CHAPTER 1

Agile Principles and Mindset

Domain I Summary

This chapter discusses domain I in the exam content outline, which is 16 percent of the exam, or about 19 exam questions. This domain focuses on the agile mindset, its fundamental values and principles, the agile methodologies, and agile leadership.

Key Topics

- » Agile frameworks and terminology
- » Agile Manifesto
  - 4 values
  - 12 principles
- » Agile methods and approaches
- » Agile process overview
- » Kanban
  - Five principles
  - Pull system
  - WIP limits
- » Leadership practices and principles
  - Management vs. leadership
  - Servant leadership (4 duties)
- » Lean
  - Core concepts
  - 7 wastes
- » Scrum
  - Activities
  - Artifacts
  - Team roles
- » XP
  - Core practices
  - Core values
  - Team roles

Tasks

1. Advocate for agile principles and values to create a shared mindset.
2. Ensure a common understanding of agile.
3. Support change through educating and influencing.
4. Practice transparency in order to enhance trust.
5. Create a safe environment for experimentation.
6. Experiment with new techniques and processes.
7. Share knowledge through collaboration.
8. Encourage emergent leadership via a safe environment.
9. Practice servant leadership.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

In this chapter, we'll define some key concepts that provide a foundation for the material presented in the rest of this book. In essence, this chapter sets the stage for understanding agile for the PMI-ACP exam. It summarizes the agile mindset and methodologies, outlines the major agile methodologies, and explores the nature of agile leadership and how it differs from traditional management.

Even when an exam question deals with a specific tool or technique, an understanding of the broader agile framework will often be required to answer it correctly. So if you are faced with a question that appears to have two or more reasonable answers, think about the underlying values and principles of agile explained in this chapter to help you make the right selection.

Why Use Agile?

Let's start with a basic question. Why do we need another approach for running projects, anyway? Isn't a method like the one extensively documented in the *PMBOK® Guide* sufficient?

The answer is simple: different types of projects require different methods. In our everyday lives, we continually customize our approach depending on the situation, often in small and subtle ways. For example, we choose what information to communicate and how to present it based on our audience. We don't approach every issue we face in the exact same way, like a robot; instead, we adjust our methods to be most effective for each situation. This same concept applies to projects—and some projects, especially knowledge work projects in a fast-moving, complex environment, call for an agile approach.

Knowledge Work Projects Are Different

First let's discuss a little history that isn't tested on the exam but will help set the scene. Initially humans wandered the earth as hunter-gatherers. When people started planting crops and herding animals, it changed society and work. This was the Agricultural Revolution. As a result, people wandered less, and they lived and worked in one place.

The next big transformation came with the development of machines and factories, when people left their farms and villages to move into cities. This was the Industrial Revolution, which eventually led to the development of many classic project management tools and concepts, including Gantt charts, functional decomposition, and localized labor. In turn, these developments led to the creation of more advanced project management tools, such as the work breakdown structure (WBS).

The latest stage—which we are in now—is known as the Information Revolution. This revolution is focused on information and collaboration, rather than manufacturing. It places value on the ownership of knowledge and the ability to use that knowledge to create or improve goods and services.

The Information Revolution relies on knowledge workers. These are people with subject matter expertise who communicate their knowledge and take part in analysis or development efforts. Knowledge workers are not only found in the IT industry; they are also engineers, teachers, scientists, lawyers, doctors, writers, and many others employed today. In fact, knowledge workers have become the largest segment of the North American workforce.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

So what makes knowledge work projects different from manufacturing projects? To find out, let's compare the key characteristics of industrial work and knowledge work:[1]

Characteristics of Industrial Work	Characteristics of Knowledge Work
Work is visible	Work is invisible
Work is stable	Work is changing
Emphasis is on running things	Emphasis is on changing things
More structure with fewer decisions	Less structure with more decisions
Focus on the right answers	Focus on the right questions
Define the task	Understand the task
Command and control	Give autonomy
Strict standards	Continuous innovation
Focus on quantity	Focus on quality
Measure performance to strict standards	Continuously learn and teach
Minimize cost of workers for a task	Treat workers as assets, not as costs

EXERCISE

Review the items in each column of the following table and place a check mark next to any that describe your job. When you are finished, look at the pattern of the check marks. Are they mostly on the left (industrial work) or mostly on the right (knowledge work)?

Industrial Work	Knowledge Work
☐ Work is visible	☐ Work is invisible
☐ Work is stable	☐ Work is changing
☐ Emphasis is on running things	☐ Emphasis is on changing things
☐ More structure with fewer decisions	☐ Less structure with more decisions
☐ Focus on the right answers	☐ Focus on the right questions
☐ Define the task	☐ Understand the task
☐ Command and control	☐ Give autonomy
☐ Strict standards	☐ Continuous innovation
☐ Focus on quantity	☐ Focus on quality
☐ Measure performance to strict standards	☐ Continuously learn and teach
☐ Minimize cost of workers for a task	☐ Treat workers as assets, not as costs

When knowledge work projects became more common, people found that the communication and collaboration involved in these projects made the work more uncertain and less definable than industrial work. As people tried to apply industrial work techniques to knowledge work projects, frustration—and project failures—increased. Agile methods were developed in response to this problem. Agile pioneers collected the most effective techniques for knowledge work and adapted them for use on projects, experimenting to see what worked best. This new initiative began in the software development field, but is now used in all kinds of knowledge work projects.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

This development of agile methods took place over many years and was done by different people. As a result, agile has multiple methodologies that use different terminology. For example, Scrum calls its timeboxed development efforts "sprints," while XP calls them "iterations." This chapter will establish a framework for understanding the agile mindset and explain some fundamental concepts you need to understand to do well on the PMI-ACP exam.

Defined versus Empirical Processes

Another way of looking at the difference between knowledge work and industrial work is to examine the different kinds of processes they use. Industrial work typically uses a defined process, while knowledge work relies on an empirical process.

In a *defined* process, as the name implies, we can define the constituent steps in advance. If we know the optimum way to tie our shoelaces, then we can follow the same process each time. This is typically the most efficient way to proceed for a well-understood project in an unchanging environment, such as construction projects that use well-understood materials and building approaches. In fact, most industrial projects can be planned and managed by using a defined approach.

Other processes are not as well defined. When faced with a new or uncertain process, such as building an underwater home for the first time or using carbon nanotubes instead of steel, there will be many unknowns and uncertainties involved in the risks and solutions required for the new environment or materials.

When faced with such uncertainty, a process of trial and experiment is required to determine what works, surface issues, and incrementally build on small successes. The resulting process will be iterative and incremental, with frequent reviews and adaptation. The result is an *empirical* process. This approach is required for projects where the execution stage is characterized by uncertainty and risks—in other words, projects that would benefit from the agile approach.

K&S The Agile Mindset

Although there are a lot of tools, practices, and concepts that you'll need to know for the PMI-ACP exam, the underlying goal of the exam questions isn't really to test specific knowledge—instead, the exam wants to see if you have a comprehensive grasp of what it means to "be agile" (not just "do agile"). "Being agile" isn't simply a matter of using a certain set of tools or practices, or following a specific methodology. Agility really involves adopting a new mindset—way of thinking—that is based on agile values and principles.

The most important statement of these values and principles is the Agile Manifesto, which we'll examine in detail in the next section of this chapter. But first, let's look at some of the general characteristics of the agile mindset. We'll start with the "Declaration of Interdependence" (DOI) written in 2005 by the cofounders of the Agile Project Leadership Network (now the Agile Leadership Network). This document outlines six precepts:[2]

1. We **increase return on investment** by making continuous flow of value our focus.
2. We **deliver reliable results** by engaging customers in frequent interactions and shared ownership.
3. We **expect uncertainty** and manage for it through iterations, anticipation, and adaptation.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

4. We **unleash creativity and innovation** by recognizing that individuals are the ultimate source of value, and creating an environment where they can make a difference.
5. We **boost performance** through group accountability for results and shared responsibility for team effectiveness.
6. We **improve effectiveness and reliability** through situationally specific strategies, processes, and practices.

Since the DOI is aimed at leaders—it focuses on the management side of agile projects—these six principles can serve as a kind of topline summary to introduce you to the agile mindset, although the DOI won't be tested on the exam.

Read through these six statements again and see if you understand them. Do they make sense to you? If this mindset sounds very different from the working environment you are used to, then you will need to carefully study this chapter in order to do well on the exam. The exam questions require you to have a firm grasp of "agile thinking"; they can't be answered simply by memorizing information.

Exam tip

In this section ("The Agile Mindset") we are presenting various perspectives on, or aspects of, the agile way of thinking. With the exception of the inverted agile triangle (discussed below) this information won't be directly tested on the exam. Here, our goal is simply to provide a basic answer to the question "What is agile?"

Moving on, here's another way of summing up the core principles of agile:

» Welcoming change
» Working in small value-added increments
» Using build and feedback loops
» Learning through discovery
» Value-driven development
» Failing fast with learning
» Continuous delivery
» Continuous improvement

As you think about this list, remember that agile isn't just a matter of adopting practices designed to achieve these outcomes. We actually have to take the agile mindset to heart and use it to guide our approach. Applying agile values and principles to how we use agile methods changes not only our approach, but also the effectiveness of the practices. This is the difference between "being agile" and "doing agile," as illustrated below.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

Figure 1.1: Being Agile versus Doing Agile

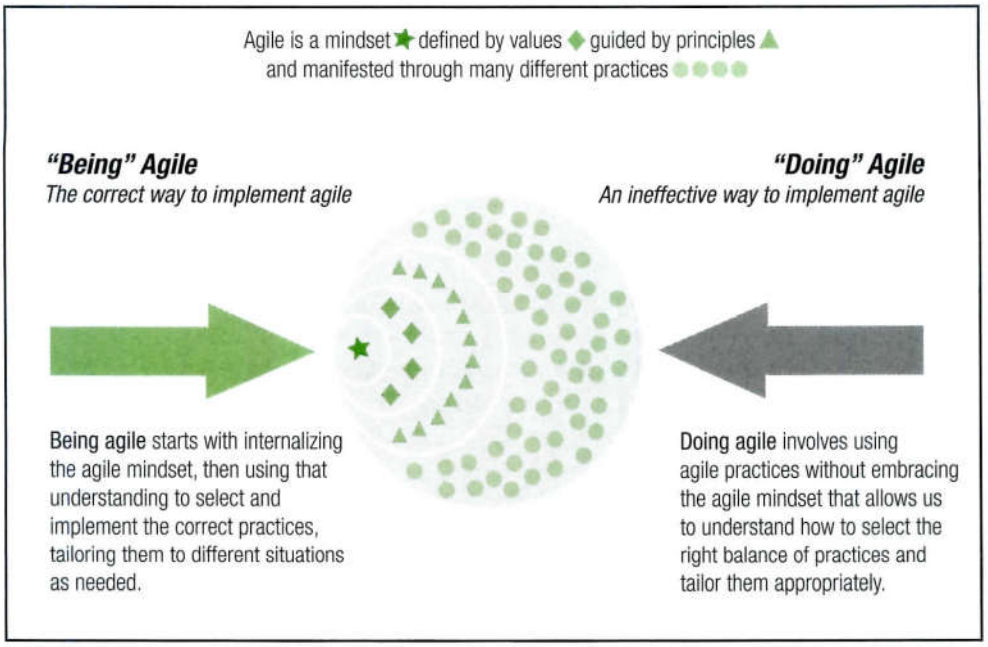


Image copyright © Ahmed Sidky, Santeon Group, www.santeon.com

Let's break down this diagram.

» The correct way to adopt agile is shown by the green arrow on the left. Here we start by internalizing the agile mindset (welcoming change, small increments, etc.), and then we use those principles to guide our selection and implementation of agile practices. We start with a good understanding of *why* we are using the practices, which in turn helps us understand *how* to use them most effectively.

» The gray arrow on the right represents a team that decides to adopt agile practices (such as daily stand-up meetings and short iterations), without taking the time to understand what these practices are designed to accomplish. Here, we jump directly into the *how* of agile without first understanding the *why*. This is a common problem in agile adoption.

Based on this diagram, people sometimes use the term "left-to-right adoption" as a shorthand way of saying "teach agile values first."

Personal, Team, and Organizational Agility

Within an organization, people will develop an understanding of the agile mindset at different rates. To some, agile values will be intuitively easy to grasp—they seem to describe familiar beliefs and behavior patterns. To others, these values will have to be consciously learned and then actively practiced before they can be understood and accepted. Regardless of how this understanding develops, the more people there are in an organization who embrace and act upon agile principles, the more effective agile practices will be.

» If just one member of an organization adopts an agile mindset, it can help that person become more effective. However, they will feel continually frustrated that others in the organization don't seem to realize what is important, or are focused on the wrong goals and metrics.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

» If one team in the organization adopts agile principles and practices, it can help the team members become more effective at delivering their project work. However, they will feel inhibited or misunderstood by other groups or systems in the organization, such as the project management office (PMO) or functional silos.

» If the entire organization adopts the agile way of thinking, then everyone will be working together to improve agility and the delivery of value. By adopting common goals and values, such as continuous improvement and welcoming change, everyone's effectiveness will be enhanced.

Although organizational agility is the ideal goal, today most organizations are not there yet. So how can we help our organizations get there? The way organizations change is through influence exerted by individuals. The diagram below depicts the steps involved in this process as the layers of an onion skin.

Figure 1.2: Creating Organizational Change

Let's look more closely at each of these layers:

Think

First, we need *think*—this means individually learning and internalizing agile principles. Only once we fully understand the agile mindset can we move on to the next step.

Do

Doing is the practice of agile. For example, this might involve visualizing work items, using short iterations, or building in feedback and improvement steps. Once we understand and can practice these steps ourselves, we can move on to the next step, where we begin influencing others.

Encourage Others

This final step involves *encouraging others* to become agile. Although this may appear to be exponentially harder to achieve than the first two steps, it is also the most worthwhile when accomplished. That's because persuading others to adopt the agile mindset and practices will magnify agile learning and effectiveness across the entire organization. Also, the more people in your circle of work you can successfully educate about the merits of agile, the more allies you will have in advocating the cause. The end result of this process can be a complete transformation of the organization based on agile principles.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

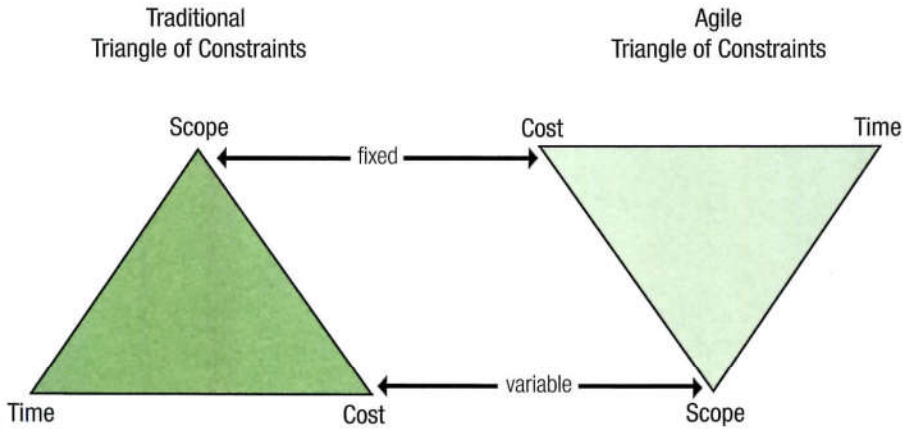
Exam tip

Like the "Think-Do-Encourage others" model described above, you'll see many process steps and bulleted lists in this book. Don't approach this material by trying to cram it, as if you were studying for a college exam. That won't work because the PMI-ACP exam doesn't test information—most of the questions will be situational. So your study goals should be, first, to fully absorb the agile mindset and then, second, to become generally familiar with agile tools and practices. So instead of memorizing details, focus on the big picture: "What is important about this list or process for agile?" "How does each item listed fit into the agile way of doing things?" This is what you need to understand for the exam.

The Agile Triangle

Another key difference between the agile mindset and traditional project management is the agile or "inverted" triangle of constraints. This triangle, shown below, was introduced in the first edition of the DSDM Manual, published in 1994.

Figure 1.3: Inverted Triangle Model



This reversal of the traditional triangle means that agile teams allow scope to vary within the fixed parameters of cost and time. In other words, we aim to deliver the most value we can by X date within X budget. Although we'll begin with a high-level vision of the end product, we can't define up front how much we'll be able to get done—that will emerge as we get closer to the target date.

You might hear criticism that this approach doesn't make sense, because it won't work for tangible, industrial work projects. For example, "If I'm paying to have my bathroom remodeled, I want the whole thing to be completed, not just most of it!" However, remodeling a bathroom should be a defined, repeatable process with little R&D or uncertainty. That isn't the kind of project that calls for an agile approach or an inverted triangle of constraints.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

Knowledge work projects, on the other hand, are characterized by experimentation and uncertainty—and the end product can be refined forever. This is why we need to determine acceptable operating boundaries, which usually take the form of cost and time constraints. For example, let's say we are developing training materials for a course on short story writing. For a project such as this, we could continue adding and tweaking our lesson material and exercises indefinitely. Instead we need to get to an acceptable performance point, and then have the discipline to stop.

K&S The Agile Manifesto

The name of domain I ("Agile Mindset and Principles") highlights not only the agile mindset, but also the values and principles it is based on. For the exam, you'll need a thorough understanding of the most important statement of agile values and principles— a document called the Agile Manifesto.

The Agile Manifesto was created during a meeting in February 2001 that brought together a number of software and methodology experts who were in the forefront of the emerging agile methods. The people in attendance were:

- Kent Beck
- Mike Beedle
- Arie van Bennekum
- Alistair Cockburn
- Ward Cunningham
- Martin Fowler
- James Grenning
- Jim Highsmith
- Andrew Hunt
- Ron Jeffries
- Jon Kern
- Brian Marick
- Robert C. Martin
- Steve Mellor
- Ken Schwaber
- Jeff Sutherland
- Dave Thomas

Exam tip

The PMI-ACP exam won't ask about the origin of the Agile Manifesto or its creators. We are including this information to set the context for the discussion that follows. The exam will focus on your ability to apply the values and principles set forth in this document in situational questions. It won't ask "What is the fourth value?" or "How is the second principle phrased?"

The Four Values

The Agile Manifesto includes a statement of four values and twelve guiding principles. We'll examine the values first. This section of the document reads as follows:[3]

> ***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:*
>
> ***Individuals and interactions*** *over processes and tools*
> ***Working software*** *over comprehensive documentation*
> ***Customer collaboration*** *over contract negotiation*
> ***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.*

While the Agile Manifesto is simple in structure and sparse in words, there is a lot of good stuff in these four values. Understanding the ideas conveyed in these values is important not just for the exam, but also for the application of an agile approach on any kind of knowledge work project.

Exam tip

The Agile Manifesto was written by software development experts, so it uses terms from that field to express its ideas; however, those ideas are applicable to any kind of knowledge work project. So although you are likely to see some IT-specific terms on the exam, you'll need to be able to recognize these ideas in other contexts as well. As you read about the Manifesto, look beyond the specific terms and think about how these concepts apply to other types of knowledge work. For example, we might rephrase the second value as "Working *systems* over comprehensive documentation," substituting "systems" for "software."

The format of the four values—A *over* B ("Individuals and interactions *over* processes and tools")—addresses intention, focus, and effort. This isn't as black and white as just saying, "Do A *instead* of B." Instead, it acknowledges that both A and B will be components of our projects, but that we should apply more of our focus, emphasis, and intention to A than to B.

The Agile Manifesto isn't a set of rules telling us to do one thing instead of another. It is both more subtle and more powerful—it guides us to consider projects from a value-based perspective. Yes, we will need processes, tools, documentation, and plans on our projects. Yet while dealing with these assets, we should remember that our focus must be on the people engaged, the product we are building, cooperation, and flexibility. Agility is the capacity to execute projects while focusing our efforts on the items on the left side of these value statements, rather than those on the right.

With this in mind, let's look at these four statements in more detail.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

Value 1: Individuals and Interactions Over Processes and Tools

The message here is that while processes and tools will likely be necessary on our projects, we should try to focus the team's attention on the individuals and interactions involved. This is because projects are undertaken by people, not tools, and problems get solved by people, not processes. Likewise, projects are accepted by people, scope is debated by people, and the definition of a successfully "done" project is negotiated by people. Focusing early on developing the individuals involved in the project and emphasizing productive and effective interactions help set up a project for success.

This is not to say that processes and tools cannot help in successfully completing a project. They are certainly important assets, and for those of us who have an engineering background, we may naturally tend toward the logic and predictability of processes and tools. Yet projects are ultimately about people, so to be successful, we need to spend the majority of our time in what may be the less comfortable, messy, and unpredictable world of people. If you tend toward processes rather than people, the first value of "individuals and interactions over processes and tools" is a great reminder of where to focus your time, energy, and passion.

Value 2: Working Software Over Comprehensive Documentation

This value speaks to the need to deliver. As mentioned in the exam tip on the previous page, it can be rephrased more broadly as "Working *systems* over comprehensive documentation," where "system" refers to the product or service that the project is delivering. This value reminds us to focus on the purpose or business value we're trying to deliver, rather than paperwork. (Some highly regulated industries, such as medical devices, require a lot of documentation—that's just part of the work that needs to be done. This value refers to the kind of documentation that isn't included in getting the work done properly.)

The agile approach to documentation is "just enough, just in time—and sometimes, just because." This phrase is shorthand to remind us of three important concepts:

» Agile documentation should be "barely sufficient"—just enough to cover our needs. This keeps most of our efforts focused on the emerging system.

» Agile documentation is done "just in time"—also known as the "last responsible moment"—so we don't have to spend extra time to keep it updated as our requirements and designs change.

» Finally, "just because" reminds us that sometimes when documentation is required or requested, it's easier and preferable to just produce it than to face the consequences of not doing so.

That last point doesn't mean we just pander to unnecessary documentation requests; for example, the lean methodology includes the powerful concept of asking "Why?" five times to find out what we really need. However, it pays to be smart, too—given the limited time and effort available on a project, we need to decide where to best focus our energy, and which battles to pursue. So although we value working on the product over documentation, that doesn't mean we abandon the opposing value.

Software projects are typically initiated with the goal of creating valuable, high-quality software, yet they often get caught up on interim deliverables such as extensive documentation that doesn't support the ultimate goal of working software. Software without documentation is certainly problematic and hampers support and maintenance. But comprehensive documentation without software has no value in most organizations.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

Many software developers are detail-oriented and process-driven; although these characteristics are often highly beneficial, they can also mean the developers' focus can easily be distracted from the real reason they are undertaking software projects—to write valuable software. So this emphasis on valuing working software over comprehensive documentation acts as a necessary and useful reminder of why these projects are commissioned in the first place—to build something useful. Documentation by itself, or at the expense of working software, is not useful.

Value 3: Customer Collaboration Over Contract Negotiation

This value reminds us to be flexible and accommodating, rather than fixed and uncooperative. It is similar to the difference between "being right" and "doing the right thing." We could build the product exactly as originally specified, but if the customer's preferences or priorities change, it would be better to be flexible and work toward the new goal, as opposed to the goal that was originally stated.

It is notoriously difficult to define an up-front, unchanging view of what should be built. This challenge stems from the dynamic nature of knowledge work products, especially software systems; software is intangible and difficult to reference, companies rarely build the same systems twice, business needs change quickly, and technology changes rapidly. Rather than subject the client to a change management process that is really more of a change suppression process, we should recognize at the start that things are going to change, and we'll need to work with the customer throughout the project to reach a shared definition of "done." This requires a more trusting relationship and more flexible contract models than we often see on projects, but—like the previous value—it moves the emphasis from nonvalue-adding activities (such as arguing about scope) to productive work.

Value 4: Responding to Change Over Following a Plan

In knowledge work projects, we know that our initial plans are inadequate, since they are based on insufficient information about what it will take to complete the project. So instead of investing effort in trying to bring the project back in line with our original plan, we want to spend more of our effort and energy responding to the changes that will inevitably arise. Now, this doesn't mean we should abandon planning and just react to changes. We still need to plan, but we also need to acknowledge that our initial plans were made when we knew least about the project (at the beginning) and will need to be updated as the work progresses.

The importance of responding to change over following a plan is particularly true for software projects, where high rates of change are common. Again, instead of suppressing changes and spending a lot of time managing and tracking a largely static plan, we need to acknowledge that things will change. Agile projects have highly visible queues of work and plans in the form of backlogs and task boards. The intent of this value is to broaden the number of people who can be readily engaged in the planning process by adjusting the plans and discussing the impact of changes.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

EXERCISE: MATCH THE AGILE MANIFESTO VALUES

Note: As you do the exercises in this book, you may want to have a sheet of paper handy to cover up the answers to the exercises, since they are just after the questions.

Draw arrows to match the start of the Agile Manifesto values on the left to their correct endings on the right.

Start		End
Working software	*Over*	Processes and tools
Responding to change	*Over*	Comprehensive documentation
Individuals and interactions	*Over*	Contract negotiation
Customer collaboration	*Over*	Following a plan

ANSWER

Start		End
Working software	*Over*	Processes and tools
Responding to change	*Over*	Comprehensive documentation
Individuals and interactions	*Over*	Contract negotiation
Customer collaboration	*Over*	Following a plan

The Twelve Principles

In addition to the four agile values, the authors of the Manifesto created twelve guiding principles for agile methods. This part of the Manifesto reads as follows:[4]

> ***Principles behind the Agile Manifesto***
>
> *We follow these principles:*
>
> *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.*
>
> *Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.*
>
> *Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.*
>
> *Business people and developers must work together daily throughout the project.*
>
> *Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.*

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity—the art of maximizing the amount of work not done—is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Exam tip

Since these values and principles are fundamental to understanding agile, think of them when you are faced with two seemingly correct options on the exam. In such instances, look for the answer that best matches the Agile Manifesto values and principles.

Let's take a closer look at each of the Agile Manifesto's twelve principles. Again, although the principles may use software development terms, as you read about them, think about how these concepts apply to other types of knowledge work projects.

Principle 1: Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

This principle includes three main points. The first is to satisfy the *customer*. If we produce perfect plans and documentation, but only delight the project management office (PMO) or the quality assurance (QA) group, we have failed; our focus should be on the customer.

The second point is *early and continuous delivery*. We must structure the project and the development team to deliver value early and frequently. This can be a struggle if people are reluctant to share incomplete work, and it takes courage and support for everyone to become comfortable with this idea. However, we must achieve this point if we are to learn of problems while we still have time to fix them. It is better to get something wrong up front and have time to correct it than to discover the issue much later when so much more has been built on top of a faulty foundation.

The final point is that what we are delivering is *valuable software* (i.e., systems), not completed work products, WBS items, documentation, or plans. We need to stay focused on the end goal. For software projects, this is the software; for other types of projects, the end goal will be the product or service that the project was undertaken to deliver or enhance.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

Principle 2: Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Changes can be great for a project—for example, if they allow us to deliver a late-breaking, high-priority feature. Yet in non-agile projects, changes are often seen in a negative light; they may be considered "scope creep" or blamed for the project deviating from the plan. Many non-agile projects have such rigorous change control procedures that only the highest priorities make it through. On such projects, much of the time and effort is spent logging and managing change requests.

This kind of rigorous change management is problematic for any project in a highly changeable environment, such as software development—so agile accepts that changes will occur. In fact, the XP methodology advocates that we "embrace change."

Instead of creating a high-overhead mechanism for suppressing or processing changes, agile methods use a lightweight, high-visibility approach—for example, continuously updating and prioritizing changes into the backlog of work to be done. Agile's well-understood, high-visibility methods for handling changes keep the project adaptive and flexible as long as possible. By welcoming the changes that will inevitably happen and setting up an efficient way to deal with them, we can spend more time developing the end product.

Principle 3: Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Despite our best intentions and knowledge that early feedback is valuable, it's human nature to want our work to be as complete as possible before sharing it. However, we are doing ourselves a disservice by holding on to our work for too long. It's better to get feedback early and often, to avoid going too far down the wrong track.

This principle emphasizes the importance of releasing work to a test environment and getting feedback. Frequent testing and feedback are so important that software developers use continuous integration tools to provide feedback about any code they have written that breaks the build. Agile teams need feedback on what they have created thus far to see if they can proceed, or if a change of course is needed.

Delivering within a short timeframe also has the benefit of keeping the product owner engaged and keeping dialogue about the project going. With frequent deliveries, we will regularly have results to show the customer and opportunities to get feedback. Often at these demos, we learn of new requirements or changes in business priorities that are valuable planning inputs.

Principle 4: Business people and developers must work together daily throughout the project.

The frequent demos mentioned above are one example of how the business representatives and developers work together throughout the project. Daily face-to-face engagement with the customer is one of the most difficult principles to ensure from a practical standpoint, but it is really worth pushing for. Written documents, e-mails, and even telephone calls are less efficient ways of transferring information than face-to-face interactions.

By working with business representatives daily, we can learn about the business in a way that is far beyond what a collection of requirements-gathering meetings can ever achieve. As a result, we are better able to suggest solutions and alternatives to business requests. The business representatives also learn what types of solutions are expensive or slow to develop, and what features are cheap. They can then begin to fine-tune their requests in response.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

When it isn't possible to have daily interactions between the business representatives and the development team, agile methods try to get the two groups working together regularly in some way, perhaps every two days or whatever type of frequent involvement will work. (Some teams use a "proxy customer," in which an experienced business analyst [BA] who is familiar with the business interests serves as a substitute, but this isn't an ideal option.)

Principle 5: Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

As we'll see in chapter 4, software estimation data gathered by the COCOMO II® model has found that having the best people is significantly more important for a project than having the best processes and tools—by a factor of 10 to 1! So making sure we have smart and motivated people on the team is likely to make a big difference in whether our project is delivered successfully and efficiently.

While we may not always be able to pick our dream team, we can motivate and empower the team members we do have. Since the development team is such an important factor, agile methods promote empowered teams. People work better when they are given the autonomy to organize and plan their own work. Agile methods advocate freeing the team from the micromanagement of completing tasks on a Gantt chart. Instead, the emphasis is on craftsmanship, peer collaboration, and teamwork, which result in higher rates of productivity.

Knowledge work projects involve team members who have unique areas of expertise. Such people do their best work when they are allowed to make many of the day-to-day decisions and local planning for the project. For leaders, this doesn't mean abdicating involvement or abandoning the team to fend for itseslf; instead, we recognize that our team members are experts in what they do, and we provide the support they need to ensure they are successful.

Principle 6: The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Written documents are great for creating a lasting record of events and decisions, but they are slow and costly to produce. In contrast, face-to-face communication allows us to quickly transfer a lot of information in a richer way that includes emotions and body language.

In a face-to-face conversation, questions can be immediately answered, instead of "parked" with the hope that there will be a follow-up explanation, or the answer will become clear later. For example, if instead of reading this book, you were talking to me directly, we could quickly skip all the stuff you already know and focus on the areas you want to learn more about. Written documents have to assume a lower starting point so as not to confuse their broader audience.

Of course, this recommendation for face-to-face conversations can't be applied to all project communications, but agile teams aim to follow it whenever possible. This is one example of how agile methods need to be customized or scaled for each project. As team sizes grow, it becomes harder to rely on face-to-face communications, and we have to introduce an appropriate level of written documentation.

Principle 7: Working software is the primary measure of progress.

By adopting "working software" (or "working systems") as our primary measure of progress, we shift our focus to working results rather than documentation and design. In agile, we assess progress based on the emerging product or service we are creating. Questions like "How much of the solution is done and accepted?" are preferred over "Is the design complete?" since we want to focus on usability and utility rather than conceptual progress.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

Expressions like "What gets measured gets done" and "You get what you measure" apply here, since measurement reinforces activity. If a feature can't be measured or tested—in other words, if it doesn't "work"—it isn't considered complete. This emphasis on a working product helps ensure that we get features accepted by the customer, rather than marking items as "completed development" when they haven't yet been accepted as "done."

This definition of progress as "working systems" creates a results-oriented view of the project. Interim deliverables and partially completed work will get no external recognition. So we want to focus instead on the primary goal of the project—a product that delivers value to the business.

Principle 8: Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Agile methods strive to maximize value over the long term. Some of the rapid application development (RAD) techniques that preceded agile promoted—or at least accepted—long, intense periods of prototyping prior to demos. The trouble with the RAD approach of working teams for long hours over an extended period of time is that people start to burn out and make mistakes. This is not a sustainable practice.

Instead of long, intense development periods, agile methods recognize the value of a sustainable pace that allows team members to maintain a work-life balance. A sustainable pace is not only better for the team; it benefits the organization as well. Long workdays lead to resignations, which means the organization loses talent and domain knowledge. Hiring and integrating new members into a team is a slow and expensive process.

In contrast, working at a pace that can be maintained indefinitely leads to a happier and more productive team. Happy teams also get along with the business representatives better than overworked teams. There is less tension, and work relationships improve. Some agile methods, like XP, previously recommended maintaining a regular 40-hour workweek as a guiding principle. Today most methods don't establish specific limits, but they do recommend paying close attention to the level of effort the team is putting forth to ensure a sustainable pace.

Principle 9: Continuous attention to technical excellence and good design enhances agility.

While we want the development team to work hard and deliver a lot of value, we also have to be mindful of keeping the design clean, efficient, and open to changes. Technical excellence and good design allow the development team to understand and update the design easily.

An agile team needs to balance its efforts to deliver high-value features with continuous attention to the design of the solutions. This balance allows the product to deliver long-term value without becoming difficult to maintain, change, or extend—cleaning and preventative maintenance are preferable to fixing problems. This helps the project run more smoothly and speeds up the team's progress.

In the software world, once the code base becomes tangled, the organization loses its ability to respond to changing needs. In other words, it loses its agility. So we need to give the development team enough time to undertake refactoring. Refactoring is the housekeeping, cleanup, and simplifications that need to be made to code to ensure it is stable and can be maintained over the long term.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

Principle 10: Simplicity—the art of maximizing the amount of work not done—is essential.

The most reliable features are those we don't build—since there is nothing that could go wrong with them. And, in the software world, up to 60 percent of features that are built are used either infrequently or never.[5] Because so many features that are built are never actually used, and because complex systems have an increased potential to be unreliable, agile methods focus on simplicity. This means boiling down our requirements to their essential elements only.

Complex projects take longer to complete, are exposed to a longer horizon of risk, and have more potential failure points and opportunities for cost overruns. Therefore, agile methods seek the "simplest thing that could possibly work" and recommend that this solution be built first. This approach is not intended to preclude further extension and elaboration of the product—instead, it simply says, "Let's get the plain-vanilla version built first." This approach not only mitigates risk but also helps boost sponsor confidence.

Principle 11: The best architectures, requirements, and designs emerge from self-organizing teams.

This principle may sound odd, depending on how literally you read it. Essentially it is saying that to get the best out of people, we have to let them self-organize. People like self-organizing; it allows them to find an approach that works best for their methods, their relationships, and their environment. They will thoroughly understand and support the approach, because they helped create it. As a result, they will produce better work.

However, you may well ask why this principle says that the best architectures, requirements, and designs come from the project team—rather than the organization's best architects, business analysts, and designers, who may not be on the team. In my experience, the answer to this question is that our architectures, requirements, and designs work best when they are implemented by those who originate them. Although external recommendations may have more technical merit on paper, if they are implemented differently than originally envisioned, or without conviction by the team, they will ultimately be less successful.

Self-organizing teams that have the autonomy to make local decisions have a higher level of ownership and pride in the architectures, requirements, and designs they create than in those that are forced on them or "suggested" by external sources. Ideas created by the team have already gone through the team vetting process for alignment and approval—so they don't need to be "sold" to the team. In contrast, ideas that come from outside sources need to be sold to the team for the implementation to be successful, and this is sometimes a challenging task.

Another factor that supports this principle is that the members of a self-organizing project team are closest to the technical details of the project. As a result, they are best able to spot implementation issues, along with opportunities for improvements. So instead of trying to educate external people about the evolving structure of the project, agile methods leverage the capacity of the team to best diagnose and improve the project's architectures, requirements, and designs. After all, the team members are the most informed about the project and have the most vested in it.

Principle 12: At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Gathering lessons learned at the end of a project is, frankly, too little, too late. Instead, we need to gather our lessons learned while they are still applicable and actionable. This means we need to gather them *during* the project and—most importantly—make sure we do something about what we've learned to adjust how we complete the remainder of the project.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

Agile methods employ frequent lookbacks, called "retrospectives," to reflect on how things are working on the project and identify opportunities for improvements. These retrospectives are typically done at the end of each iteration, ensuring that the team has regular opportunities to review their process. One advantage of doing retrospectives so frequently is that we don't forget about problems and issues. Compare this to conducting a single lessons learned review at the end of a project, in which team members are asked to think back over a year or more to recall what went well and where they ran into issues.

Another disadvantage of only gathering lessons learned at the end of a project is that the lessons won't be really helpful to the organization until another project with a similar business or technical domain or team dynamics comes along. And at that point, it is easy to dismiss the lessons learned from an earlier project as not applicable to the current situation. On an agile project, we capture the lessons learned as the project progresses, so we can't pretend they aren't applicable. We know they are relevant, and we are motivated to tune and adjust our process accordingly.

EXERCISE: SEVERE SUMMARIES

To make sure you understand the intent behind each of the Agile Manifesto principles, shorten them to just five words or less that describe the essence of the idea. You can use these shortened descriptions as a memory aid to help you recall these concepts. The first two have been done for you as an example.

	Principle	Shortened Version
1	Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	*Satisfy customer with great systems*
2	Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	*Welcome change*
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.	

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

	Principle	Shortened Version
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.	

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

ANSWER

Your description may vary, depending on what part of the principle stands out most for you, but the following are possible abbreviations.

	Principle	Shortened Version
1	Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	*Satisfy customer with great systems*
2	Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	*Welcome change*
3	Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.	*Deliver frequently*
4	Business people and developers must work together daily throughout the project.	*Work with business*
5	Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.	*Motivate people*
6	The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	*Face-to-face communications*
7	Working software is the primary measure of progress.	*Measure systems done*
8	Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.	*Maintain sustainable pace*
9	Continuous attention to technical excellence and good design enhances agility.	*Maintain design*
10	Simplicity—the art of maximizing the amount of work not done—is essential.	*Keep it simple*
11	The best architectures, requirements, and designs emerge from self-organizing teams.	*Team creates architecture*
12	At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.	*Reflect and adjust*

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

K&S Agile Methodologies

There are over a dozen actively used agile methodologies. The most common approaches are Scrum, Extreme Programming (XP), lean product development, Kanban, Feature-Driven Development (FDD), Dynamic Systems Development Method (DSDM), and the Crystal family of methods.

In the discussion that follows, we'll describe each of these methodologies in the level of detail you'll need for the exam. The two most widely used agile methods are Scrum and XP, which are prominently featured on the PMI-ACP exam. We'll explore these two approaches in the most detail, covering their key concepts, teams, activities, and deliverables. The exam may also include multiple questions about the lean and Kanban approaches. Although these sections aren't as long, it's important to understand all the information provided about these methods. Finally, the exam might not have any questions about FDD, DSDM, or Crystal—or perhaps just one question about them. However, it's still helpful to have a topline understanding of these methods, since they might appear as incorrect answer options that you need to rule out.

We've said that agile is a mindset, not a specific set of practices, and many agile teams end up tailoring their agile methodology for their own circumstances. This is called process tailoring, and we'll discuss its pros and cons in more detail in chapter 7. At this point, just know that some of these methodologies work best when used "as is" (for teams new to agile), while others lend themselves quite easily to adjustment and adaptation.

For example, Kanban encourages modification and adaptation; Crystal offers a range of methodologies for different kinds of projects; and DSDM offers suitability filters for assessing how well an approach fits a project. On the other hand, both Scrum and XP consist of a balanced set of practices that works best when used "out of the box" as an integrated system, at least until the team has gained enough agile experience to understand how all the pieces fit together.

Exam tip

The different agile methods, and their inherent flexibility, presented a dilemma for PMI in creating the exam, since there is no single "right" way to do agile. When the PMI-ACP Steering Committee originally developed the exam content outline, we recognized the irony of creating a prescriptive exam outline for a scalable, adaptive process. However, we had to make some assumptions about the team size, composition, and project complexity that would be tested. In the end, we decided to focus on the practices that are most commonly used on small- to medium-sized projects. So if you are struggling with the context of an exam question, think of a project that has a development team of up to 12 people and is using discretionary or essential funds.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

In this chapter, we'll be describing each agile approach as a stand alone methodology, since that information will be tested on the exam. However, it's not uncommon for agile teams to use a customized hybrid approach that incorporates elements from more than one methodology. For example, a team might combine some of XP's team practices, such as refactoring, pair programming, and simple design, with Scrum's approach for planning sprints. We'll talk more about these "hybrid models" in chapter 7, providing some examples of how different approaches can be used together.

Exam tip

I've gotten feedback that some people who are steeped in the Scrum methodology are having difficulty passing the PMI-ACP exam. These students believe that "Scrum *is* agile"—and so by extension, agile is Scrum, and *only* Scrum—and since they already know Scrum, they don't study properly for the exam. For example, they assume that all agile teams have a product owner and work in timeboxed sprints followed by sprint review meetings. However, the PMI-ACP exam tests a much broader picture of agile—it also encompasses Kanban and lean methods that don't have a product owner, timeboxed sprints, or scrum meetings. The underlying principles of all the agile methodologies are still compatible. But if you're mostly familiar with Scrum, you'll need to think of agile more broadly and learn about other agile approaches to pass the exam.

Scrum

Scrum is a popular agile model that is lightweight and easy to understand—but like all agile methods, it is difficult to truly master. The methodology documented in the "Scrum framework" is a set of practices, roles, events, artifacts, and rules that are designed to guide the team in executing the project.

Scrum Pillars and Values

The theory behind Scrum is based on the three pillars of transparency, inspection, and adaptation. These principles guide all aspects of the Scrum methodology:

» **Transparency**: This involves giving visibility to those responsible for the outcome. An example of transparency would be creating a common definition of what "done" means, to ensure that all stakeholders are in agreement.

» **Inspection**: This involves doing timely checks of how well the project is progressing toward its goals, looking for problematic deviations or differences from the goals.

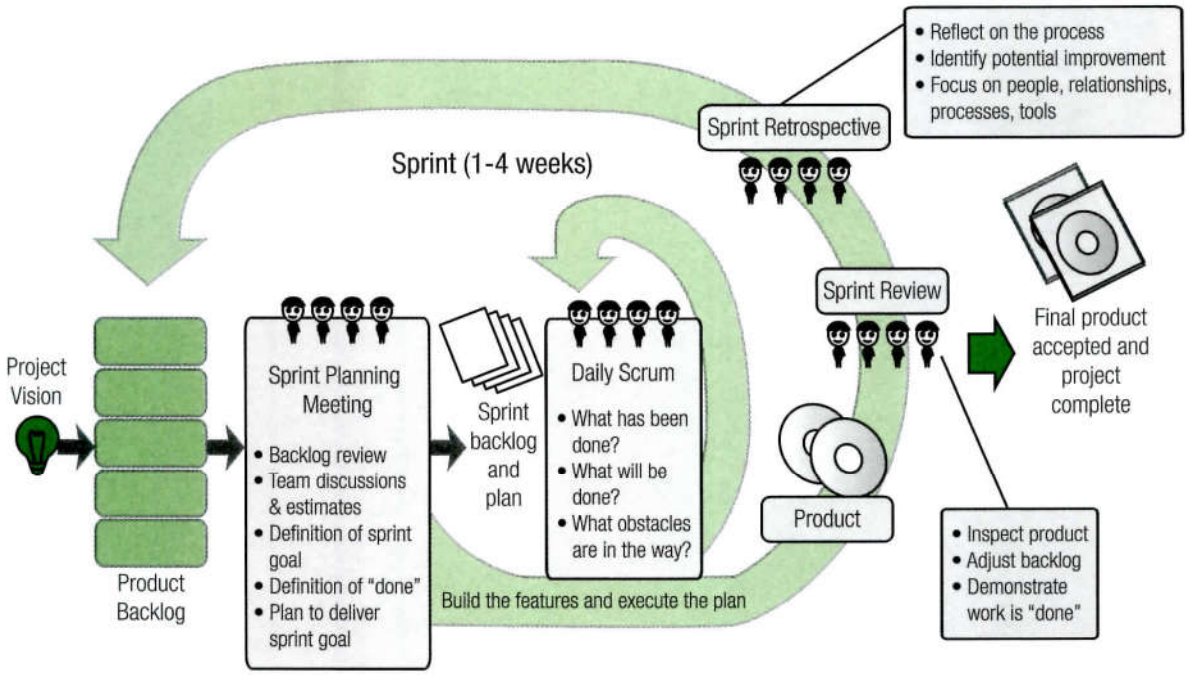
» **Adaptation**: This involves adjusting the team's process to minimize further issues if an inspection shows a problem or undesirable trend.

In addition to the three pillars, Scrum also recognizes five fundamental values—focus, courage, openness, commitment, and respect. These values reflect the concepts laid out in the Agile Manifesto, and they are also similar to the core values of XP (as we'll see when we get to that methodology).

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

The Scrum process is shown on the diagram below. As you continue reading about Scrum, refer back to this diagram to see how each component fits into the Scrum process.

Figure 1.4: Scrum Process



In this section, we'll discuss the key terms and components of Scrum, including the roles on a Scrum team, the activities that occur during a sprint, and the sprint deliverables or artifacts. We'll kick off this discussion by examining a key Scrum concept—sprints.

Sprints

A sprint is a timeboxed (time-limited) iteration of one month or less in which the team builds a potentially releasable product. (The Scrum term "sprint" is equivalent to the generic agile term "iteration.") Most Scrum sprints are one or two weeks long. Each sprint is like a mini-project. During a sprint, no changes are made that would affect the sprint goal. The scope might be clarified or renegotiated as new information becomes available, but if (for example) a change has to be made to the members of the development team, that wouldn't be done during a sprint.

A sprint can be cancelled before the timebox is over if the sprint goal becomes obsolete because of a change in business direction or technology conditions. However, only the product owner can cancel the sprint. When that happens, any incomplete product backlog items are re-estimated and returned to the product backlog.

The sequence of activities within each sprint consists of a sprint planning meeting, a development period that includes daily scrums, a sprint review meeting, and a sprint retrospective meeting. (We'll be explaining these terms shortly.)

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

Scrum Team Roles

Scrum teams are made up of the development team, the product owner, and the ScrumMaster. Let's look at each of these roles to see what they are responsible for on a Scrum project.

Development Team

The development team is the group of professionals who build the product increments in each sprint. The members of the development team are self-organizing—that is, they are empowered to manage their own work. Scrum teams are also cross-functional; each team member can fulfill more than one of the roles needed to complete the work (such as analysis, build, and test on a software development team).

Product Owner

The product owner is responsible for maximizing the value of the product by managing the product backlog, or list of work to be done. This includes ensuring that the work items in the backlog are up to date and accurately prioritized based on business value. The product owner is also responsible for making sure that the business and the team have a shared understanding of the project vision, the project goals, and the details of the work to be done, so that the team can plan and build the work items.

Although the product owner is ultimately responsible for keeping the backlog prioritized and updated, the ScrumMaster or development team also assists in this process by sharing information about estimates, dependencies, technical work items, and so on.

ScrumMaster

The ScrumMaster is responsible for ensuring that the Scrum methodology is understood and used effectively. This person is a servant leader to the development team, removing any impediments to their progress, facilitating their events (meetings), and coaching the team members. The ScrumMaster also assists the product owner with managing the backlog and communicating the project vision, project goals, and the details of the backlog items to the development team. Finally, the ScrumMaster serves the organization by facilitating its adoption of Scrum, not just on one project, but on a wider scale throughout the organization.

Scrum Activities (Events, Ceremonies)

The Scrum methodology defines five "activities," which are actually meetings that are focused on a specific purpose. These include product backlog refinement, sprint planning meetings, daily scrums, sprint reviews, and sprint retrospectives. The last four of these meetings are the team's planned opportunities for inspection and adaptation (two of the three pillars) in each sprint.

Exam tip

On the exam, you may see the terms "Scrum events" or "Scrum ceremonies." These are older terms for the Scrum activities (meetings).

Backlog Refinement

The backlog refinement meetings are where "grooming the backlog" is done. This basically means that everyone involved in the project gathers to discuss and update the items in the backlog. (If you aren't sure what this means, keep reading—we'll explain this activity more fully when we describe the product backlog.)

Sprint Planning Meeting

In the sprint planning meeting, everyone gathers to determine what will be delivered in the upcoming sprint and how that work will be achieved. The product owner presents the updated backlog items, and the group discusses them to ensure they have a shared understanding. The development team forecasts what can be delivered in the sprint, based on their estimates, projected capacity, and past performance. With this forecast in mind, they define the sprint goal. The development team then determines how that functionality will be built and how they will organize themselves to deliver the sprint goal. (Sprint planning—i.e., iteration planning—is covered in more detail in chapter 5.)

Daily Scrum

The daily scrum is a 15-minute timeboxed meeting that is held at the same time and place every day while the team is working toward the sprint goal. The ScrumMaster makes sure the meeting happens every day, and follows up on any identified obstacles. The daily scrum is primarily for the members of the development team, who use it to synchronize their work and report any issues they are facing. The scope of this meeting is strictly limited—each member of the team briefly answers three questions about what they are doing to meet the sprint goal:

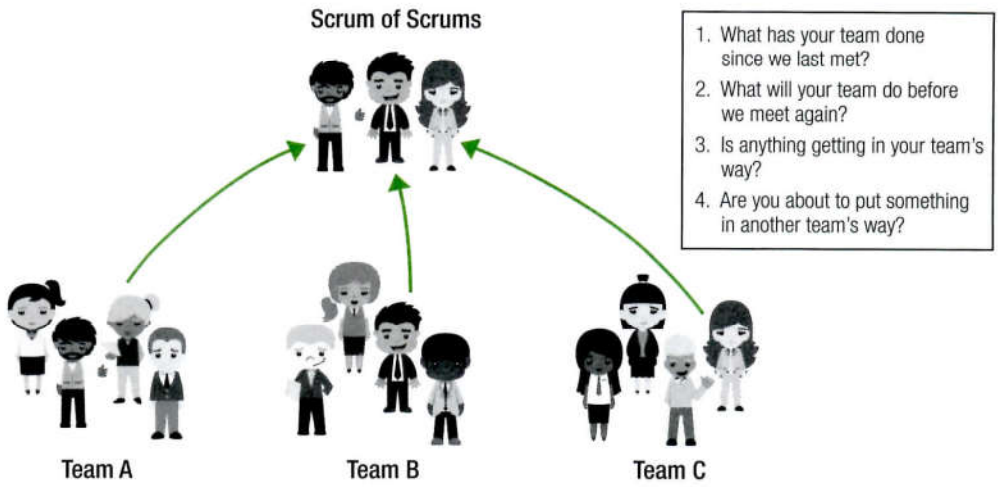
1. What have I done since the last daily scrum?
2. What do I plan to do today?
3. Are there any impediments to my progress?

These questions help the team members assess their progress toward the sprint goal. In answering the questions, they address each other, and any further discussion or problem solving takes place offline. (We'll cover these meetings in more detail in chapter 5, where we talk about daily stand-ups—the generic agile term for "daily scrum.")

As Scrum projects get larger, multiple Scrum teams might need to coordinate their work. To do this, they commonly use an approach called "scrum of scrums." As the name suggests, this is a meeting in which a representative from each team reports their progress to the representatives of the other teams. Just like a normal daily scrum, the participants answer three questions, and they might also address a fourth question to help surface potential conflicts between teams: "Are you about to put something in another team's way?" This process is illustrated below.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

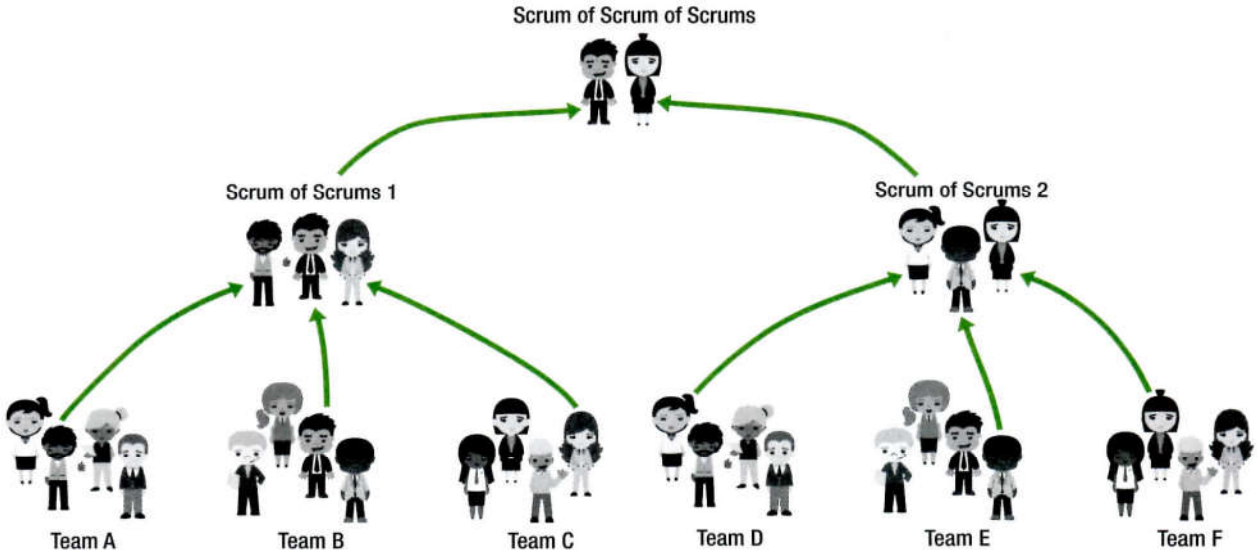
Figure 1.5: Scrum of Scrums



Like daily scrums, the scrum of scrums meetings take place on a regular schedule, but not necessarily daily. Let's say the individual teams have their daily scrum meeting at 9:00 a.m. After those meetings, at 9:30—perhaps daily or twice a week, as needed—someone from each team will take part in the scrum of scums meeting. If a participant learns about any issues that impact their team, they will report them to the rest of their team after the meeting.

Larger endeavors may even warrant a "scrum of scrum of scrums" where the teams repeat this pattern with a third-level meeting. Here, a representative from each scrum of scrums will attend a "scrum of scrum of scrums" to coordinate the work across a larger set of teams.

Figure 1.6: Scrum of Scrum of Scrums



Sprint Review

The sprint review meeting is held at the end of the sprint and includes the development team, the product owner, and the ScrumMaster (and potentially other stakeholders). In this meeting, the team demos the increment, or evolving product, that they built in the sprint to the product owner. The product owner inspects the work to see whether it is acceptable—deciding if it is "done" or explaining what is missing. The team and the product owner discuss the increment and the remaining items in the product backlog. Together, they make any changes needed to the backlog and decide what to work on next.

Sprint Retrospective

After the sprint review, but before the next sprint planning meeting, the development team holds their final "inspect and adapt" activity for the sprint—the sprint retrospective. Although others may be invited to participate, this meeting is primarily for the development team. This is their opportunity to gather lessons learned and look for opportunities for improvement. The timing of this meeting—after the sprint review, but before the next planning meeting—allows them to consider the product owner's feedback from the sprint review in their deliberations and also factor any improvements they identify during the retrospective into their plan for the next sprint.

The reflection that takes place during the retrospective might cover anything that occurred during the sprint, including the areas of people, process, and product. The team members explore what went well, look for opportunities for improvement, and decide what changes to implement in the next sprint.

Scrum Artifacts

Now that we've discussed the Scrum roles and activities, we'll turn to the scrum artifacts. These are three tangible items that are produced or used by the team during a sprint—the product increment, the product backlog, and the sprint backlog.

Product Increment

During each sprint, the members of the development team build an increment of the solution (the end product of the project). As we've seen, in the sprint review, they demo their latest increment to get feedback from the product owner and find out if that item is "done." To maximize the chances that the product increment will be acceptable, the team and the product owner need to agree upon a definition of done before the team begins working on it so that everyone has a shared understanding of what "done" will look like for that increment.

Product Backlog

The product backlog is a prioritized list of all the work that needs to be done to build the product—it serves as the single source for all the product requirements. The items in the backlog may include features to be built, functions, requirements, quality attributes (i.e., nonfunctional requirements), enhancements, and fixes. This list is dynamic; it evolves as the product evolves and needs to be continually updated.

The items in the backlog are prioritized by the product owner and sorted so that the highest-priority work is at the top of the list. The team will always try to work on these top-priority items next—or as soon as possible, given dependencies and other factors. The work items are progressively refined as they get closer to being worked on, so the highest-priority items are the most detailed, and the team's estimates for those items are more precise. The lowest-priority items might not get developed at all, since they may be continually deferred in favor of higher-priority work.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

The backlog is constantly being refined, and the process of adding more detail to the backlog items and refining their estimates is the "backlog refinement" Scrum activity described earlier, which is also known as "grooming the backlog." This activity is done by the development team and the product owner working together. The team estimates the work items, and product owner prioritizes them. For example, each time the team refines their estimates with a higher level of detail, the product owner might need to adjust the priority of those items in the backlog.

Sprint Backlog

The sprint backlog is the subset of items in the product backlog that have been selected as the goal of a specific sprint (the sprint backlog isn't a separate list from the product backlog). Along with the sprint backlog, the team develops a plan for how they will achieve the sprint goal—this is their commitment for the functionality that they will deliver in the sprint. The sprint backlog serves as a highly visible view of the work being undertaken and may only be updated by the development team.

EXERCISE: SCRUM OWNERSHIP/RESPONSIBILITY

In the following table, place a check mark in the appropriate column(s) for the development team, product owner, or ScrumMaster to indicate who owns or is responsible for each item.

Item	Development Team	Product Owner	ScrumMaster
Estimates			
Backlog priorities			
Agile coaching			
Coordination of work			
The definition of "done"			
Process adherence			
Technical decisions			
Sprint planning			

ANSWER

Item	Development Team	Product Owner	ScrumMaster
Estimates	×		
Backlog priorities		×	
Agile coaching			×
Coordination of work	×		
The definition of "done"	×	×	×
Process adherence			×
Technical decisions	×		
Sprint planning	×	×	×

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

Extreme Programming (XP)

Extreme Programming—commonly referred to as "XP" based on the initials eXtreme Programming—is an agile method that is focused on software development. While Scrum focuses at the project management level on prioritizing work and getting feedback, XP focuses on software development best practices. Therefore, there will be a lot of software references in the following discussion. As with the Agile Manifesto values and principles, as you read this section see if you can think of ways that XP values and practices can be applied to other kinds of knowledge work projects.

XP Core Values

The core values of XP are simplicity, communication, feedback, courage, and respect; these values manifest themselves in the practices undertaken throughout the XP life cycle.

» **Simplicity**: This value focuses on reducing complexity, extra features, and waste. XP teams keep the phrase "Find the simplest thing that could possibly work" in mind, and build that solution first.

» **Communication**: This value focuses on making sure all the team members know what is expected of them and what other people are working on. For example, the daily stand-up meeting is a key communication tool (these meetings will be explained in chapter 5).

» **Feedback**: The team should get impressions of suitability early. Failing fast can be useful, especially if in doing so we get new information while we still have time to improve the product.

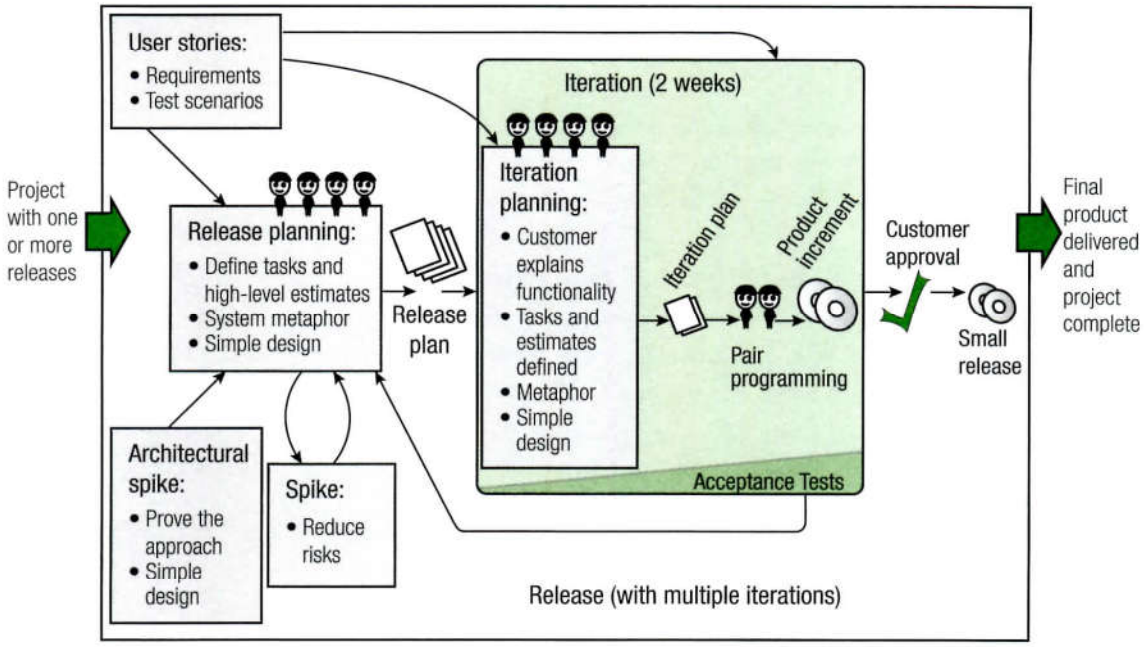
» **Courage**: It takes courage to allow our work to be entirely visible to others. In pair programming (described below under "XP Core Practices"), team members share code and often need to make bold simplifications and changes to that code. Backed up by automated builds and unit tests, developers need to have the confidence to make important changes.

» **Respect**: Respect is essential on XP projects, where people work together as a team and everyone is accountable for the success or failure of the project. This value also relates to pair programming; team members need to recognize that people work differently, and respect those differences.

The figure below illustrates the XP process. If some of the terms on this diagram are Greek to you, don't worry; we'll revisit the technical terms used in this diagram and in the discussion of XP that follows in more detail in later chapters.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

Figure 1.7: XP Life Cycle



As shown in this diagram, XP teams use lightweight requirements called "user stories" to plan their releases and iterations. Iterations are typically two weeks long, and developers work in pairs to write code during these iterations. All software developed is subjected to rigorous and frequent testing. Then, upon approval by the on-site customer, the software is delivered as small releases.

"Spikes" are periods of work undertaken to reduce threats and issues, and "architectural spikes" are iterations used to prove a technological approach. The spikes are blended into the release planning processes.

XP Team Roles

XP defines the team roles differently than Scrum—the XP roles are coach, customer, programmer, and tester. Let's see how each of these roles participates in an XP project.

Coach

The coach acts as a mentor to the team, guiding the process and helping the team members stay on track. The coach is also a facilitator—helping the team become more effective—and a conduit, reinforcing communication both within the team and across teams. This role shares many responsibilities with a ScrumMaster. Although the formal definitions of these two terms differ, they are often used interchangeably.

In addition to the coach, there may also be a manager who facilitates external communications and coordinates the team's activities, but this isn't a formal XP role.

Customer

On an XP team the "customer" is the business representative who provides the requirements, priorities, and business direction for the project. This person defines the product that will be built, determines the priority of its features, and confirms that the product actually works as intended. This role is similar to the product owner in Scrum.

Programmers

The programmers are the developers who build the product by writing and implementing the code for the requested user stories.

Testers

The testers provide quality assurance and help the customer define and write acceptance tests for the user stories. This role may also be filled by the developers (programmers), if they have the required skills.

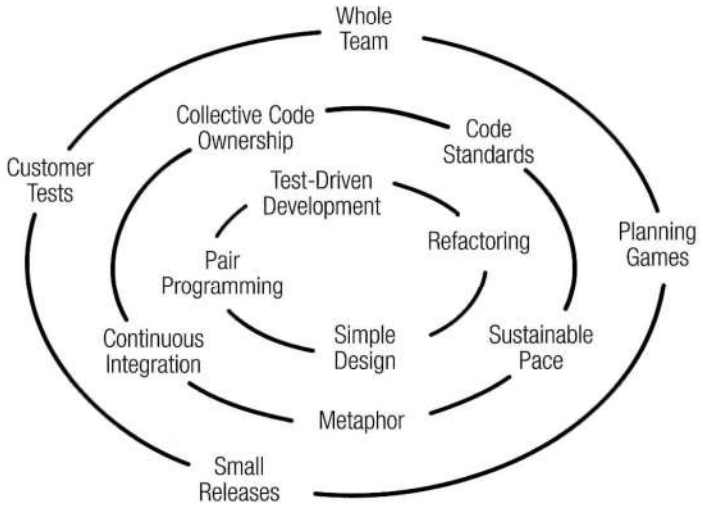
Exam tip

Think about the XP and Scrum terms for team roles—you might encounter an exam question that requires you to recognize a methodology based on the terms being used. You should also understand that from a broader agile perspective, "product owner" and "customer" are roughly equivalent roles, as are "ScrumMaster" and "coach."

XP Core Practices

The XP method draws upon 13 simple but powerful core practices, as shown below. We'll examine each of these practices in more detail, beginning with the outer ring and working our way in.

Figure 1.8: XP Core Practices



Whole Team

The whole team practice is the idea that all the contributors to an XP project sit together in the same location, as members of a single team. XP emphasizes the notion of generalizing specialists, as opposed to role specialists. In other words, anyone who is qualified to perform a role can undertake it—the roles are not reserved for people who specialize in one particular area. This practice helps optimize the use of resources, since people who can perform multiple jobs are able to switch from one role to another as the demand arises. The practice also allows for more efficient sharing of information and helps eliminate the possibility that people in certain roles will be idle or overstretched at certain points in the project.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

Planning Games

XP has two primary planning activities, or planning games—release planning and iteration planning.

A release is a push of new functionality all the way to the production user. A project typically has one or more releases, with no more than one or two releases in a single year. During release planning, the customer outlines the functionality required, and the developers estimate how difficult the functionality will be to build. Armed with these estimates and priorities, the customer lays out the plan for the project delivery. Since the initial attempts at estimating will likely be imprecise, this process is revisited frequently and improved as the priorities and estimates evolve.

Iterations are the short development cycles within a release that Scrum calls "sprints." Iteration planning is done at the start of every iteration. The customer explains what functionality they would like to see in the next iteration, and then the developers break this functionality into tasks and estimate the work. Based on these estimates (which are more refined than the release planning estimates) and the amount of work accomplished in the previous iteration, the team commits to the work items they think they can complete in the two-week period.

Small Releases

Frequent, small releases to a test environment are encouraged in XP, both at the iteration level, to demonstrate progress and increase visibility to the customer, and at the release level, to rapidly deploy working software to the end users. Quality is maintained in these short delivery timeframes by rigorous testing and through practices like continuous integration, in which suites of tests are run as frequently as possible.

Customer Tests

As part of defining the required functionality, the customer describes one or more test criteria that will indicate that the software is working as intended. The team then builds automated tests to prove to themselves and the customer that the software has met those criteria.

Collective Code Ownership

In XP, any pair of developers can improve or amend any code. This means multiple people will work on all the code, which results in increased visibility and broader knowledge of the code base. This practice leads to a higher level of quality; with more people looking at the code, there is a greater chance defects will be discovered. There is also less of an impact to the project if one of the programmers leaves, since the knowledge is shared.

Code Standards

Although collective code ownership has its advantages, allowing anyone to amend any code can result in issues if the team members take different approaches to coding. To address this risk, XP teams follow a consistent coding standard so that all the code looks as if it has been written by a single, knowledgeable programmer. The specifics of the standard each team uses are not important; what matters is that the team takes a consistent approach to writing the code.

Sustainable Pace

XP recognizes that the highest level of productivity is achieved by a team operating at a sustainable pace. While periods of overtime might be necessary, repeated long hours of work are unsustainable and counterproductive. The practice of maintaining a sustainable pace of development optimizes the delivery of long-term value.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

Metaphor

XP uses metaphors and similes to explain designs and create a shared technical vision. These descriptions establish comparisons that all the stakeholders can understand to help explain how the system should work. For example, "The billing module is like an accountant who makes sure transactions are entered into the appropriate accounts and balances are created."

Even if the team cannot come up with an effective metaphor to describe something, they can use a common set of names for different elements to ensure everyone understands where and why changes should be applied.

Continuous Integration

Integration involves bringing the code together and making sure it all compiles and works together. This practice is critical, because it brings problems to the surface before more code is built on top of faulty or incompatible designs.

XP employs continuous integration, which means every time a programmer checks in code to the code repository (typically several times a day), integration tests are run automatically. Such tests highlight broken builds or problems with integration, so that the problems can be addressed immediately.

Test-Driven Development

Testing is a critical part of the XP methodology. To ensure good test coverage so that problems can be highlighted early in development, XP teams often use the practice of test-driven development. With this approach, the team writes the acceptance tests prior to developing the new code.

If the tests are working correctly, the initial code that is entered will fail the tests, since the required functionality has not yet been developed. The code will pass the test once it is written correctly. The test-driven development process strives to shorten the test-feedback cycle as much as possible to get the benefits of early feedback.

Refactoring

Refactoring is the process of improving the design of existing code without altering its external behavior or adding new functionality. By keeping the design efficient, changes and new functionality can easily be applied to the code. Refactoring focuses on removing duplicated code, lowering coupling (dependent connections between code modules), and increasing cohesion.

Simple Design

By focusing on keeping the design simple but adequate, XP teams can develop code quickly and adapt it as necessary. The design is kept appropriate for what the project currently requires. It is then revisited iteratively and incrementally to ensure it remains appropriate.

XP follows a deliberate design philosophy that asks, "What is the simplest thing that could work?" as opposed to complex structures that attempt to accommodate possible future flexibility. Since code bloat and complexity are linked to many failed projects, simple design is also a risk mitigation strategy.

Pair Programming

In XP, production code is written by two developers working as a pair. While one person writes the code, the other developer reviews the code as it is being written—and the two change roles frequently. This practice may seem inefficient, but XP advocates assert that it saves time because the pairs catch issues early and there is a benefit in that the two people will have a larger knowledge base. Working in pairs also helps spread knowledge about the system through the team.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

By taking a disciplined and rigorous approach to applying these practices, XP teams succeed in delivering high-quality software systems.

EXERCISE: XP CHUNK OR MADE-UP JUNK?

Separate the real XP practices and concepts (the XP Chunks) from the made-up terms (Made-up Junk) in the following table by placing a check mark in the appropriate column.

Item	XP Chunk	Made-up Junk
Sample design		
Small revisions		
Courage		
Passion		
Execution game		
Complexity		
Refactoring		
Remarketing		
Simplicity		
Architectural spoke		
Iteration		
Sustainable pace		
Continuous interpretation		

ANSWER

Item	XP Chunk	Made-up Junk
Sample design		×
Small revisions		×
Courage	×	
Passion		×
Execution game		×
Complexity		×
Refactoring	×	
Remarketing		×
Simplicity	×	
Architectural spoke		×
Iteration	×	
Sustainable pace	×	
Continuous interpretation		×

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

Lean Product Development

Strictly speaking, lean is not an agile methodology; however, the lean approach is closely aligned with agile, as we'll see. Lean originated in the Toyota Production System that was developed to improve upon Henry Ford's mass production system for building cars.[6] So lean began as a manufacturing approach, which was then applied to software development, and eventually adapted for other kinds of knowledge work.

When referring to lean in an agile context, we are really talking about a subset of lean called "lean product development." While the original lean production systems dealt with manufacturing products, lean product development deals with developing new and better products. The high-level principles of lean product development include:

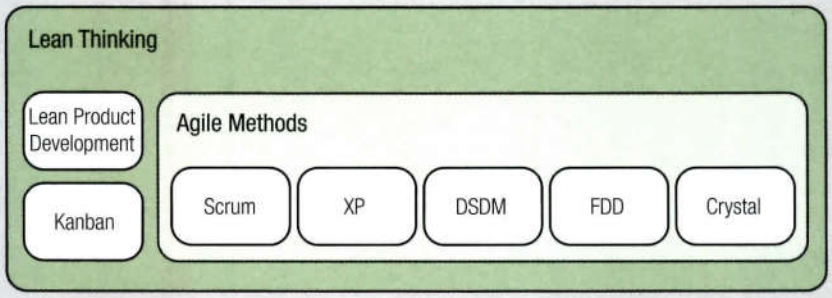
» Using visual management tools
» Identifying customer-defined value
» Building in learning and continuous improvement

Background information

How Are Agile and Lean Related?

The high-level lean principles listed above are common to Kanban and all agile methods. Although opinions vary, I see lean as a superset of agile and Kanban methods. From this viewpoint, lean product development, Kanban, and agile methods are all specialized instances of lean thinking. They are offshoots that start with lean's general principles and add further guidance for their specific domains. While this viewpoint won't be tested on the exam, it may help you understand the relationship between lean, Kanban, and agile methods.

Figure 1.9: Relationship between Lean, Kanban, and Agile Methods



© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

Lean Core Concepts

Lean focuses on seven core concepts, as shown below:

Figure 1.10: Lean Core Concepts

Let's examine these seven concepts in more detail:

» **Eliminate waste**: To maximize value, we must minimize waste. In knowledge work, waste can take the form of partially done work, delays, handoffs, unnecessary features, etc. Therefore, to increase the value we are getting from projects, we must develop ways to identify, and then remove, waste.

» **Empower the team**: Rather than taking a micromanagement approach, we should respect the team members' superior knowledge of the technical steps required on the project and let them make local decisions to be productive and successful.

» **Deliver fast**: We can maximize the project's return on investment (ROI) by quickly producing valuable deliverables and iterating through designs. We will find the best solution through the rapid evolution of options.

» **Optimize the whole**: We aim to see the system as more than the sum of its parts. We go beyond the pieces of the project and look for how it aligns with the organization. As part of optimizing the whole, we also focus on forming better intergroup relations.

» **Build quality in**: Lean development doesn't try to "test in" quality at the end; instead, we build quality into the product and continually assure quality throughout the development process, using techniques like refactoring, continuous integration, and unit testing.

» **Defer decisions**: We balance early planning with making decisions and commitments as late as possible. For example, this may mean reprioritizing the backlog right up until it is time to do the work, or avoiding being tied to an early technology-bounded solution.

» **Amplify learning**: This concept involves facilitating communication early and often, getting feedback as soon as possible, and building on what we learn. Since knowledge work projects are business and technology learning experiences, we should start early and keep learning.

Exam tip

The exam questions may ask about these seven core concepts, either directly or indirectly. However, you should be aware that these ideas have been worded in different ways. This means that you can't just memorize the seven phrases listed above; you have to understand these concepts well enough to recognize them even if they are worded a bit differently.

The Seven Wastes of Lean

Although it's important to understand all seven of these concepts, the goal of eliminating waste is the primary driver for the lean approach. Lean uses the Japanese term *muda* to refer to the seven kinds of wastes that should be eliminated. Lean experts Mary and Tom Poppendieck, who have written extensively on the use of lean in software projects, have converted the seven traditional manufacturing wastes into seven comparable software development wastes, as shown below. As you read these examples, think about how these wastes could apply to other kinds of knowledge work.[7]

Waste	Description	Example
Partially done work	Work started, but not complete; partially done work can entropy	Code waiting for testing Specs waiting for development
Extra processes	Extra work that does not add value	Unused documentation Unnecessary approvals
Extra features	Features that are not required, or are thought of as "nice-to-haves"	Gold-plating Technology features
Task switching	Multitasking between several different projects when there are context-switching penalties	People assigned to multiple projects
Waiting	Delays waiting for reviews and approvals	Waiting for prototype reviews Waiting for document approvals
Motion	The effort required to communicate or move information or deliverables from one group to another; if teams are not co-located, this effort may need to be greater	Distributed teams Handoffs
Defects	Defective documents or software that needs correction	Requirements defects Software bugs

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

EXERCISE: CATEGORIZE WASTES

Using the Poppendiecks' table of wastes as a reference, categorize the following day-to-day office wastes shown in the table below. The first one is done for you as an example.

Activity	Type of Waste
Queuing for elevator	*Waiting*
Rebooting a computer after a program crash	
Saving documents in old formats for compatibility	
Creating notices in French and Spanish to comply with company standards, even though nobody at your location speaks these languages	
Submitting stationery and letterhead orders for approval	

ANSWER

Activity	Type of Waste
Queuing for elevator	Waiting
Rebooting a computer after a program crash	Defects, waiting
Saving documents in old formats for compatibility	Extra processes, motion
Creating notices in French and Spanish to comply with company standards, even though nobody at your location speaks these languages	Extra processes, extra features
Submitting stationery and letterhead orders for approval	Extra processes, motion, partially done work

Of course, these types of activities aren't always waste. Waste occurs only if there aren't any benefits to be gained from the activity. So while safety notices should be provided for anyone likely to need them, unnecessary translations of such notices can be considered waste. Likewise, approvals for stationery and letterhead orders might be useful to prevent abuse or petty theft, but for people who just need some basic supplies, this process adds no value and can be considered waste.

Lean has contributed important techniques and concepts to agile, including the seven forms of waste, pull systems, value stream mapping, and work in progress, or WIP. (These concepts will be described as we continue.) As we've seen, lean is also the source of the Kanban methodology, which will be discussed next. However, before moving on, try your hand at the next exercise, which is designed to reinforce your understanding of lean principles.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

EXERCISE: MATCH AGILE PRACTICES TO LEAN PRINCIPLES

In the following table, match the agile practice to the most relevant lean principle.

Agile Practice	Lean Principle						
	Eliminate waste	Empower the team	Deliver fast	Optimize the whole	Build quality in	Defer decisions	Amplify learning
Teams make their own decisions							
Just-in-time iteration planning							
Team retrospectives							
Two-week iterations							
Unit test as we go							
Shadow the users to learn what they do							
The evolving prototype is the specification							

ANSWER

While some of these agile practices map to multiple lean principles, the most direct matches are indicated below:

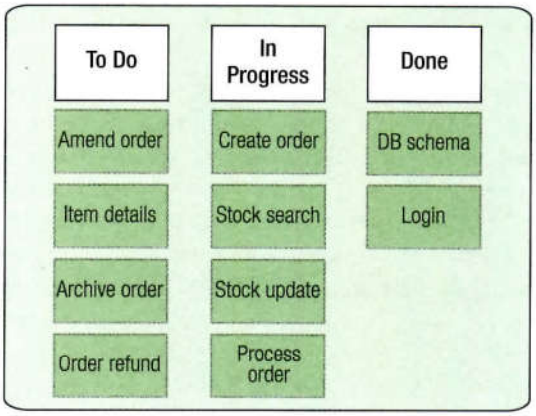
Agile Practice	Lean Principle						
	Eliminate waste	Empower the team	Deliver fast	Optimize the whole	Build quality in	Defer decisions	Amplify learning
Teams make their own decisions		×					
Just-in-time iteration planning						×	
Team retrospectives							×
Two-week iterations			×				
Unit test as we go					×		
Shadow the users to learn what they do				×			
The evolving prototype is the specification	×						

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

Kanban

The Kanban method is derived from the lean production system developed at Toyota. "Kanban" is a Japanese word meaning "signboard." The signboard, or Kanban board as it is called, plays an important role in the Kanban methodology. This board shows the work items in each stage of the production process, as defined by the team. Here is a simple example of such a board:

Figure 1.11: Kanban Board Example



Five Principles of Kanban

Kanban development operates on five core principles:

» **Visualize the workflow**. Knowledge work projects, by definition, manipulate knowledge, which is intangible and invisible. Therefore, having some way to visualize the workflow is very important for organizing, optimizing, and tracking it.

» **Limit WIP (work in progress)**. Restricting the amount of work in progress improves productivity, increases the visibility of issues and bottlenecks, and facilitates continuous improvement. This makes it easier for the team to identify issues and minimize the waste and cost associated with changes. It also results in a steady "pull" of work through the development effort, since new work can only be moved forward as existing work is completed.

» **Manage flow**. By tracking the flow of work through a system, issues can be identified and changes can be measured for effectiveness.

» **Make process policies explicit**. It is important to clearly explain how things work so the team can have open discussions about improvements in an objective, rather than an emotional or subjective, way.

» **Improve collaboratively**. Through scientific measurement and experimentation, the team should collectively own and improve the processes it uses.

Kanban's Pull System

Kanban has some distinct features that differentiate it from Scrum, XP, and generic agile. The main difference you should understand for the exam is that Kanban teams employ a "pull system" to move work through the development process, rather than planning their work in timeboxed iterations. Each time a Kanban team completes an item of work, it triggers a "pull" to bring in the next item they will work on.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

As we'll see, only a certain number of slots are available for each column on the Kanban board, and whenever there is an empty slot on the board, that's a signal for the team to pull work from the previous stages, if there are any items available. So work is continuously being pulled from the left side of the board to the right side.

(Note: This might look the same as moving work items through the process from left to right on an agile task board, but it's not. On a Kanban board, capacity is what signals the team to pull work items into the next stage of the process. On an agile task board, the team simply moves the work items from left to right to show their status in the process.)

This pull system means that Kanban has much less emphasis on iterations than agile methods. Although a Kanban team may use iterations if they wish, the pull mechanism just described means that there's no need for it. Of course, most organizations will want the team to commit to a delivery cadence that defines how often the work increments will be available for consumption. However, the team's development cadence doesn't need to be coupled to that delivery cadence, or any aggregation of it.

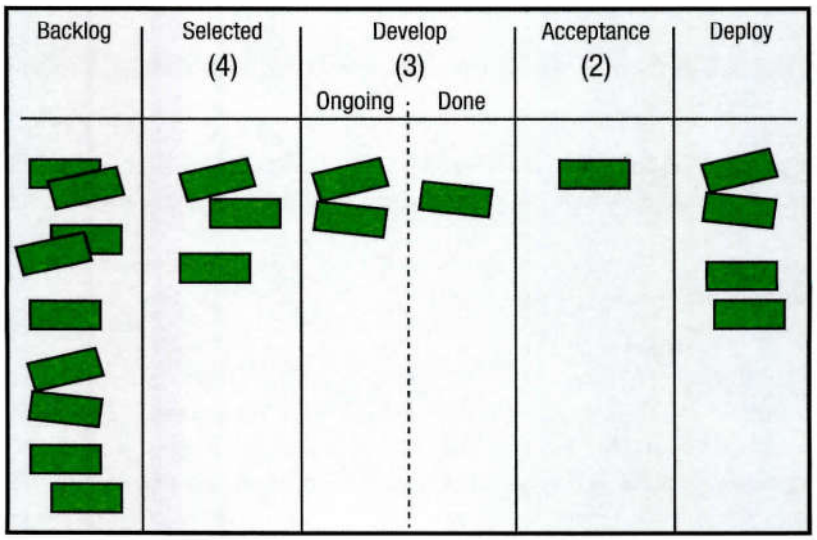
For example, an agile team might decide to use two-week iterations with deployments into a test environment as well as live deployments to the customers every quarter. So within each quarterly live deployment timeframe (13 weeks) they would typically get six iterations of work done. So, the team knows they will have six iterations between each major release and will plan their features and stories accordingly.

A Kanban team, on the other hand, might still have quarterly customer releases, but they wouldn't use any internal iterations. Instead, any accepted work packages they have ready would be candidates for the next release. They can still plan and track their work for the releases, but instead of using velocity metrics such as points accepted per iteration, they would use cycle time, lead time, and throughput metrics (as described in chapter 6).

WIP Limits in Kanban

WIP limits will be explained more fully in the next chapter—but basically, this term refers to capping the number of items that can be in a given state of progress, as defined by the columns on the team's Kanban board. Once the limit at the top of a column is reached, no new items may be moved into that column until another item is moved out. Here's an example of a Kanban board with WIP limits:

Figure 1.12: Kanban Board with WIP Limits

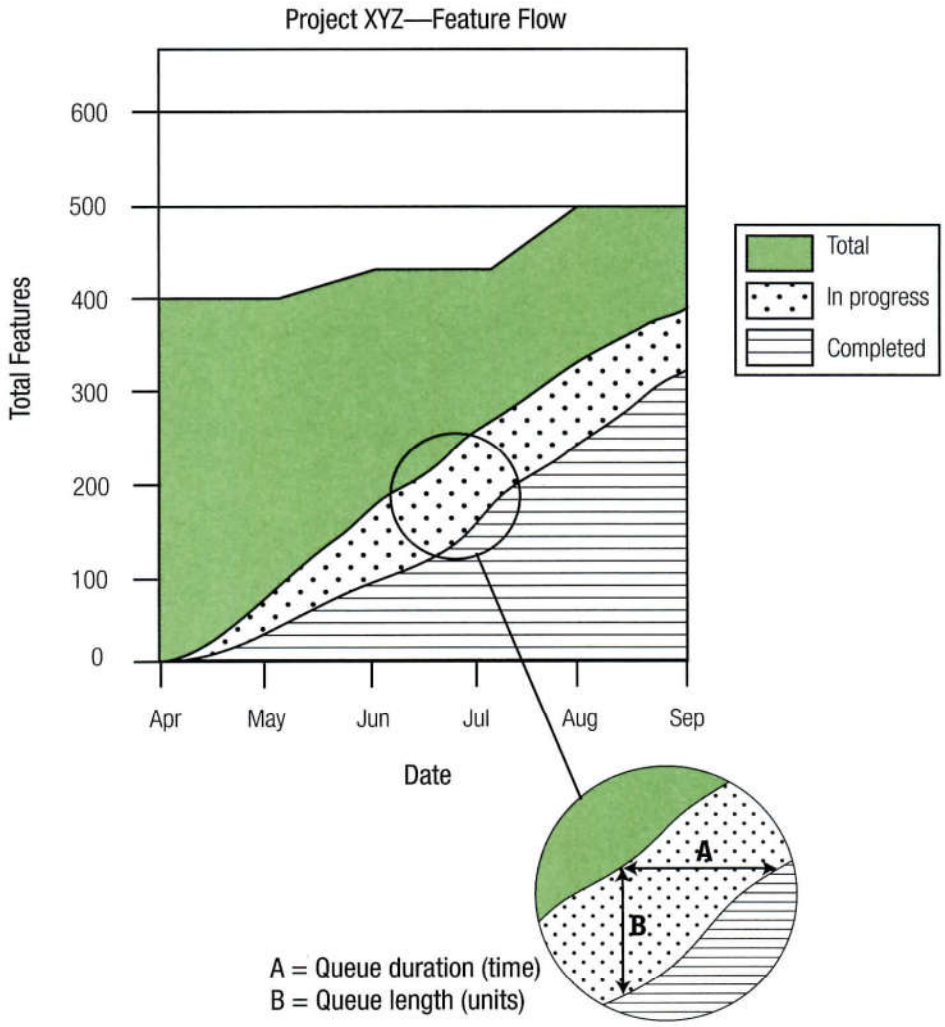


© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

In this figure, we see the team's backlog in the first column. The second column shows that there are three user stories currently selected for development (as shown by the green index cards), which is fine, since the WIP limit at the top of that column is four. The Develop and Acceptance columns also have WIP limits—three and two, respectively.

Why is limiting WIP so important? The reason is that lowering WIP actually increases a team's productivity—it speeds up the rate at which the work is completed. If that claim seems debatable, the relationship between work in progress and productivity has actually been proven mathematically by Little's Law, as illustrated below. This law demonstrates that the *duration* of a queue (how long it will take to complete the work) is proportional to its *size* (how much work is in progress). In other words, teams that limit WIP will get their work done faster.

Figure 1.13: Little's Law

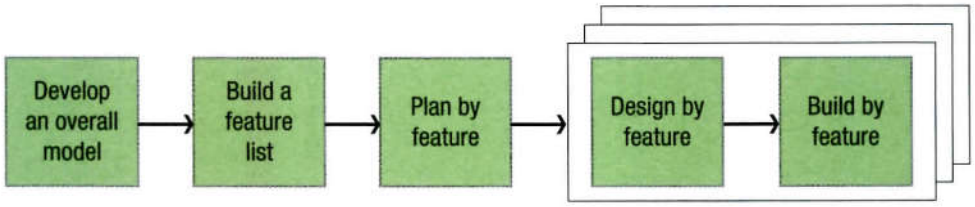


The Kanban development methodology has helped popularize task boards with WIP limits and pull systems created by limiting WIP. It also gives permission and encourages team members to try new approaches and change processes. With a continuous pull model, iterations may not be required, and therefore activities like creating estimates can be considered waste and reduced or eliminated entirely.

Feature-Driven Development (FDD)

Feature-Driven Development (FDD) is a simple-to-understand yet powerful approach to building products or solutions. A project team following the FDD method will first develop an overall model for the product, build a feature list, and plan the work. The team then moves through design and build iterations to develop the features.

Figure 1.14: Feature-Driven Development Process



FDD recommends a set of good practices, derived from software engineering. These practices include:

» **Domain object modeling**: In this practice, teams explore and explain the domain (or business environment) of the problem to be solved.

» **Developing by feature**: This involves breaking functions down into two-week or shorter chunks of work and calling them features.

» **Individual class (code) ownership**: With this practice, areas of code have a single owner for consistency, performance, and conceptual integrity. (Note that this is quite different from XP's collective code ownership idea that aims to spread the knowledge to other team members.)

» **Feature teams**: These are small, dynamically formed teams that vet designs and allow multiple design options to be evaluated before a design is chosen. Feature teams help mitigate the risks associated with individual ownership.

» **Inspections**: These are reviews that help ensure good-quality design and code.

» **Configuration management**: This involves labeling code, tracking changes, and managing the source code.

» **Regular builds**: Through regular builds, the team makes sure the new code integrates with the existing code. This practice also allows them to easily create a demo.

» **Visibility of progress and results**: This practice tracks progress based on completed work.

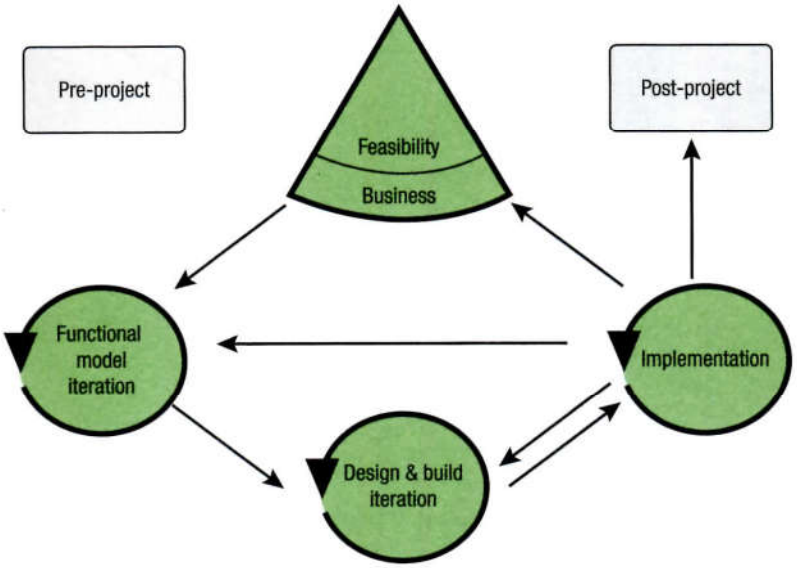
Feature-Driven Development is the agile methodology that popularized cumulative flow diagrams (discussed in chapter 2) and parking lot diagrams, which are one-page summaries of project progress. Both are useful tracking and diagnostic tools that are now used by other agile approaches.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

Dynamic Systems Development Method (DSDM)

DSDM was one of the earlier agile methods, and it started out quite prescriptive and detailed. Its coverage of the project life cycle is broad, encompassing aspects of an agile project ranging from feasibility and the business case to implementation. The figure below illustrates the DSDM life cycle.

Figure 1.15: DSDM Life Cycle



DSDM is centered on eight principles. Although these principles were created before the Agile Manifesto was written, they are closely aligned to the Manifesto. The eight principles are:

1. Focus on the business need
2. Deliver on time
3. Collaborate
4. Never compromise quality
5. Build incrementally from firm foundations
6. Develop iteratively
7. Communicate continuously and clearly
8. Demonstrate control

DSDM has influenced the development of agile by helping to popularize early architectural considerations, agile suitability filters, and agile contracts.

Crystal

Crystal isn't just one method; it consists of a family of situationally specific, customized methodologies that are coded by color names. Each methodology is customized by criticality and team size, which allows Crystal to cover a wide range of projects, from a small team building a low-criticality system (Crystal Clear) to a large team building a high-criticality system (Crystal Magenta).

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

Figure 1.16: Agile Process Overview

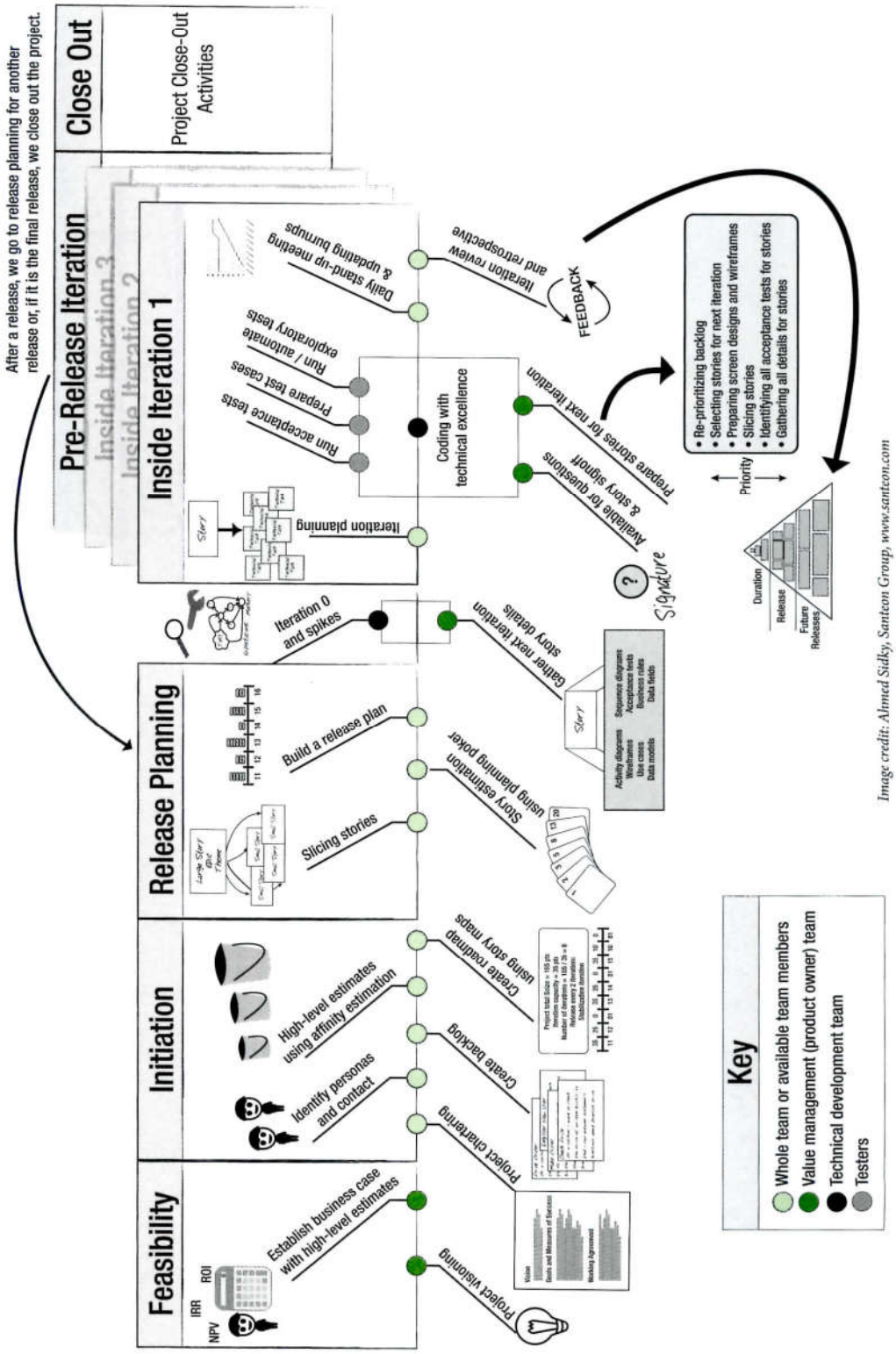


Image credit: Ahmed Sidky, Santeon Group, www.santeon.com

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

The Crystal framework provides a great example of how agile methods can be tailored to match the specific characteristics of a project. For example, the figure below shows the first five "colors" (methodologies) of Crystal.

Figure 1.17: Crystal Methods

Criticality*	Crystal Clear	Crystal Yellow	Crystal Orange	Crystal Red	Crystal Magenta
Life					L200
Essential		E20	E40	E100	E200
Discretionary	D6	D20	D40	D100	D200
Comfort	C6	C20	C40	C100	C200
	1–6	*7–20*	*21–40*	*41–100*	*101–200*
			Team Size		

* *"Criticality" refers to the potential impact of a product defect—whether it could cause loss of comfort, discretionary funds, essential funds, or life.*

In this diagram, team size is shown on the horizontal axis and project criticality is shown on the vertical axis. For example, we can see that Crystal Clear is designed for small projects with a core team size of one to six people, in which failure might result in a loss of comfort (for example, a video game crash) or a low-level financial loss (for example, a word processor crash resulting in lost business time). As projects engage more people and develop increasingly critical applications, we need to scale our processes beyond face-to-face communications and introduce additional validation and traceability measures.

For the exam, you don't need to understand the details of how the different Crystal methods work, but you should understand how Crystal uses the factors of criticality and team size to classify projects, as shown above. Crystal's use of this scaling method to determine the best methodology for a project has significantly influenced agile thinking. Another influential concept that Crystal has contributed to agile is the concept of osmotic communication (covered in chapter 4). Crystal methodologies also embrace and promote many other agile principles.

Agile Process Overview

The remaining chapters explain the agile tools, practices, and concepts that are covered in the remaining exam domains. The full-page figure on the facing page provides a high-level, timeline-oriented view of when many of the activities we will be discussing typically occur in a simple agile project, including which activities are iterated (repeated). As you read this book, revisit this diagram as needed to see how the different concepts fit into the overall agile process.

There is one thing to bear in mind, however. This diagram shows the typical process for a generic agile project, but that process isn't the same for lean and Kanban projects. Lean and Kanban teams might not use iterations, instead using a "pull model" to move work through the system. They would still use demonstrations, reviews, and retrospectives, but they wouldn't directly link the timing of these meetings to the delivery of an increment of functionality, as generic agile does at the end of each iteration.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

K&S Agile Leadership

Agile is more humanistic than mechanistic, as evidenced by the agile value of "Individuals and interactions over processes and tools." The concept of valuing people over processes goes beyond how we manage the work to be done on the project; it also impacts how we organize and motivate our team members, as well as how we assume our role as leaders. Leadership is about tapping into people's intrinsic motivations. To be effective leaders, we need to discover why team members want to do things, understand what motivates them, and then align their project tasks and goals accordingly. It is by aligning project objectives with personal objectives that we can get higher levels of productivity.

Management versus Leadership

The leadership techniques employed on agile projects involve taking an interpersonal approach, rather than directing, command-and-control methods. There is a famous quote from Warren Bennis that speaks to the difference between these two approaches: "Management is getting people to do what needs to be done. Leadership is getting people to want to do what needs to be done."[8]

Instead of telling people what to do, we need to create an environment where people want to do what needs to be done. The difference between the two environments is like the difference between pulling a rope (when people want to do what needs to be done) and pushing the rope (when people are simply told what to do).

Management has a more mechanical focus than leadership; it is concerned with tasks, control, and speed. In contrast, leadership assumes a humanistic focus on people and purpose; it is more concerned with empowerment, effectiveness, and doing the right things. The following table illustrates the differences between a management focus and leadership focus:

Management Focus	Leadership Focus
Tasks/things	People
Control	Empowerment
Efficiency	Effectiveness
Doing things right	Doing the right things
Speed	Direction
Practices	Principles
Command	Communication

So does this mean that leadership is better than management? Can we have just leadership without management? No; we definitely need the mechanics of management in place. But to be truly effective, we need to layer leadership on top of those mechanics. We can best amplify team productivity through a combination of management and leadership.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

EXERCISE

Review the activities in the table below, and determine whether they are mainly leadership- or management-based.

Activity	Mainly Leadership or Management?
Human resource management	
Career planning	
Team time tracking	
Team member recognition	
Task assignment	
Team brainstorming	
Planning workshops	
Creating Gantt charts	

ANSWER

Activity	Mainly Leadership or Management?
Human Resource management	Management
Career planning	Leadership
Team time tracking	Management
Team member recognition	Leadership
Task assignment	Management
Team brainstorming	Leadership
Planning workshops	Leadership
Creating Gantt charts	Management

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

T&T Servant Leadership

Agile promotes a servant leadership model that recognizes it is the team members, not the leader, coach, or ScrumMaster, who get the technical work done and achieve the business value. The servant leadership approach redefines the leader's role in relation to the team. It focuses the leader on providing what the team members need, removing impediments to their progress, and performing supporting tasks to maximize their productivity.

There are four primary duties a leader performs in this role of serving the team:

1. **Shield the team from interruptions**. Servant leaders need to isolate and protect the team members from diversions, interruptions, and requests for work that aren't part of the project.

   When business representatives are closely involved in a project, it can be tempting for them to make side requests for changes or enhancements directly to the developers that would sidetrack the planned development effort. While agile projects positively encourage these business insights and requests, they need to go through the proper channels. Business representatives should make such requests during the iteration planning meeting or submit them to the product owner for inclusion in the backlog. Part of being a servant leader involves reminding people about the designated channels so the team can maintain their focus on the iteration and establish a reliable velocity that allows the team's progress to be used to help plan future work.

   Although it is also important to shield the team from internal diversions, the leader must be especially vigilant in protecting the team from external diversions. Breaking people away from focusing on the project work and moving them back and forth between initiatives saps productivity. If the leader is able to shield the team from noncritical external demands, their productivity will be enhanced. Physically co-locating team members is an effective way to prevent external interference. If people are still located in their functional departments or workspaces, it is too easy for them to be drawn into nonproject work.

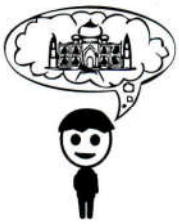
2. **Remove impediments to progress**. Servant leaders need to clear obstacles from the team's path that would cause delay or waste. These obstacles may include documentation or compliance activities that divert the team from completing the objectives of the current iteration. In lean terms, compliance work refers to efforts that do not directly contribute toward delivering business value. This might include duplicated time recording tasks, nonproject meetings, and other administrative activities.

   At the daily stand-up meeting where the team reports on its progress, planned work, and issues, the leader needs to note the issues and work to resolve them that same day, if possible. Removing or easing such impediments will allow the team to work faster and ultimately deliver more value to the business.

   Some agile project management tools now support impediment backlogs. These backlogs are a kind of prioritized obstacle removal list. Servant leaders can use such tools to track their impediment removal work.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

3. **Communicate (and re-communicate) the project vision.** This may seem like an odd duty to place in the category of servant leadership, but communicating and re-communicating the project vision is critical to successfully leading a team. Only if stakeholders have a clear image of the goals for the completed product and project can they align their decisions with, and work toward, the common project objective. In their best-selling book *The Leadership Challenge*, James Kouzes and Barry Posner say that leaders need to reveal a "beckoning summit" toward which others can "chart their course."[9] Simply stated, a common vision helps to keep people all pulling in the same direction.



   Divergent views commonly develop between well-intentioned team members. In software projects, for example, a developer's desire for simplicity or a new technology can cause his or her work to diverge from the user's requirements. An analyst's or quality assurance specialist's desire for completeness and conformance may diverge from the sponsor's requirements for progress and completion. Communicating and re-communicating the project vision helps stakeholders recognize these divergences and bring them back in line with the project's objectives.

   Kouzes and Posner found that the most effective leaders dedicate a much higher percentage of their work time to communicating and re-communicating the project and corporate vision than people in the lower leadership levels. They believe it is almost impossible for leaders to overcommunicate project vision, and state that it is a critical step for effective leadership.

   So agile projects should not just have a vision exercise at the project kickoff or when developing the iteration goals. Such a limited focus is not enough. Instead, servant leaders need to continually look for opportunities to communicate the project vision and find new ways to illustrate and reinforce that vision.

4. **Carry food and water**. This duty isn't literally about food and water; it's about providing the essential resources a team needs to keep them nourished and productive. Such resources could include tools, compensation, and encouragement. People who are fueled by professionalism and duty alone can't continue to contribute to the best of their ability, iteration after iteration. Leaders need to learn what motivates their team members as individuals and find ways to reward them for good work. As a simple measure, a great place to start is a sincere "thank you" to someone for their hard work.

   Leaders also need to celebrate victories—the large ones, of course, but also the small ones—as the project progresses. It is often tempting to save the project celebrations for the end, but if the team members aren't receiving some regular recognition, the project may never reach a successful conclusion. Celebrations and recognition help build momentum, and leaders need to nourish their teams with such rewards frequently to keep the project moving forward productively.

   Training and other professional development activities are also examples of resources the team may need to be productive. The ScrumMaster or coach should take an interest in and arrange appropriate training for the individuals on the team. By building the team members' skills, the project will gain the benefits of their new knowledge. Such actions also show that we're not just trying to extract work and information from the team members, we also want them to grow as individuals.

Figure 1.18: Servant Leadership Approach

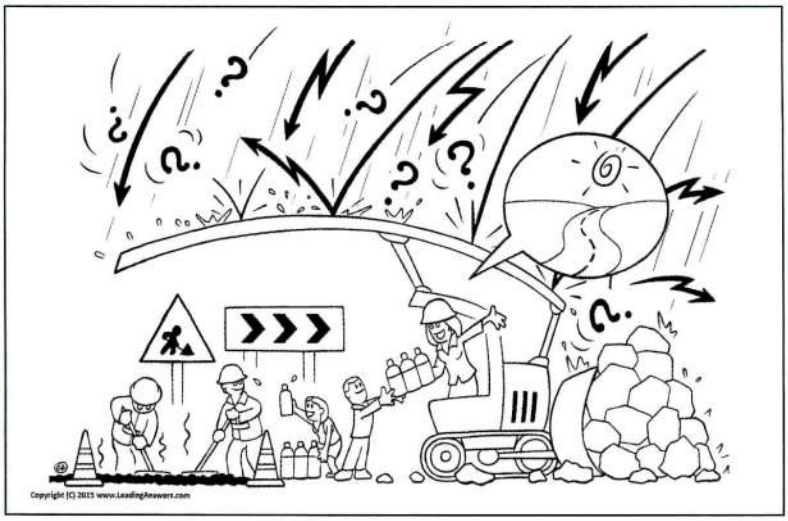


Image copyright © Leading Answers, Inc. Reproduced with permission from Leading Answers, Inc. www.leadinganswers.com

Twelve Principles for Leading Agile Projects

In addition to the four core duties we've just discussed, there are other activities that servant leaders should keep in mind. Jeffrey Pinto, in *Project Leadership: from Theory to Practice*, offers the following great list of principles for leaders to follow:[10]

» **Learn the team members' needs.** Find out what motivates and demotivates people.

» **Learn the project's requirements.** Talk to the customers and sponsors; find out their priorities.

» **Act for the simultaneous welfare of the team and the project.** Balance and promote the needs and desires of both the team and the other project stakeholders.

» **Create an environment of *functional accountability*.** Make sure people know what success and failure look like, and empower the team to self-organize to reach the goal. Be proud of accomplishments, but don't hide or shy away from failures—instead, examine them, learn from them, and adapt.

» **Have a vision of the completed project.** Create a "beckoning summit" to which others can chart their own course. When we are head-down in the weeds, it's good to know where we are trying to get to, so we can navigate our own course.

» **Use the project vision to drive your own behavior.** Model action toward the project goals.

» **Serve as the central figure in successful project team development.** Model desired behavior for the team.

» **Recognize team conflict as a positive step.** Unfiltered debate builds strong buy-in for well-discussed topics.

» **Manage with an eye toward ethics.** Be honest and ethical, because people don't want to be associated with goals or missions they feel are unethical.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

» **Remember that ethics is not an afterthought, but an integral part of our thinking.** You cannot add trust in later—like quality, it has to be a core ingredient.

» **Take time to reflect on the project.** Review, diagnose, and adapt; improve through progressive change and learning.

» **Develop the trick of *thinking backwards*.** This means imagining that we have reached the end goal and then working backwards to determine what had to happen to get there and what problems and risks we were able to avoid. So first discuss and decide what "done" will look like; then chart the path to get there, and plan how you will avoid any obstacles in your way.

We've already discussed some of these principles in other parts of this chapter, but this list provides a nice summary to keep in mind as we strive to be effective servant leaders.

Exam tip

You don't need to memorize these leadership principles, or the leadership practices we'll discuss next—the exam won't test these concepts directly. Instead, these are the unstated assumptions behind the exam questions that deal with leadership scenarios. So as you read this section, think about the implications of this approach to leadership, and aim to integrate these ideas into your agile mindset.

Agile Leadership Practices

As leaders, we can help create a productive project environment by using practices like modeling the behavior we want the team to follow, communicating the project vision, enabling stakeholders to act, and being willing to challenge the status quo.[11]

Model Desired Behavior

In *The Leadership Challenge,* Kouzes and Posner describe a 10-year study that asked more than 75,000 people, "What values, personal traits, or characteristics do you look for or admire in your leader?"[12] The following were the highest-ranked values:

» **Honesty**: People will not follow leaders they know are deceptive, since doing so undermines their own feelings of self-worth. Therefore, we should pay special attention to transparency and make sure we follow through on what we say we will do. We shouldn't hide our mistakes—we should admit them openly. This is not only a healthy approach for us as leaders, but it also sets an example for how we want our team to operate. We shouldn't do things like ask our team members for estimates and then say we will double the estimates before we give them to management. Such statements hurt our credibility with the team and give them reason to doubt our integrity. (A better approach would be to explain the concept of adding a contingency to estimates and discuss how to base that contingency on realistic expectations of the risks involved in the project.)

» **Forward-looking**: People expect those who lead them to understand where they are going. Leaders should be able to paint the picture for the team so everyone understands what they are ultimately aiming for.

» **Competent**: Leaders don't need to have the strongest technical skills on the team, since team members are typically happy to provide specialist knowledge when required. However, leaders should be competent and not be an embarrassment or liability to the group.

» **Inspiring**: People want to be inspired in their work, rather than be met each day with a sense of doom and gloom. Therefore, leaders need to find ways to explain the project's vision and journey with genuine enthusiasm and spirit.

When we embody these traits as leaders, not only do we encourage people to follow us, we also model the behaviors that we want our team members to emulate. We are, in effect, leading by example.

Communicate the Project Vision

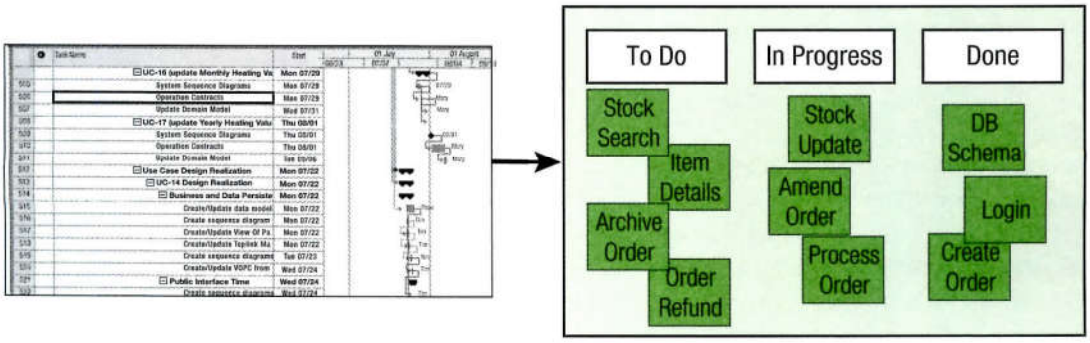
In our earlier discussion of the duties of a servant leader, we talked about the importance of communicating and re-communicating the project vision to keep stakeholders aligned with the project objectives. A leader can use a variety of practices to achieve this, based on what is most effective for the particular team. For example, XP teams use metaphors, some teams develop mantras, and other teams create elevator pitches or project Tweets, in which they explain the project purpose in 140 characters or less. Whatever the method used, it is important to frequently communicate the project's goals and objectives to ensure that all stakeholders are aware of and aligned with the vision.

Enable Others to Act

In order to enable our team members to feel confident in making decisions and taking actions that move the project forward in a productive way, we need to foster a collaborative environment. This involves building trust among team members and strengthening others by sharing power. We also need to create a safe work environment where people are not afraid to ask what they may think are dumb questions. People learn much more quickly when they can raise questions without fear of reprisal or ridicule.

Strengthening others by sharing power means the project manager, ScrumMaster, or leader does not keep the project plan or estimates to him- or herself. Instead, the leader makes sure that information and knowledge are spread throughout the team. For example, this might involve switching from planning tools like Gantt charts that only one or two people can update and maintain to using a task board that the whole team can engage with. In doing so, the planning and status information will be more accessible to the team, and the project will benefit because more people are able to vet, update, and optimize the plan.

Figure 1.19: Switching from Exclusive Tools Like Gantt Charts to Inclusive Tools Like Task Boards



© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

Be Willing to Challenge the Status Quo

Challenging the status quo means we search for innovative ways to change, grow, and improve—and then experiment and take risks by constantly generating small wins and learning from our mistakes. Iterations are perfect microcosms for experimentation. We can try new ideas for one or two iterations before committing to them. If the ideas work, we can institutionalize them. If they do not work, it's no big loss; at least we tried, and we learned something from the experience.

Allowing stakeholders to suggest new ideas for improvement and then giving them a chance to try out those ideas is one way to cement the concept that everyone's ideas have value. There is nothing more disheartening than having a good idea fall on deaf ears. If this happens, people will soon stop trying to make suggestions and will no longer care about the project. So to keep our stakeholders engaged, we should take advantage of the opportunities agile projects present us for small-scale, localized experiments in a supportive, low-risk environment.

As leaders, we need to encourage our team to challenge the status quo of how we operate, not only because the team members are in a great position to suggest process improvements, but also because doing so helps to motivate them. To be successful in this effort, we need to have analytical thinking skills to help our team brainstorm ideas and solutions and active listening skills to make sure we accurately understand their suggestions.

Leadership Tasks

Before moving on to the next chapter, we'll conclude our discussion of the agile principles and mindset by examining some of the leadership tasks in the exam content outline for this domain. Although these concepts will be discussed in more detail in later chapters, we'll introduce them here to help you understand not only these specific Domain I tasks, but also the underlying goals and values of the agile approach to leadership.

Practice Transparency through Visualization

"Transparency" means being open and honest, not only about our progress and achievements, but also about our issues and setbacks. Agile teams demonstrate transparency by openly displaying their work, progress, and review findings for other stakeholders to see. When walking into an agile team room, it's normal to see graphs showing velocity, defect rates, and the results of their last retrospective—including what is working well and what needs improvement. When there is nothing to hide and everything is shared, conversations can be more frank, and people can be less guarded and focus on improvements. People won't be scared or reluctant to discuss problems or mistakes, because the transparent environment all around them sets the example that it is accepted and expected to talk about issues as well as progress.

Create a Safe Environment for Experimentation

When people believe that it's okay to try new approaches and they won't be criticized if they don't succeed the first time, they are more likely to have a go and try new things. It is during these experiments that the most useful innovations and breakthroughs are often made. It's a common pattern that as people get more established in their careers, they grow more concerned about preventing failure and looking inferior to their peers, which can limit their willingness to experiment.

If a friend asks us to help out over the weekend as an entertainer at a children's party, we might try a number of different games to keep the kids happy. If the children don't like the first one, it's not a big problem—we'll just try something else. However, if we happen to have a PhD in child psychology, many published papers on child happiness, and a counseling practice at a high hourly rate, then fears for our credibility and reputation might limit our willingness to try new approaches.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com

On agile teams we need to create a safe environment for experimentation. This includes encouraging trials and tests, not dwelling on experiments that have failed, and encouraging the generation of new ideas. At Toyota they have a suggestion box for employees to submit ideas. Now, this is nothing new; most organizations have such a box. In most companies management will select the best one or two ideas to implement and then give the submitter a big bonus if their suggestion works out. However, Toyota gives small rewards to everyone who submits an idea. This results in *lots* of suggestions. In Matthew May's book, *The Elegant Solution*, he says that Toyota implements over one million suggestions each year! That's 3,000 suggestions or experiments per day. When we're innovating at that speed, it's really hard for competitors to catch up.[13]

Experiment with New Techniques and Processes

Not all suggestions for improvements are successful. Some likely-sounding ideas fail, and some unlikely ideas may work very well. The only real way to find out if a new process or technique is going to work for your project is to try it out. One important agile element that supports experimentation is short iterations. They provide perfect test cases for short experiments to see what works and should be institutionalized, and what does not and should be dropped.

Geneticists like to study inheritance patterns in mayflies because these insects reproduce in days, so they get their results quickly (also, it is illegal to breed people for scientific purposes). Short iterations offer the same short feedback cycles and provide ready-made test periods for new approaches.

Share Knowledge through Collaboration

It's important for teams to share knowledge about the product they are working on and how things work within their team. However, writing all this down takes a long time, and people often don't like to read it anyway. Engineers also have a deserved stereotype of not liking to produce documentation when they could be "doing real work" and even less of a penchant for "reading the instructions" when they could be tinkering.

This is why collaborative work—either through pairing workers or ad hoc knowledge transfer through co-location—has emerged as an agile best practice. By working with people we learn what they do, how they solve problems, and how to help them most effectively when they are stuck. By simply changing who people work with, project knowledge can be dispersed throughout the team, reducing the impact of information loss if a team member leaves.

Encourage Emergent Leadership via a Safe Environment.

Emergent leadership is when a team member takes the initiative and tries a new approach after gaining team approval. Agile team members are not only empowered to make decisions but also to lead the charge on improvements. If someone identifies a more efficient process, they are encouraged to try an experiment to prove if it can work.

© 2015 RMC Publications, Inc • 952.846.4484 • info@rmcls.com • www.rmcls.com