Part I

Core Concepts

This page intentionally left blank

Chapter 2
Scrum Framework

This chapter provides an overview of the Scrum framework with a primary focus on its practices, including roles, activities, and artifacts. Subsequent chapters will provide a deeper treatment of each of these practices, including an in-depth look at the principles that underlie the practices.

Overview

Scrum is not a standardized process where you methodically follow a series of sequential steps that are guaranteed to produce, on time and on budget, a high-quality product that delights customers. Instead, Scrum is a **framework** for organizing and managing work. The Scrum framework is based on a set of values, principles, and practices that provide the foundation to which your organization will add its unique implementation of relevant engineering practices and your specific approaches for realizing the Scrum practices. The result will be a version of Scrum that is uniquely yours.

To better grasp the framework concept, imagine that the Scrum framework is like the foundation and walls of a building. The Scrum values, principles, and practices would be the key structural components. You can't ignore or fundamentally change a value, principle, or practice without risking collapse. What you can do, however, is customize inside the structure of Scrum, adding fixtures and features until you have a process that works for you.

Scrum is a refreshingly simple, people-centric framework based on the values of honesty, openness, courage, respect, focus, trust, empowerment, and collaboration. Chapter 3 will describe the Scrum principles in depth; subsequent chapters will highlight how specific practices and approaches are rooted in these principles and values.

The Scrum practices themselves are embodied in specific roles, activities, artifacts, and their associated rules (see Figure 2.1).

The remainder of this chapter will focus on Scrum practices.

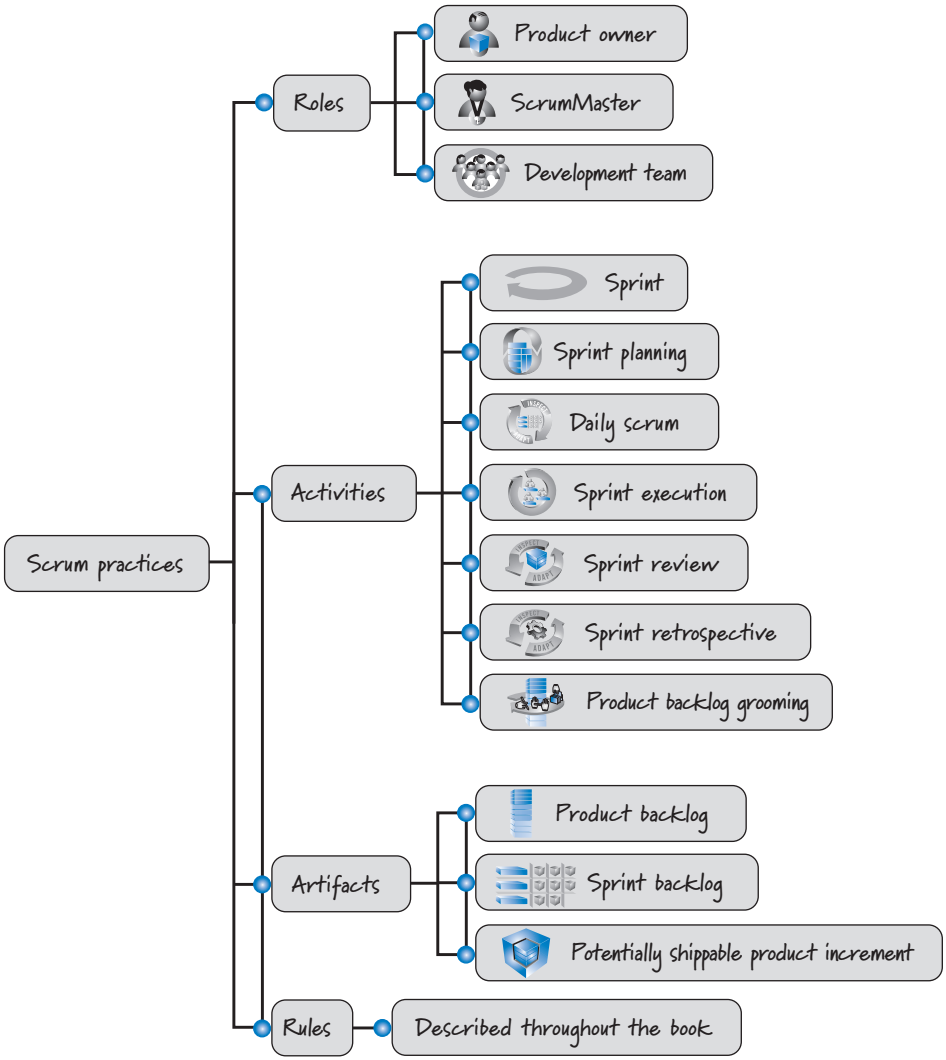

FIGURE 2.1 Scrum practices

Scrum Roles

Scrum development efforts consist of one or more **Scrum teams**, each made up of three Scrum roles: **product owner**, **ScrumMaster**, and the **development team** (see Figure 2.2). There can be other roles when using Scrum, but the Scrum framework requires only the three listed here.

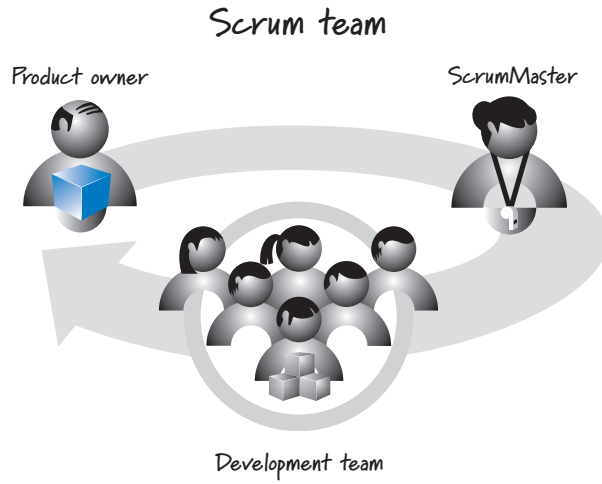

FIGURE 2.2 Scrum roles

The product owner is responsible for what will be developed and in what order. The ScrumMaster is responsible for guiding the team in creating and following its own process based on the broader Scrum framework. The development team is responsible for determining how to deliver what the product owner has asked for.

If you are a manager, don't be concerned that "manager" doesn't appear as a role in Figure 2.2; managers still have an important role in organizations that use Scrum (see Chapter 13). The Scrum framework defines just the roles that are specific to Scrum, not all of the roles that can and should exist within an organization that uses Scrum.

Product Owner

The product owner is the empowered central point of product leadership. He[1] is the single authority responsible for deciding which features and functionality to build and the order in which to build them. The product owner maintains and communicates to all other participants a clear vision of what the Scrum team is trying to achieve. As such, the product owner is responsible for the overall success of the solution being developed or maintained.

It doesn't matter if the focus is on an external product or an internal application; the product owner still has the obligation to make sure that the most valuable work possible, which can include technically focused work, is always performed. To

1. In this book the product owner will always be referred to as "he" or "him" and the ScrumMaster as "she" or "her." This is consistent with the visual representation of each role within the figures.

ensure that the team rapidly builds what the product owner wants, the product owner actively collaborates with the ScrumMaster and development team and must be available to answer questions soon after they are posed. See Chapter 9 for a detailed description of the product owner role.

ScrumMaster

The ScrumMaster helps everyone involved understand and embrace the Scrum values, principles, and practices. She acts as a coach, providing process leadership and helping the Scrum team and the rest of the organization develop their own high-performance, organization-specific Scrum approach. At the same time, the ScrumMaster helps the organization through the challenging change management process that can occur during a Scrum adoption.

As a facilitator, the ScrumMaster helps the team resolve issues and make improvements to its use of Scrum. She is also responsible for protecting the team from outside interference and takes a leadership role in removing **impediments** that inhibit team productivity (when the individuals themselves cannot reasonably resolve them). The ScrumMaster has no authority to exert control over the team, so this role is not the same as the traditional role of project manager or development manager. The ScrumMaster functions as a leader, not a manager. I will discuss the roles of functional manager and project manager in Chapter 13. See Chapter 10 for more details on the ScrumMaster role.

Development Team

Traditional software development approaches discuss various job types, such as architect, programmer, tester, database administrator, UI designer, and so on. Scrum defines the role of a development team, which is simply a diverse, cross-functional collection of these types of people who are responsible for designing, building, and testing the desired product.

The development team self-organizes to determine the best way to accomplish the goal set out by the product owner. The development team is typically five to nine people in size; its members must collectively have all of the skills needed to produce good-quality, working software. Of course, Scrum can be used on development efforts that require much larger teams. However, rather than having one Scrum team with, say, 35 people, there would more likely be four or more Scrum teams, each with a development team of nine or fewer people. See Chapter 11 for more details on the development team role and Chapter 12 for more details on coordinating multiple teams.

Scrum Activities and Artifacts

Figure 2.3 illustrates most of the Scrum activities and artifacts and how they fit together.

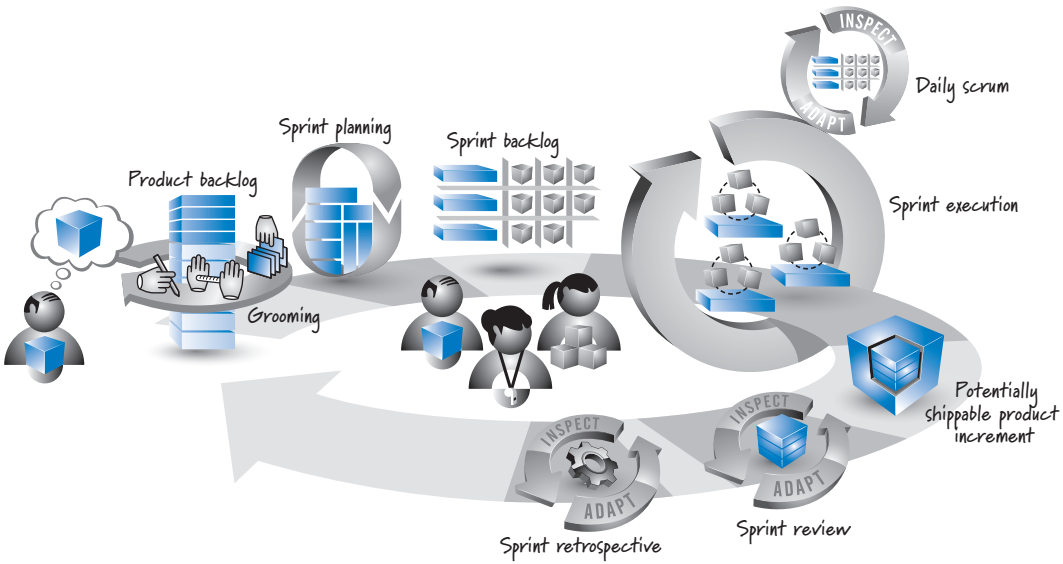

FIGURE 2.3 Scrum framework

Let's summarize the diagram, starting on the left side of the figure and working clockwise around the main looping arrow (the sprint).

The product owner has a vision of what he wants to create (the big cube). Because the cube can be large, through an activity called **grooming** it is broken down into a set of features that are collected into a prioritized list called the product backlog.

A sprint starts with sprint planning, encompasses the development work during the sprint (called sprint execution), and ends with the review and retrospective. The sprint is represented by the large, looping arrow that dominates the center of the figure. The number of items in the product backlog is likely to be more than a development team can complete in a short-duration sprint. For that reason, at the beginning of each sprint, the development team must determine a subset of the product backlog items it believes it can complete—an activity called sprint planning, shown just to the right of the large product backlog cube.

As a brief aside, in 2011 a change in "The Scrum Guide" (Schwaber and Sutherland 2011) generated debate about whether the appropriate term for describing the result of sprint planning is a **forecast** or a **commitment**. Advocates of the word *forecast* like it because they feel that although the development team is making the best estimate that it can at the time, the estimate might change as more information becomes known during the course of the sprint. Some also believe that a commitment on the part of the team will cause the team to sacrifice quality to meet the commitment or will cause the team to "under-commit" to guarantee that the commitment is met.

I agree that all development teams should generate a forecast (estimate) of what they can deliver each sprint. However, many development teams would benefit from

using the forecast to derive a commitment. Commitments support mutual trust between the product owner and the development team as well as within the development team. Also, commitments support reasonable short-term planning and decision making within an organization. And, when performing multiteam product development, commitments support synchronized planning—one team can make decisions based on what another team has committed to do. In this book, I favor the term *commitment*; however, I occasionally use *forecast* if it seems correct in context.

To acquire confidence that the development team has made a reasonable commitment, the team members create a second backlog during sprint planning, called the sprint backlog. The sprint backlog describes, through a set of detailed **tasks**, how the team plans to design, build, integrate, and test the selected subset of features from the product backlog during that particular sprint.

Next is sprint execution, where the development team performs the tasks necessary to realize the selected features. Each day during sprint execution, the team members help manage the flow of work by conducting a synchronization, inspection, and adaptive planning activity known as the daily scrum. At the end of sprint execution the team has produced a potentially shippable product increment that represents some, but not all, of the product owner's vision.

The Scrum team completes the sprint by performing two inspect-and-adapt activities. In the first, called the sprint review, the stakeholders and Scrum team inspect the product being built. In the second, called the sprint retrospective, the Scrum team inspects the Scrum process being used to create the product. The outcome of these activities might be adaptations that will make their way into the product backlog or be included as part of the team's development process.

At this point the Scrum sprint cycle repeats, beginning anew with the development team determining the next most important set of product backlog items it can complete. After an appropriate number of sprints have been completed, the product owner's vision will be realized and the solution can be released.

In the remainder of this chapter I will discuss each of these activities and artifacts in greater detail.

Product Backlog

Using Scrum, we always do the most valuable work first. The product owner, with input from the rest of the Scrum team and stakeholders, is ultimately responsible for determining and managing the sequence of this work and communicating it in the form of a prioritized (or ordered) list known as the **product backlog** (see Figure 2.4). On new-product development the product backlog items initially are features required to meet the product owner's vision. For ongoing product development, the product backlog might also contain new features, changes to existing features, defects needing repair, technical improvements, and so on.

The product owner collaborates with internal and external stakeholders to gather and define the product backlog items. He then ensures that product backlog items

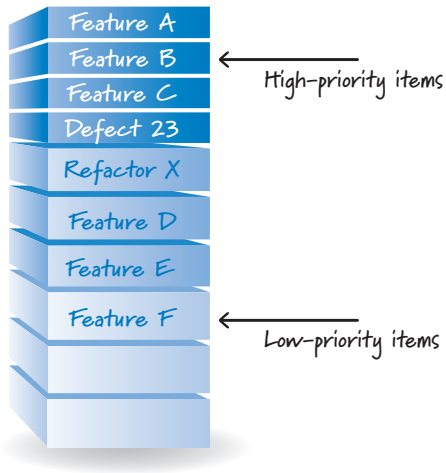

FIGURE 2.4 Product backlog

are placed in the correct sequence (using factors such as value, cost, knowledge, and risk) so that the high-value items appear at the top of the product backlog and the lower-value items appear toward the bottom. The product backlog is a constantly evolving artifact. Items can be added, deleted, and revised by the product owner as business conditions change, or as the Scrum team's understanding of the product grows (through feedback on the software produced during each sprint).

Overall the activity of creating and refining product backlog items, estimating them, and prioritizing them is known as grooming (see Figure 2.5).

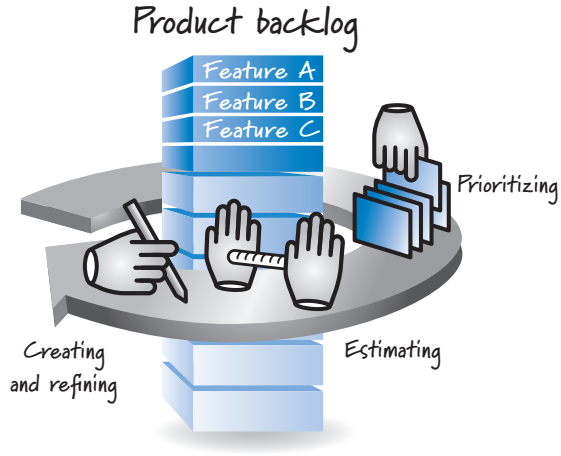

FIGURE 2.5 Product backlog grooming

As a second brief aside, in 2011 there was another debate as to whether the appropriate term for describing the sequence of items in the product backlog should be *prioritized* (the original term) or *ordered*, the term used in "The Scrum Guide" (Schwaber and Sutherland 2011). The argument was that prioritizing is simply one form of ordering (and, according to some, not even the most appropriate form of ordering). The issue of how to best sequence items in the product backlog, however, is influenced by many factors, and a single word may never capture the full breadth and depth of the concept. Although there may be theoretical merit to the ordered-versus-prioritized debate, most people (including me) use the terms interchangeably when discussing the items in the product backlog.

Before we finalize prioritizing, ordering, or otherwise arranging the product backlog, we need to know the size of each item in the product backlog (see Figure 2.6).

Size equates to cost, and product owners need to know an item's cost to properly determine its priority. Scrum does not dictate which, if any, size measure to use with product backlog items. In practice, many teams use a **relative size measure** such as **story points** or **ideal days**. A relative size measure expresses the overall size of an item in such a way that the absolute value is not considered, but the relative size of an item compared to other items is considered. For example, in Figure 2.6, feature C is size 2 and feature E is size 8. What we can conclude is that feature E is about four times larger than feature C. I will discuss these measures further in Chapter 7.

Sprints

In Scrum, work is performed in iterations or cycles of up to a calendar month called **sprints** (see Figure 2.7). The work completed in each sprint should create something of tangible value to the customer or user.

Sprints are **timeboxed** so they always have a fixed start and end date, and generally they should all be of the same duration. A new sprint immediately follows the completion of the previous sprint. As a rule we do not permit any goal-altering changes in scope or personnel during a sprint; however, business needs sometimes make adherence to this rule impossible. I will describe sprints in more detail in Chapter 4.

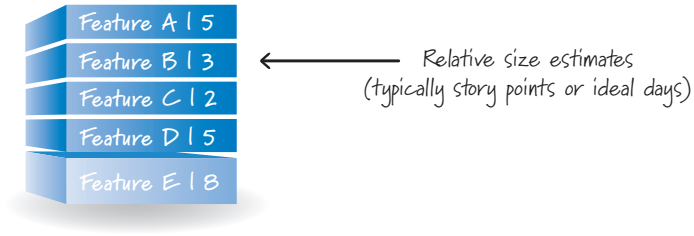

FIGURE 2.6 Product backlog item sizes

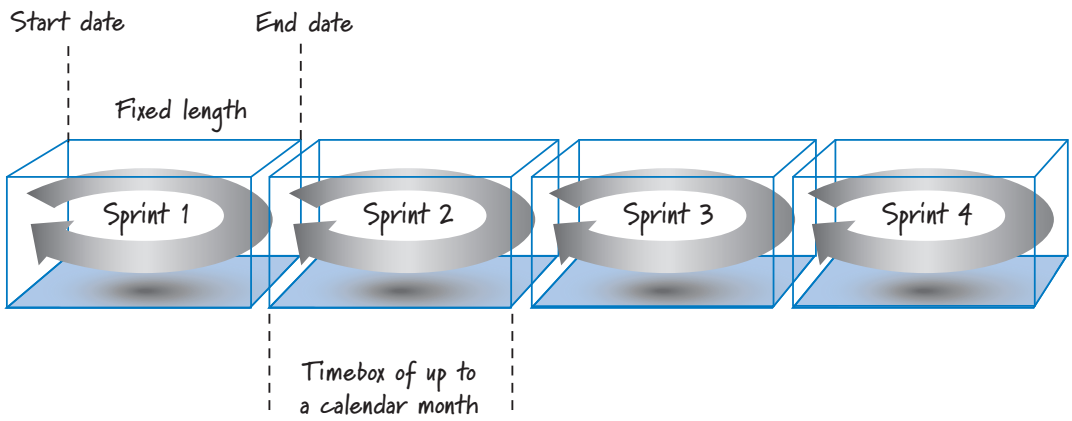

FIGURE 2.7 Sprint characteristics

Sprint Planning

A product backlog may represent many weeks or months of work, which is much more than can be completed in a single, short sprint. To determine the most important subset of product backlog items to build in the next sprint, the product owner, development team, and ScrumMaster perform **sprint planning** (see Figure 2.8).

During sprint planning, the product owner and development team agree on a **sprint goal** that defines what the upcoming sprint is supposed to achieve. Using this

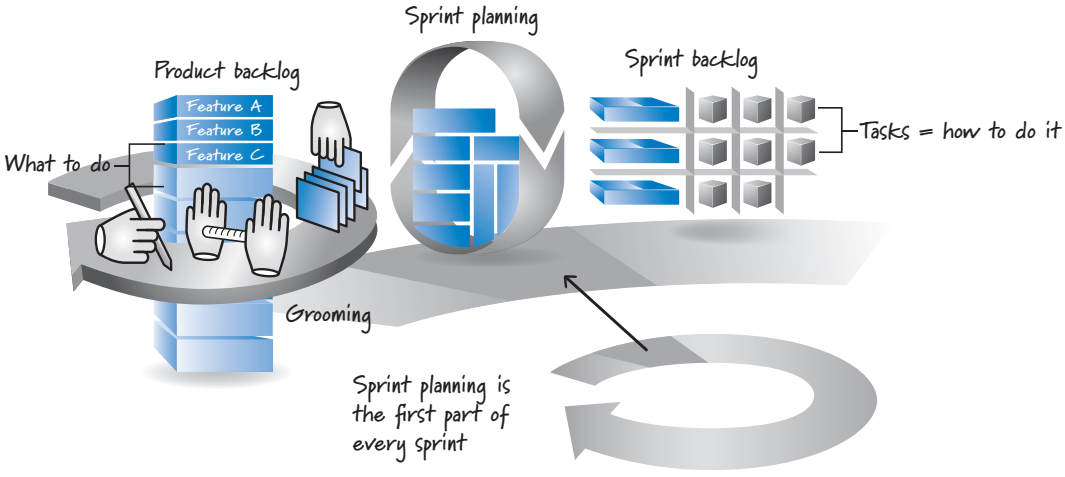

FIGURE 2.8 Sprint planning

goal, the development team reviews the product backlog and determines the high-priority items that the team can realistically accomplish in the upcoming sprint while working at a **sustainable pace**—a pace at which the development team can comfortably work for an extended period of time.

To acquire confidence in what it can get done, many development teams break down each targeted feature into a set of tasks. The collection of these tasks, along with their associated product backlog items, forms a second backlog called the **sprint backlog** (see Figure 2.9).

The development team then provides an estimate (typically in hours) of the effort required to complete each task. Breaking product backlog items into tasks is a form of design and **just-in-time** planning for how to get the features done.

Most Scrum teams performing sprints of two weeks to a month in duration try to complete sprint planning in about four to eight hours. A one-week sprint should take no more than a couple of hours to plan (and probably less). During this time there are several approaches that can be used. The approach I use most often follows a simple cycle: Select a product backlog item (whenever possible, the next-most-important item as defined by the product owner), break the item down into tasks, and determine if the selected item will reasonably fit within the sprint (in combination with other items targeted for the same sprint). If it does fit and there is more capacity to complete work, repeat the cycle until the team is out of capacity to do any more work.

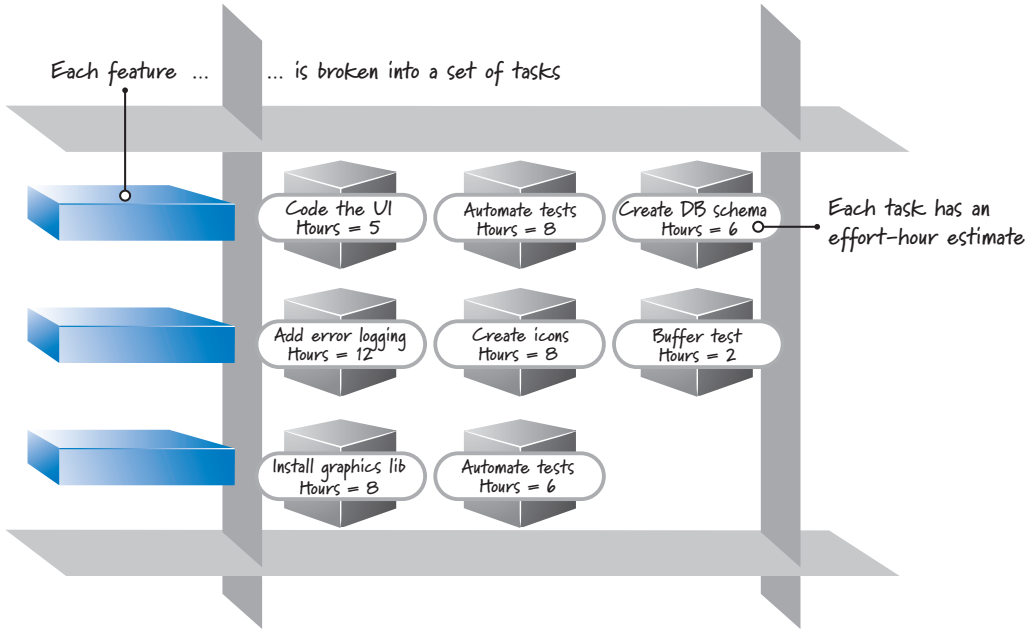

FIGURE 2.9 Sprint backlog

An alternative approach would be for the product owner and team to select all of the target product backlog items at one time. The development team alone does the task breakdowns to confirm that it really can deliver all of the selected product backlog items. I will describe each approach in more detail in Chapter 19.

Sprint Execution

Once the Scrum team finishes sprint planning and agrees on the content of the next sprint, the development team, guided by the ScrumMaster's coaching, performs all of the task-level work necessary to get the features done (see Figure 2.10), where "done" means there is a high degree of confidence that all of the work necessary for producing good-quality features has been completed.

Exactly what tasks the team performs depends of course on the nature of the work (for example, are we building software and what type of software, or are we building hardware, or is this marketing work?).

Nobody tells the development team in what order or how to do the task-level work in the sprint backlog. Instead, team members define their own task-level work and then self-organize in any manner they feel is best for achieving the sprint goal. See Chapter 20 for more details on sprint execution.

Daily Scrum

Each day of the sprint, ideally at the same time, the development team members hold a timeboxed (15 minutes or less) **daily scrum** (see Figure 2.11). This inspect-and-adapt activity is sometimes referred to as the **daily stand-up** because of the common practice of everyone standing up during the meeting to help promote brevity.

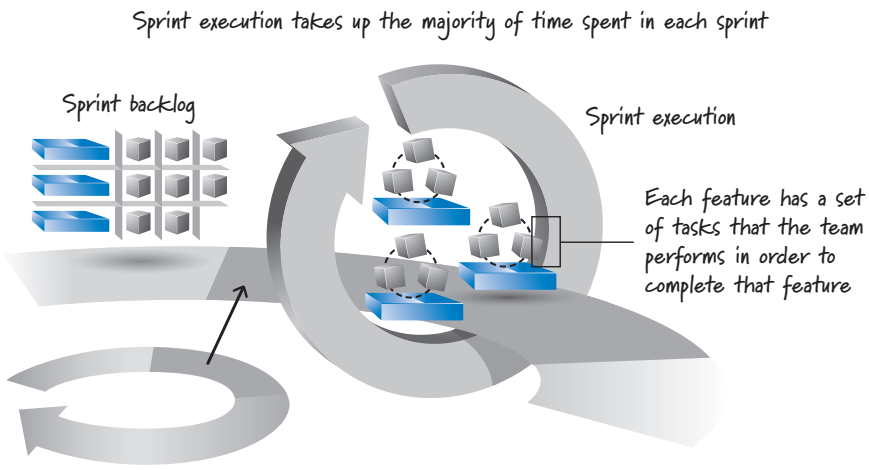

FIGURE 2.10 Sprint execution

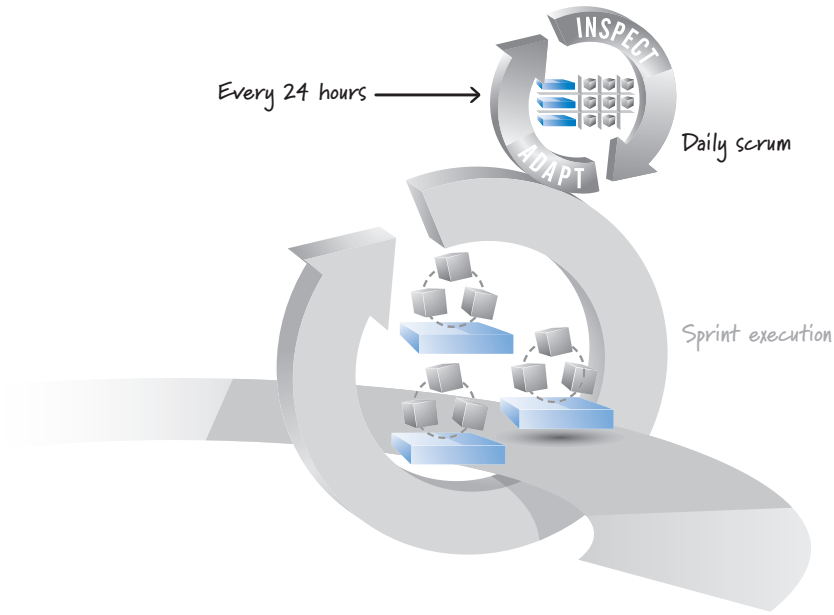

FIGURE 2.11 Daily scrum

A common approach to performing the daily scrum has the ScrumMaster facilitating and each team member taking turns answering three questions for the benefit of the other team members:

- What did I accomplish since the last daily scrum?
- What do I plan to work on by the next daily scrum?
- What are the obstacles or impediments that are preventing me from making progress?

By answering these questions, everyone understands the big picture of what is occurring, how they are progressing toward the sprint goal, any modifications they want to make to their plans for the upcoming day's work, and what issues need to be addressed. The daily scrum is essential for helping the development team manage the fast, flexible flow of work within a sprint.

The daily scrum is not a problem-solving activity. Rather, many teams decide to talk about problems after the daily scrum and do so with a small group of interested people. The daily scrum also is not a traditional status meeting, especially the kind historically called by project managers so that they can get an update on the project's status. A daily scrum, however, can be useful to communicate the status of sprint backlog items among the development team members. Mainly, the daily scrum is an inspection, synchronization, and adaptive daily planning activity that helps a self-organizing team do its job better.

Although their use has fallen out of favor, Scrum has used the terms "**pigs**" and "**chickens**" to distinguish who should participate during the daily scrum versus who simply observes. The farm animals were borrowed from an old joke (which has several variants): "In a ham-and-eggs breakfast, the chicken is involved, but the pig is committed." Obviously the intent of using these terms in Scrum is to distinguish between those who are involved (the chickens) and those who are committed to meeting the sprint goal (the pigs). At the daily scrum, only the pigs should talk; the chickens, if any, should attend as observers.

I have found it most useful to consider everyone on the Scrum team a pig and anyone who isn't, a chicken. Not everyone agrees. For example, the product owner is not required to be at the daily scrum, so some consider him to be a chicken (the logic being, how can you be "committed" if you aren't required to attend?). This seems wrong to me, because I can't imagine how the product owner, as a member of the Scrum team, is any less committed to the outcome of a sprint than the development team. The metaphor of pigs and chickens breaks down if you try to apply it within a Scrum team.

Done

In Scrum, we refer to the sprint results as a **potentially shippable product increment** (see Figure 2.12), meaning that whatever the Scrum team agreed to do is really done according to its agreed-upon definition of done. This definition specifies the degree

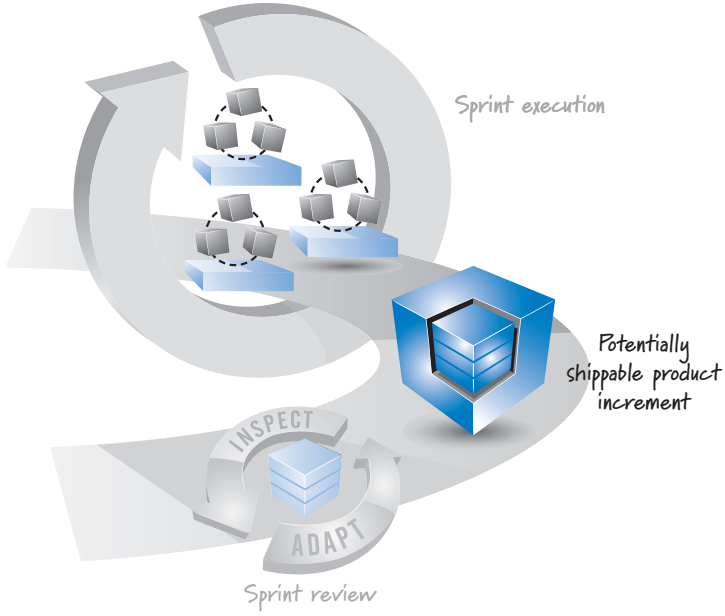

FIGURE 2.12 Sprint results (potentially shippable product increment)

of confidence that the work completed is of good quality and is potentially shippable. For example, when developing software, a bare-minimum definition of done should yield a complete slice of product functionality that is designed, built, integrated, tested, and documented.

An aggressive definition of done enables the business to decide each sprint if it wants to ship (or deploy or release) what got built to internal or external customers.

To be clear, "potentially shippable" does not mean that what got built must actually be shipped. Shipping is a business decision, which is frequently influenced by things such as "Do we have enough features or enough of a customer workflow to justify a customer deployment?" or "Can our customers absorb another change given that we just gave them a release two weeks ago?"

Potentially shippable is better thought of as a state of confidence that what got built in the sprint is actually done, meaning that there isn't materially important undone work (such as important testing or integration and so on) that needs to be completed before we can ship the results from the sprint, if shipping is our business desire.

As a practical matter, over time some teams may vary the definition of done. For example, in the early stages of game development, having features that are potentially shippable might not be economically feasible or desirable (given the exploratory nature of early game development). In these situations, an appropriate definition of done might be a slice of product functionality that is sufficiently functional and usable to generate feedback that enables the team to decide what work should be done next or how to do it. See Chapter 4 for more details on the definition of done.

Sprint Review

At the end of the sprint there are two additional inspect-and-adapt activities. One is called the **sprint review** (see Figure 2.13).

The goal of this activity is to inspect and adapt the product that is being built. Critical to this activity is the conversation that takes place among its participants, which include the Scrum team, stakeholders, sponsors, customers, and interested members of other teams. The conversation is focused on reviewing the just-completed features in the context of the overall development effort. Everyone in attendance gets clear visibility into what is occurring and has an opportunity to help guide the forthcoming development to ensure that the most business-appropriate solution is created.

A successful review results in bidirectional information flow. The people who aren't on the Scrum team get to sync up on the development effort and help guide its direction. At the same time, the Scrum team members gain a deeper appreciation for the business and marketing side of their product by getting frequent feedback on the convergence of the product toward delighted customers or users. The sprint review therefore represents a scheduled opportunity to inspect and adapt the product. As a

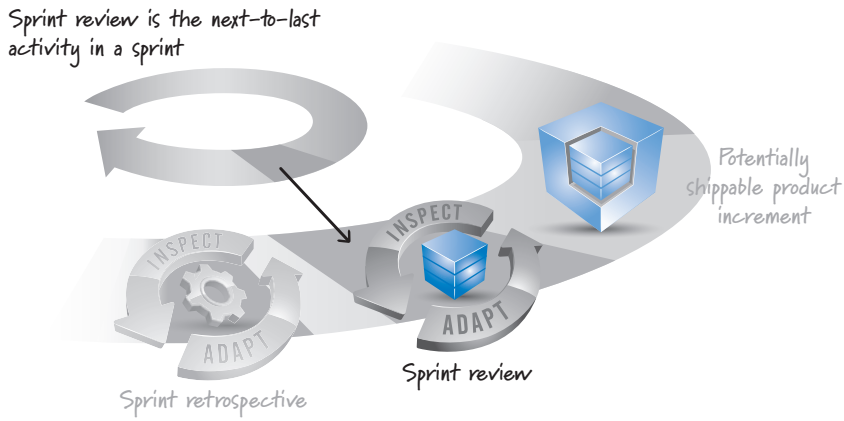

FIGURE 2.13 Sprint review

matter of practice, people outside the Scrum team can perform intra-sprint feature reviews and provide feedback to help the Scrum team better achieve its sprint goal. See Chapter 21 for more details on the sprint review.

Sprint Retrospective

The second inspect-and-adapt activity at the end of the sprint is the **sprint retrospective** (see Figure 2.14). This activity frequently occurs after the sprint review and before the next sprint planning.

Whereas the sprint review is a time to inspect and adapt the product, the sprint retrospective is an opportunity to inspect and adapt the process. During the sprint retrospective the development team, ScrumMaster, and product owner come together

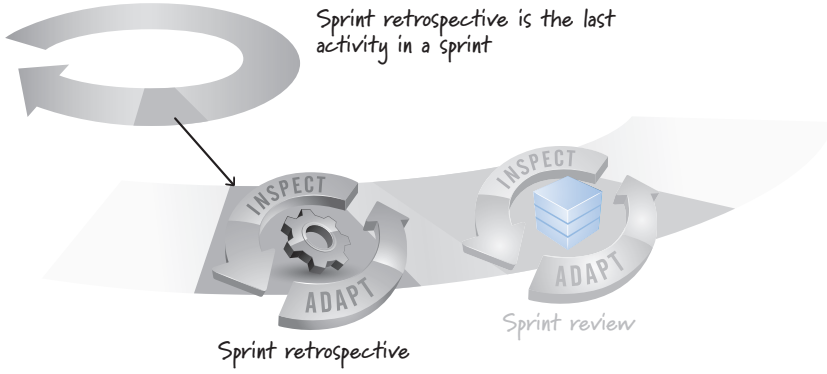

FIGURE 2.14 Sprint retrospective

to discuss what is and is not working with Scrum and associated technical practices. The focus is on the continuous process improvement necessary to help a good Scrum team become great. At the end of a sprint retrospective the Scrum team should have identified and committed to a practical number of process improvement actions that will be undertaken by the Scrum team in the next sprint. See Chapter 22 for details on the sprint retrospective.

After the sprint retrospective is completed, the whole cycle is repeated again—starting with the next sprint-planning session, held to determine the current highest-value set of work for the team to focus on.

Closing

This chapter described core Scrum practices, focusing on an end-to-end description of the Scrum framework's roles, activities, and artifacts. There are other practices, such as higher-level planning and progress-tracking practices, that many Scrum teams use. These will be described in subsequent chapters. In the next chapter, I will provide a description of the core principles on which Scrum is based. This will facilitate the deeper exploration of the Scrum framework in subsequent chapters.

What Are User Stories?

User stories are a convenient format for expressing the desired business value for many types of product backlog items, especially features. User stories are crafted in a way that makes them understandable to both business people and technical people. They are structurally simple and provide a great placeholder for a conversation. Additionally, they can be written at various levels of granularity and are easy to progressively refine.

As well adapted to our needs as user stories might be, I don't consider them to be the only way to represent product backlog items. They are simply a lightweight approach that dovetails nicely with core agile principles and our need for an efficient and effective placeholder. I use them as the central placeholder to which I will attach any other information that I think is relevant and helpful for detailing a requirement. If I find that user stories are a forced fit for a particular situation (such as representing certain defects), I'll use another approach. For example, I once saw a team write the following user story: "As a customer I would like the system to not corrupt the database." I think we can all agree that a user story is probably not the best way to represent this issue. Perhaps a simple reference to the defect in the defect-tracking system would be more appropriate.

So what exactly are user stories? Ron Jeffries offers a simple yet effective way to think about user stories (Jeffries 2001). He describes them as the three Cs: card, conversation, and confirmation.

Card

The card idea is pretty simple. People originally wrote (and many still do) user stories directly on 3 × 5-inch index cards or sticky notes (see Figure 5.2).

A common template format for writing user stories (as shown on the left in Figure 5.2) is to specify a class of users (the user role), what that class of users wants to achieve (the goal), and why the users want to achieve the goal (the benefit) (Cohn 2004). The "so that" part of a user story is optional, but unless the purpose of the

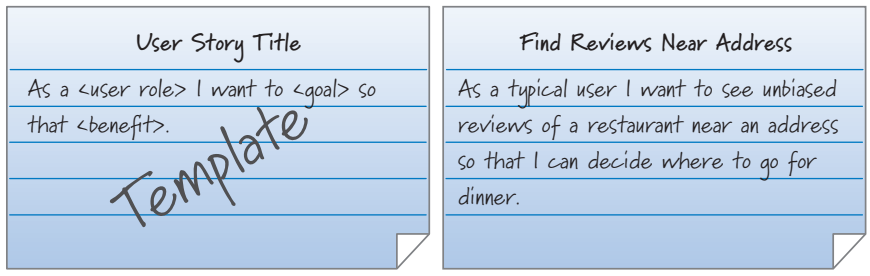

FIGURE 5.2 A user story template and card

story is completely obvious to everyone, we should include it with every user story. The right side of Figure 5.2 shows an example of a user story based on this template.

The card isn't intended to capture all of the information that makes up the requirement. In fact, we deliberately use small cards with limited space to promote brevity. A card should hold a few sentences that capture the essence or intent of a requirement. It serves as the placeholder for more detailed discussions that will take place among the stakeholders, product owner, and development team.

Conversation

The details of a requirement are exposed and communicated in a conversation among the development team, product owner, and stakeholders. The user story is simply a promise to have that conversation.

I say "that conversation," but in actuality, the conversation is typically not a one-time event, but rather an ongoing dialogue. There can be an initial conversation when the user story is written, another conversation when it's refined, yet another when it's estimated, another during sprint planning (when the team is diving into the task-level details), and finally, ongoing conversations while the user story is being designed, built, and tested during the sprint.

One of the benefits of user stories is that they shift some of the focus away from writing and onto conversations. These conversations enable a richer form of exchanging information and collaborating to ensure that the correct requirements are expressed and understood by everyone.

Although conversations are largely verbal, they can be and frequently are supplemented with documents. Conversations may lead to a UI sketch, or an elaboration of business rules that gets written down. For example, I visited an organization that was developing medical imaging software. One of its stories is shown in Figure 5.3.

Notice that the user story references an entire article for future reading and conversation.

So we're not tossing out all of our documents in favor of user stories and their associated story cards. User stories are simply a good starting point for eliciting the initial essence of what is desired, and for providing a reminder to discuss

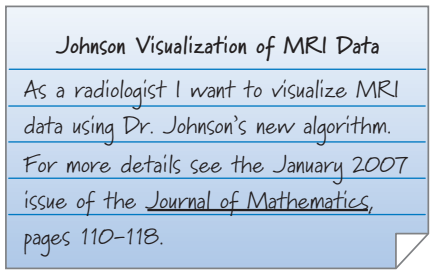

FIGURE 5.3 User story with additional data attached

requirements in more detail when appropriate. However, user stories can and should be supplemented with whatever other written information helps provide clarity regarding what is desired.

Confirmation

A user story also contains confirmation information in the form of conditions of satisfaction. These are acceptance criteria that clarify the desired behavior. They are used by the development team to better understand what to build and test and by the product owner to confirm that the user story has been implemented to his satisfaction.

If the front of the card has a few-line description of the story, the back of the card could specify the conditions of satisfaction (see Figure 5.4).

These conditions of satisfaction can be expressed as high-level acceptance tests. However, these tests would not be the only tests that are run when the story is being developed. In fact, for the handful of acceptance tests that are associated with a user story, the team will have many more tests (perhaps 10 to 100 times more) at a detailed technical level that the product owner doesn't even know about.

The acceptance tests associated with the story exist for several reasons. First, they are an important way to capture and communicate, from the product owner's perspective, how to determine if the story has been implemented correctly.

These tests can also be a helpful way to create initial stories and refine them as more details become known. This approach is sometimes called **specification by example** or **acceptance-test-driven development** (**ATTD**). The idea is fairly intuitive. Discussions about the stories can and frequently do focus on defining specific examples or desired behaviors. For example, in the "Upload File" story in Figure 5.4, the conversation likely went something like this:

> Initially, let's limit uploaded file sizes to be 1 GB or less. Also, make sure that we can properly load common text and graphics files. And for legal reasons we can't have any files with digital rights management (DRM) restrictions loaded to the wiki.

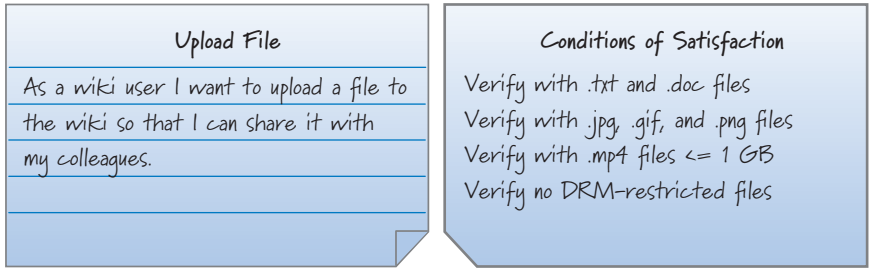

FIGURE 5.4 User story conditions of satisfaction

TABLE 5.1 Automated Test Example

Size	Valid()
0	True
1,073,741,824	True
1,073,741,825	False

If we were using a tool like Fit or FitNesse, we could conveniently define these tests in a table like Table 5.1, which shows examples of different file sizes and whether or not they are valid.

By elaborating on specific examples like these, we can drive the story creation and refinement process and have (automated) acceptance tests available for each story.

Level of Detail

User stories are an excellent vehicle for carrying items of customer or user value through the Scrum value-creation flow. However, if we have only one story size (the size that would comfortably fit within a short-duration sprint), it will be difficult to do higher-level planning and to reap the benefits of progressive refinement.

Small stories used at the sprint level are too small and too numerous to support higher-level product and release planning. At these levels we need fewer, less detailed, more abstract items. Otherwise, we'll be mired in a swamp of mostly irrelevant detail. Imagine having 500 very small stories and being asked to provide an executive-level description of the proposed product to secure your funding. Or try to prioritize among those 500 really small items to define the next release.

Also, if there is only one (small) size of story, we will be obligated to define all requirements at a very fine-grained level of detail long before we should. Having only small stories precludes the benefit of progressively refining requirements on a just-enough, just-in-time basis.

Fortunately, user stories can be written to capture customer and user needs at various levels of abstraction (see Figure 5.5).

Figure 5.5 depicts stories at multiple levels of abstraction. The largest would be stories that are a few to many months in size and might span an entire release or multiple releases. Many people refer to these as **epics**, alluding to the idea that they are *Lord of the Rings* or *War and Peace* size stories. Epics are helpful because they give a very big-picture, high-level overview of what is desired (see Figure 5.6).

We would never move an epic into a sprint for development because it is way too big and not very detailed. Instead, epics are excellent placeholders for a large

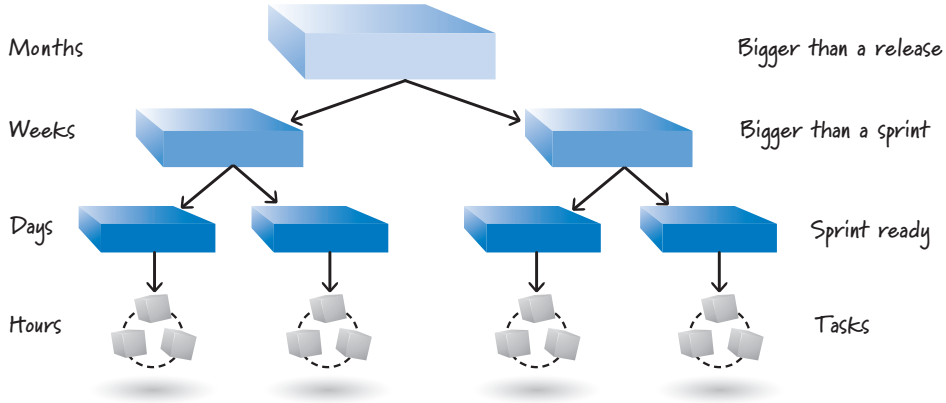

FIGURE 5.5 User story abstraction hierarchy

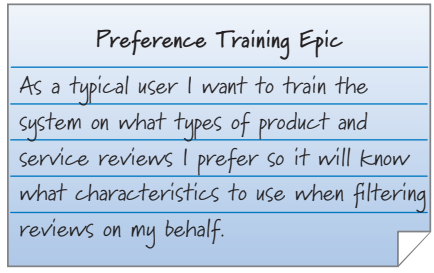

FIGURE 5.6 Example epic

collection of more detailed stories to be created at an appropriate future time. I will illustrate the use of epics during the discussion of product planning in Chapter 17.

The next-size stories in Figure 5.5 are those that are often on the order of weeks in size and therefore too big for a single sprint. Some teams might call these **features**.

The smallest forms of user stories are those I typically refer to as **stories**. To avoid any confusion with epics, features, or other larger items, which are also "stories," some people call these stories either **sprintable stories** or **implementable stories** to indicate that they are on the order of days in size and therefore small enough to fit into a sprint and be implemented. Figure 5.2 provides an example of a sprintable story.

Some teams also use the term **theme** to refer to a collection of related stories. Themes provide a convenient way to say that a bunch of stories have something in common, such as being in the same functional area. In Figure 5.7, the theme represents the collection of stories that will provide the details of how to perform keyword training.

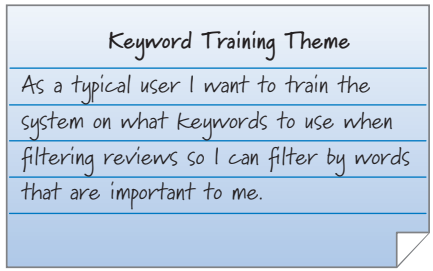

FIGURE 5.7 Example theme

I often think of a theme as the summary card for a bunch of note cards stacked together with a rubber band around them to indicate that they are similar to one another in an area that we think is important.

Tasks are the layer below stories, typically worked on by only one person, or perhaps a pair of people. Tasks typically require hours to perform. When we go to the task layer, we are specifying *how* to build something instead of *what* to build (represented by epics, features, and stories). Tasks are not stories, so we should avoid including task-level detail when writing stories.

It is important to keep in mind that terms like *epic*, *feature*, *story*, and *theme* are just labels of convenience, and they are not universally shared. It really doesn't matter what labels you use as long as you use them consistently. What does matter is recognizing that stories can exist at multiple levels of abstraction, and that doing so nicely supports our efforts to plan at multiple levels of abstraction and to progressively refine big items into small items over time.

INVEST in Good Stories

How do we know if the stories that we have written are good stories? Bill Wake has offered six criteria (summarized by the acronym INVEST) that have proved useful when evaluating whether our stories are fit for their intended use or require some additional work (Wake 2003).

The **INVEST** criteria are ***I****ndependent*, ***N****egotiable*, ***V****aluable*, ***E****stimatable*, ***S****mall* (sized appropriately), and ***T****estable*. When we combine the information derived from applying each criterion, we get a clear picture of what, if any, additional changes we might want to make to a story. Let's examine each criterion.

Independent

As much as is practical, user stories should be *independent* or at least only loosely coupled with one another. Stories that exhibit a high degree of interdependence complicate estimating, prioritizing, and planning. For example, on the left side of Figure 5.8, story #10 depends on many other stories.

On this task board each product backlog item planned to be worked on during the sprint is shown with the set of tasks necessary to get the item done. All tasks initially start off in the "to do" column. Once the team determines that it is appropriate to work on an item, team members start selecting tasks in the "to do" column for the item and move them into the "in progress" column to indicate that work on those tasks has begun. When a task is completed, it is moved to the "completed" column.

Of course, Figure 20.6 is just an example of how a task board might be structured. A team may choose to put other columns on its task board if it thinks that visualizing the flow of work through other states is helpful. In fact, an alternative agile approach called Kanban (Anderson 2010) uses just such a detailed board to visualize the flow of work through its various stages.

Sprint Burndown Chart

Each day during sprint execution team members update the estimate of how much effort remains for each uncompleted task. We could create a table to visualize this data. Table 20.1 shows an example of a 15-day sprint that initially has 30 tasks (not all of the days and tasks are shown in the table).

In Table 20.1 the number of hours remaining for each task follows the general trend of being smaller each day during the sprint—because tasks are being worked on and completed. If a task hasn't yet been started (it is still in the task board "to do" column), the size of the task might appear the same from day to day until the task is started. Of course, a task might turn out to be larger than expected, and if so, its size may actually increase day over day (see Table 20.1, task 4, days 4 and 5) or remain the same size even after the team has started working it (see Table 20.1 task 1, days 2 and 3)—either

TABLE 20.1 Sprint Backlog with Estimated Effort Remaining Each Day

Tasks	D1	D2	D3	D4	D5	D6	D7	D8	D9	. . .	D15
Task 1	8	4	4	2							
Task 2	12	8	16	14	9	6	2				
Task 3	5	5	3	3	1						
Task 4	7	7	7	5	10	6	3	1			
Task 5	3	3	3	3	3	3	3				
Task 6	14	14	14	14	14	14	14	8	4		
Task 7						8	6	4	2		
Tasks 8–30	151	139	143	134	118	99	89	101	84		0
Total	200	180	190	175	155	130	115	113	90		0

because no work took place on the task the previous day, or work did take place the previous day but the estimated effort remaining is the same.

New tasks related to the committed product backlog items can also be added to the sprint backlog at any time. For example, on day 6 in Table 20.1 the team discovered that task 7 was missing, so it added it. There is no reason to avoid adding a task to the sprint backlog. It represents real work that the team must do to complete a product backlog item that the team agreed to get done. Permitting unforeseen tasks to be added to the sprint backlog is not a loophole for introducing new work into the sprint. It simply acknowledges that during sprint planning we may not be able to fully define the complete set of tasks needed to design, build, integrate, and test the committed product backlog items. As our understanding of the work improves by doing it, we can and should adjust the sprint backlog.

If we plot the row labeled "Total" in Table 20.1, which is the sum of the remaining effort-hours across all uncompleted tasks on a given day, on a graph, we get another of the Scrum artifacts for communicating progress—the sprint burndown chart (see Figure 20.7).

In Chapter 18 I discussed release burndown charts, where the vertical axis numbers are either in story points or ideal days and the horizontal axis numbers are in sprints (see Figure 18.11). In sprint burndown charts the vertical axis numbers are the estimated effort-hours remaining, and the horizontal axis numbers are days within a sprint. Figure 20.7 shows that we have 200 estimated effort-hours remaining on the first day of the sprint and zero effort-hours remaining on day 15 (the last day of a three-week-long sprint). Each day we update this chart to show the total estimated effort remaining across all of the uncompleted tasks.

Like release burndown charts, sprint burndown charts are useful for tracking progress and can also be used as a leading indicator to predict when work will be

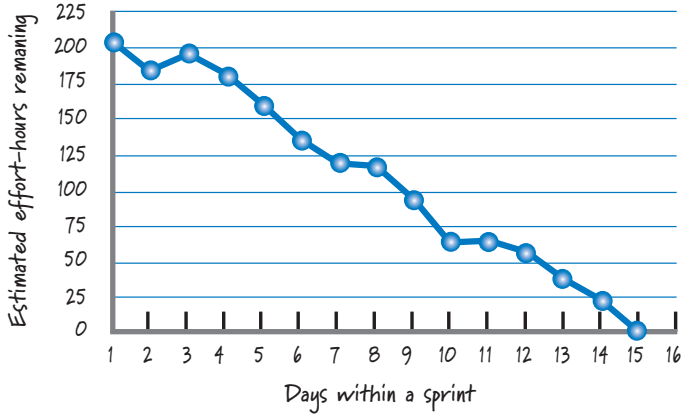

FIGURE 20.7 Sprint burndown chart

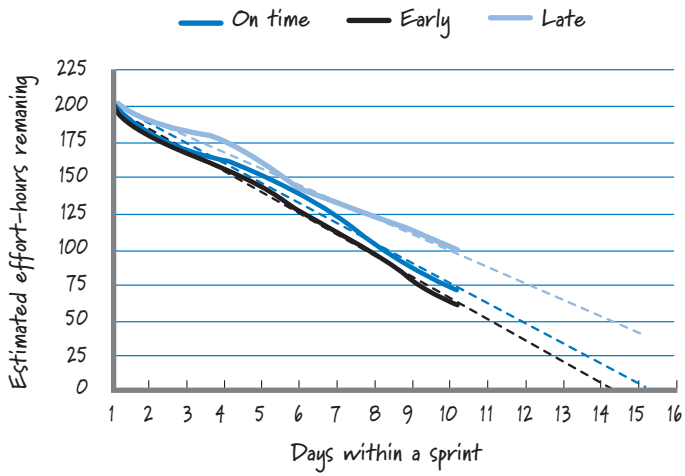

FIGURE 20.8 Sprint burndown chart with trend lines

completed. At any point in time we could compute a trend line based on historical data and use that trend line to see when we are likely to finish if the current pace and scope remain constant (see Figure 20.8).

In this figure, three different burndown lines are superimposed to illustrate distinct situations. When the trend line intersects the horizontal axis close to the end of the sprint duration, we can infer that we're in reasonable shape ("On time"). When it lands significantly to the left, we should probably take a look to see if we can safely take on additional work ("Early"). But when it lands significantly to the right ("Late"), that raises a flag that we're not proceeding at the expected pace or that we've taken on too much work (or both!). When that happens, we should dig deeper to see what's behind the data and what, if anything, needs to be done. By projecting the trend lines, we have another important set of data that adds to our knowledge of how we are managing flow within our sprint.

The sprint backlog and the sprint burndown charts always use estimated effort *remaining*. They do not capture actual effort expended. In Scrum there is no specific need to capture the *actuals*; however, your organization might choose to do so for non-Scrum reasons such as cost accounting or tax purposes.

Sprint Burnup Chart

Analogous to how a release burnup chart is an alternative way of visualizing progress through a release, a sprint burnup chart is an alternative way to visualize progress through a sprint. Both represent the amount of work completed toward achieving a goal, the release goal in one case and the sprint goal in the other.

Figure 20.9 shows an example sprint burnup chart.