: Multiple Life Cycle Support

Life cycles put an order to the activities that a team performs to build a solution. In effect, they organize the techniques that we apply to get the work done. Because solution delivery teams find themselves in a range of different situations, they need to be able to choose a life cycle that best fits the context that they face. You can see in Figure 3.5 that DAD supports six life cycles:

1. **Agile.** This is a Scrum-based life cycle for solution delivery projects.

2. **Lean.** This is a Kanban-based life cycle for solution delivery projects.
3. **Continuous Delivery: Agile.** This is a Scrum-based life cycle for long-standing teams.
4. **Continuous Delivery: Lean.** This is a Kanban-based life cycle for long-standing teams.
5. **Exploratory.** This is a Lean Startup-based life cycle for running experiments with potential customers to discover what they actually want. This life cycle supports a design thinking approach, as described in Chapter 2.
6. **Program.** This is a life cycle for a team of agile or lean teams.

Chapter 6 describes the six DAD life cycles in detail, as well as the traditional life cycle, and provides advice for when to choose each one.

Consumable Solutions Over Working Software

The Agile Manifesto suggests that we measure progress based upon "working software." But what if the customer doesn't want to use it? What if they don't like using it? From a design thinking point of view, it is clear that "working" isn't sufficient. Instead, we need to deliver something that is consumable:

- **It works.** What we produce must be functional and provide the outcomes that our stakeholders expect.
- **It's usable.** Our solution should work well, with a well-designed user experience (UX).
- **It's desirable.** People should want to work with our solution, and better yet feel a need to work with it, and where appropriate to pay us for it. As the first principle of Disciplined Agile recommends, our solution should delight our customers, not just satisfy them.

Additionally, what we produce isn't just software, but instead is a full-fledged solution that may include improvements to:

- **Software.** Software is an important part, but just a part, of our overall solution.
- **Hardware.** Our solutions run on hardware, and sometimes we need to evolve or improve that hardware.
- **Business processes.** We often improve the business processes around the

usage of the system that we produce.

- **Organizational structure.** Sometimes the organization structure of the end users of our systems evolves to reflect changes in the functionality supported by it.
- **Supporting documentation.** Deliverable documentation, such as technical overviews and user manuals/help, is often a key aspect of our solutions.

DAD Terminology

Table 3.1 maps common DAD terms to the equivalent terms in other approaches. There are several important observations that we'd like to make about the terminology:

1. **There is no standard agile terminology.** There isn't an ISO industry standard for agile and, even if there was, it very likely would be ignored by agile practitioners.
2. **Scrum terminology is questionable at best.** When Scrum was first developed in the 1990s, its creators purposefully decided to choose unusual terminology, some adopted from the game of rugby, to indicate to people that it was different. That's perfectly fine, but given that DA is a hybrid we cannot limit it to apply arbitrary terms.
3. **Terms are important.** We believe terms should be clear. You need to explain what a scrum meeting is, and that it isn't a status meeting, whereas it's pretty clear what a coordination meeting is. Nobody sprints through a marathon.
4. **Choose whatever terms you like.** Having said all this, DAD doesn't prescribe terminology, so if you want to use terms like sprint, scrum meeting, or scrum master, then go ahead.
5. **Some mappings are tenuous.** An important thing to point out is that the terms don't map perfectly. For example, we know that there are differences between team leads, scrum masters, and project managers, but those differences aren't pertinent for this discussion.

Context Counts: DAD Provides the Foundation for Scaling Agile Tactically

Disciplined Agile (DA) distinguishes between two types of "agility at scale:"

1. **Tactical agility at scale.** This is the application of agile and lean strategies on

deliver the solution. While the product owner may not be able to answer all questions, it is their responsibility to track down the answer in a timely manner so that the team can stay focused on their tasks.

Each DAD team, or subteam in the case of large programs organized as a team of teams, has a single product owner. A secondary goal for a product owner is to represent the work of the agile team to the stakeholder community. This includes arranging demonstrations of the solution as it evolves and communicating team status to key stakeholders.

As a stakeholder proxy, the product owner:

- Is the "go-to" person for domain information;
- Provides information and makes decisions in a timely manner;
- Prioritizes all work for the team, including but not limited to requirements (perhaps captured as user stories), defects to be fixed, technical debt to be paid down, and more (The product owner takes both stakeholder and team needs into account when doing so.);
- Continually reprioritizes and adjusts scope based on evolving stakeholder needs;
- Is an active participant in modeling and acceptance testing;
- Helps the team gain access to expert stakeholders;
- Accepts the work of the team as either done or not done;
- Facilitates requirements modeling sessions, including requirements envisioning and look-ahead modeling;
- Educates the team in the business domain; and
- Is the gateway to funding.

When representing the agile team to the stakeholder community, the product owner:

- Is the public face of the team to stakeholders;
- Demos the solution to key stakeholders, which may include coaching team members to run the demo;
- Announces releases;
- Monitors and communicates team status to interested stakeholders, which may include educating stakeholders on how to access and understand the team's