Scrum

Introduction

There are several agile methods, some more popular than others. In this chapter we focus on the most popular agile method: Scrum. Part of the reason for its popularity is that Scrum is a process framework rather than a detailed process specification, which makes it more adaptable to the needs and preferences of different groups. Even though there is quite a lot of cross-fertilization among agile methods, with good ideas from one being adopted by the others, the generality of Scrum makes adding ideas from other agile methods especially easy. In this section we will consider Scrum and also mention some of the practices from other agile methods that Scrum practitioners frequently adopt.

Some ideas behind Scrum go back to an article in the Harvard Business Review about what is now called *concurrent product development* [1]. These ideas were transmogrified and made more applicable to software development by Jeff Sutherland in 1993. The first Scrum paper appeared in 1995 [2], and the seminal book *Agile Software Development with Scrum* appeared in 2001 [3]. Since then Scrum has become more and more popular. The current version of Scrum is documented online in the *Scrum Guide* [4], and in various books, such as *Essential Scrum* by Kenneth Rubin [5]. You are encouraged to consult these (or other) resources for more information about Scrum.

Scrum Overview

Scrum specifies a high level *process* consisting of *activities* that create and modify certain *artifacts*. The work is divided among developers playing certain *roles*. We thus begin by discussing Scrum's process, activities, artifacts, and roles.

Process

The Scrum process is pictured in Figure 1 below. This process contains the activities and artifacts discussed subsequently, so you will have to refer back to this diagram as you read the remainder of the overview. The process begins with the creation of an initial product backlog, which contains specifications for the target product. During sprint planning, a subset of the product backlog is selected for implementation, the tasks needed to do this are identified, and the effort to do them estimated. This information together forms the sprint backlog. The items in the sprint backlog are implemented during sprint execution, resulting in a potentially shippable product. At the end of the sprint, a sprint review demonstrates the new product features to stakeholders, and the sprint retrospective provides the team with a chance to look back over the sprint and decide how to do the next one better. Then this cycle, which is called a **sprint**, repeats. Because the cycle repeats, Scrum is an *iterative* process. More importantly, since a shippable product is produced in each cycle, Scrum is also an *incremental* process.

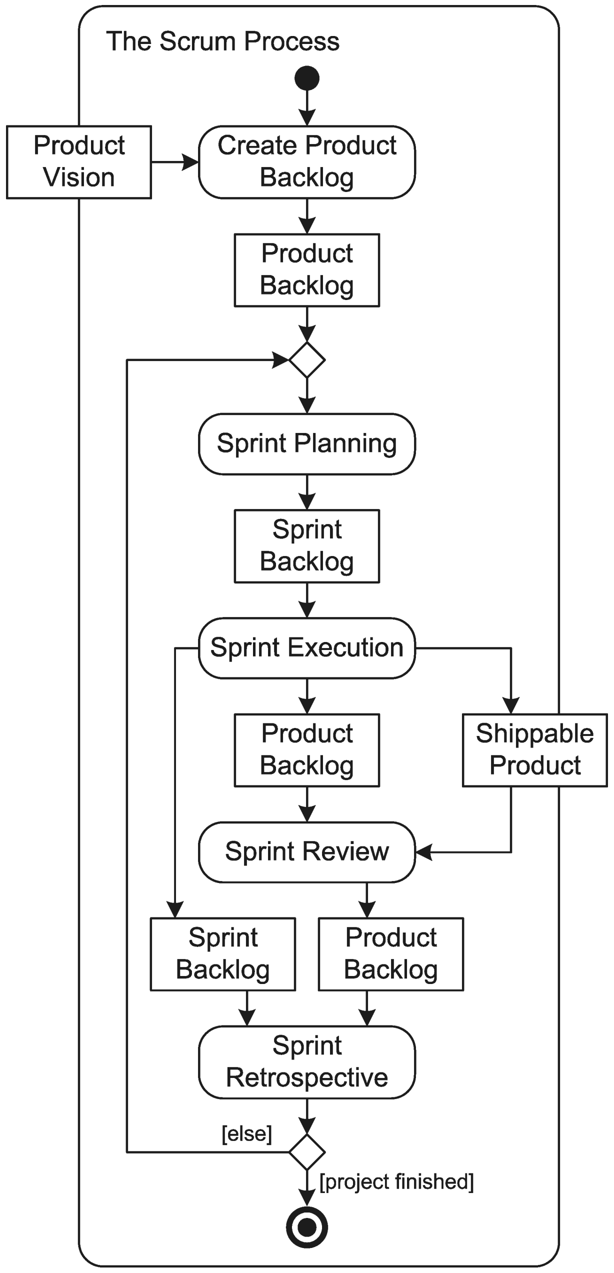

Figure 1: The Scrum Process

Roles

A Scrum project has at least one development team consisting of people playing the roles of *product owner*, *scrum master* and *team members*.

The **product owner** (PO) is responsible for deciding what will be done and in what order. The PO is in charge of the composition of the product, and as such represents the customers and users to the other members of the development team. Also, the PO collaborates with the scrum master and team members to make sure that the right features appear in the product in the right order as quickly as possible. Thus the PO is the pivotal individual mediating between the team and the other stakeholders in the project. The PO's responsibility is largely exercised through control of the product backlog.

The **scrum master** (SM) is responsible for guiding the team in following the Scrum development process. The SM helps everyone embrace Scrum values, principles and practices and use them properly. The SM is a facilitator and coach. Also the SM helps remove impediments and protects the team from being interfered with by people outside the team.

The **team members** are responsible for deciding how to build the product and for building it. Team members must have all the skills necessary to build the product. They are thus a cross-functional team. The team decides for itself how to organize to get the job done. A common Scrum practice is to have everyone work on everything so as to maximize productivity and to help team members develop new skills.

Artifacts

There are three main artifacts in Scrum: the product backlog, sprint backlogs, and the shippable product.

The **product backlog** is a prioritized list of not-yet-implemented product features or characteristics. Its elements are **product backlog items** (**PBIs**), each of which is a product feature or function, a product change or fix, a technical improvement, and so forth. The PO is responsible for the product backlog; PBI priorities are typically based on the business value of the feature described in the PBI.

A **sprint backlog** is a collection of PBIs, the tasks needed to complete them, and estimates of how much effort each task will require. The PO and the rest of the team choose items from the product backlog to be implemented in a sprint during sprint planning. The sprint backlog guides the team in its work during sprint execution.

The overarching goal of every sprint is to produce a **potentially shippable product**, that is, a product increment that could actually be shipped to customers. Usually potentially shippable products are not deployed, but the point is that they *could* be. This means that a PBI can only be included in a product increment if it is actually completely done: designed, built, tested, and documented. A team may not succeed in completing every sprint backlog PBI in a sprint. PBIs that are not done are returned to the product backlog and may be included in a subsequent sprint.

Activities

Scrum activities consist of the steps in the process as well as a continuous background grooming activity. A **sprint** is a product development iteration whose goal is to add something to the product of value to a customer or user. Once the product backlog is created, the team is sprinting continuously: as soon as one sprint ends, the next begins. Sprints stop when the project is finished. Sprints have four parts: planning, execution, reviews and retrospectives.

Product Backlog Creation—The product backlog is initially a list of high-level features and functions based on whatever product descriptions the PO has available. There may be a written product vision statement the PO can use for guidance. The PO can also discuss the product with the managers sponsoring the product, salespeople, potential customers and users, and so forth. The product backlog need not have a lot of detail at first because its refinement will be a major activity during the entire development process.

Grooming—PBIs are constantly being added to and deleted from the product backlog, and also modified, refined, and reprioritized in the product backlog; this is called **grooming**. The PO works with stakeholders and the team to groom the product backlog. The PO grooms the product backlog more or less continuously, and typically schedules meetings with the team members during sprints to get help with this task.

Sprint Planning—Sprints begin with sprint planning, during which the PO, SM, and other team members select PBIs from the product backlog to accomplish in a sprint. This choice may be guided by an agreed **sprint goal**, which is the main thing to accomplish in the sprint. PBIs are chosen based on their priority and the effort expected to complete them. The idea is to choose the most valuable PBIs that can be completed in the sprint with everyone working at a sustainable pace. To accomplish this, the team determines for each PBI the tasks required to accomplish it and how long each task will take. These tasks and estimates, along with the PBIs, comprise the sprint backlog.

Sprint Execution—After sprint planning, sprint execution occurs; in other words, the tasks are performed and the PBIs implemented. Various tools and practices (discussed below) may be used to track progress and to maintain productivity during a sprint.

Sprint Review—At the end of a sprint, a sprint review occurs. All stakeholders are invited to the sprint review to see a product demonstration and to discuss the features or functions added, changes made, and so forth, in the sprint. (Remember, each sprint should result in a shippable product so, at the end of the sprint, it should be possible to demonstrate it.) The sprint review thus serves to bring stakeholders up-to-date on the state of development, and to provide feedback to the team about the product and its features and characteristics.

Sprint Retrospective—After the sprint review, the team conducts a sprint retrospective during which they discuss what went well, what did not, and how the next sprint can be done better. The sprint retrospective provides a means to improve the development process. In other words, the sprint review is concerned with the product and the sprint retrospective is concerned with the process. Once the retrospective is done, the sprint is complete and a new one begins.

Scrum Core Practices

In this section we consider in greater detail how scrum is supposed to work by considering how some of the central practices of Scrum are carried out.

Managing the Product Backlog

As we have already noted, the product backlog is a prioritized collection of **product backlog items**, or PBIs. Each PBI should consist of (a) a specification, (b) a priority, (c) an estimate of effort, and (d) acceptance criteria. We consider each of these in turn.

PBI specifications can take any form; for example, they may be traditional requirements statements, user interface diagrams, use cases, user stories (discussed below), defects that need to be fixed, a design task, a research task, or something else. Regardless of their form, it is essential that PBIs be expressible at different levels of abstraction. PBIs often start out as very broad, abstract statements about features or functions or modifications, and then are refined over time into several more detailed PBIs. The product backlog starts out with several very abstract PBIs. The highest priority PBIs are refined (during grooming) into more detailed PBIs that are small enough to be implemented in a sprint. Lower priority PBIs remain abstract until shortly before they are implemented, and then they are refined in preparation for sprinting. In general, a product backlog should contain enough refined PBIs to support two to three sprints, and the remaining PBIs should be more abstract. This deferred refinement helps eliminate waste and rework because several abstract PBIs can be modified more easily than many detailed PBIs. Fortunately, most ways of stating requirements accommodate a wide range of abstraction.

Even when PBIs have been refined to the point that they are small enough to be implemented in a sprint, they are often still not detailed enough to serve as the basis for writing code. It is understood in Scrum that the team will often need to meet with the PO and perhaps customers and users to iron out PBI details during a sprint. Thus even a detailed PBI should be considered a placeholder for (or a promise to conduct) an interaction or conversation among the developers and other stakeholders aimed at nailing down the final details about a portion of the system just about to be implemented.

Perhaps the most popular form of specifying product features in Scrum is the user story. A **user story** is a description of a product feature or characteristic in the form

As a <*user role*> I want to <*goal*> so that <*benefit*>.

The *goal* is what the user wants to accomplish and the *benefit* explains why. For example, the following are user stories.

> As a book critic I want to be able to search for words and phases in a book so that I can analyze the book.
>
> As an author I want to create an index for my document so that I can make it easier to find material.
>
> As a course scheduler I want to determine whether students can take other sections of a course so that I can see if I can cancel a section with students already enrolled in it.
>
> As a shopper I want to see whether an item is still on sale so that I can buy it more cheaply.
>
> As an internet user I want to secure my devices so that I can protect my private information.
>
> As an instructor I want to combine sections of the same course in the course management system so that I only have to maintain one site.
>
> As an electric utility customer I want to see my usage over several years so that I can analyze it to budget my electricity costs more exactly.
>
> As a student I want to analyze my coursework and graduation requirements to plan my schedule for the remainder of my time at school so that I can see what courses I need to graduate on time.
>
> As a traveller I want to find the limits on checked luggage size so that I avoid luggage charges.
>
> As an investor I want to see all my investments from different companies on the same web page so that I can track my portfolio.

As you can see, even though all these sentences are quite similar in size and grammatical complexity, they are at very different levels of abstraction. For example, seeing whether an item is on sale or searching for text in a book are fairly detailed tasks, while displaying investments in a portfolio or creating an index is a mid-level task, and protecting devices or planning an entire four years of school are higher level tasks. The largest, most abstract user stories are sometimes called **epics**; they describe product functions that would take months or entire releases to produce. User stories smaller than epics that still span several sprints are sometimes called **features**. User stories that are small and detailed enough to do in a single sprint are called **sprintable stories** or sometimes just

stories. A product backlog may have a few epics at the bottom, many features in the middle, and enough sprintable stories at the top to support a few sprints.

Note also that even sprintable stories do not have nearly enough detail to actually design and write code from. For example, in searching for text, there are all sorts of questions that come up about how the text should be specified, what search options there might be, and how the results should be displayed. These are all details that the team must determine in conversation with the PO and other stakeholders. Agilists sometimes say that a user story is a placeholder for a conversation; in fact they can be placeholders for several conversations. In principle, the results of these conversations can be documented along with the user story (in any format desired) so that details are not forgotten. However, the emphasis on conversation rather than documentation is consistent with the agile effort to avoid waste and rework in the form of obsolete documentation.

In addition to a specification of the feature, each PBI must have a priority. A **priority** can be a number or a classification (low, medium, high, very high, for example). The priority determines when the PBI will be implemented. Priorities are set by the product owner with feedback from the team and stakeholders. Priorities should take dependencies into account—in other words, if PBI X cannot be done before PBI Y is in place, then Y should have a priority at least as high as X. High-priority PBIs must be small enough that they can be included in a sprint (as discussed above), and they should have more precise effort estimates and acceptance criteria (discussed below), while low-priority items may be larger and more abstract, may have rough estimates, and may not have acceptance criteria.

PBIs must also include **effort estimates**. These are needed for planning sprints and product releases, and for determining priorities. Sprint planning must be done quite precisely, so effort estimates for high-priority items must be in numeric units (such as person days). Low-priority items need only rough effort estimates for use in release planning and prioritizing, so categorical estimates (like small, medium, large, and extra-large) are adequate. As PBIs are refined during grooming (as they near the top of the product backlog), their effort estimates are refined as well. Detailed estimates are deferred to avoid the perhaps wasted work of making detailed effort estimates on low-priority items that may be radically changed, eliminated, or never make it to the top of the product backlog. As PBIs get close to the top of the product backlog, they are refined and their effort estimated carefully so that sprint planning will be more accurate.

Finally, PBIs may also have acceptance criteria. **Acceptance criteria** are checks that a user can make to determine whether a PBI is implemented correctly and completely. They are useful because they help flesh out the user story, making it clearer. These checks can often be used as part of the suite of tests developed to ensure that the PBI has been implemented correctly (more about this below when we discus the definition of done). PBIs not near the top of the product backlog do not need to have acceptance criteria (though they may), but detailed stories near the top of the product backlog should have acceptance criteria developed when they are refined. The acceptance criteria can also be expanded during the sprint in light of further conversations with the PO and other stakeholders.

As noted above, the product backlog must be groomed regularly. Grooming involves adding, removing, or modifying PBIs, elaborating PBIs that are approaching the top of the product backlog, re-estimating PBIs, and re-prioritizing PBIs. Grooming occurs naturally during the sprint review and sometimes during sprint planning. It should also be done at least once during a sprint so that there are enough small and detailed items for the next sprint(s). Some organizations schedule weekly or daily grooming sessions. Grooming requires participation of the team for generating effort estimates, recognizing dependencies, and so forth. Thus it is important that the team factor in time for helping the PO groom the product backlog when planning a sprint.

There are several commercial tools that POs can use to record the PBIs in a product backlog. Some POs just use a spreadsheet or a word processor document. A product backlog can even be kept in paper form on notecards. The tools are not that important; the important point about product backlogs is that they be groomed frequently so that a good selection of sprintable PBIs is ready for sprint planning at the start of every sprint.

A final note about completed PBIs: Once a PBI is done, it is no longer part of the product backlog, which is supposed to be the collection of features, functions, modifications, and so forth yet to be implemented. However, many POs are loath to throw PBIs away. Although finished PBIs should be removed from the product backlog, there is no reason not to put them in some other repository, called perhaps a *product feature list*, that contains all the completed items from the product backlog.

In conclusion, the product backlog is the central artifact of Scrum: it is essentially a stakeholder needs and desires list, and as such it is the lever by which the PO controls development and drives the entire process. Taking care of the product backlog is the most important job the PO has.

Estimating

When developing a product, it is important to be able to answer questions about how long it will take to develop the product, how much it will cost, and when it can be released. To answer questions like this, we need two basic sorts of information: the size of the job, and how quickly the team can get a job of a certain size done. The first requires size estimates, and the second requires data about how much work the team can get done in a sprint.

Scrum uses PBIs to specify products, so estimates of the size of the job are made by estimating the sizes of the PBIs comprising the product. PBI sizes are estimated in either story points or ideal hours. A **story point** is a unit of relative effort. Small PBIs are chosen as representative of one or two story points, and then other PBIs are assigned story point estimates based on how much bigger they are than the representative PBIs. An **ideal hour** or **person hour** is how much an average developer can accomplish in one uninterrupted hour of work. It is usually easier to estimate using story points than using ideal hours, so most Scrum estimators use story points.

Velocity is the amount of work completed by a team in a sprint. Once PBI sizes are estimated, a team need only add up the sizes of the PBIs completed in a sprint to compute its velocity. Thus velocity is measured in either story points per sprint or ideal hours per sprint.

PBI estimates can be used to predict how much effort will be required to complete a product or a subset of the product backlog. Using velocity, we can also estimate how many sprints will be required to implement a product or a subset of the product backlog.

Size estimates and velocity are also important for sprint planning. When preparing for a sprint, the team and PO must agree on the PBIs for the sprint backlog. Obviously, the total estimated size of the PBIs in a sprint backlog should be approximately the team velocity. However, the team typically will want to make more detailed estimates before committing to a set of PBIs for the sprint backlog. We discuss this next.

Creating the Sprint Backlog

Ideally, a team will complete all the PBIs in its sprint backlog on the last day of the sprint, thus having no slack time in the sprint and also having no unfinished PBIs. It is not easy to time things so perfectly, even with a short sprint. During sprint planning, teams work to make more precise estimates before finalizing the sprint backlog. This is done by considering, for each PBI, every task needed to complete it and estimating the effort of each task in ideal hours. For example, suppose a PBI is captured by the following user story.

> As an auditor I want to toggle an invoice annotation display using an annotation checkbox so that I can understand product and service codes.

This feature will require modifying the user interface to include checkboxes for each invoice, then writing code to scan an invoice, extract its product and service codes, look them up in a master code list, and display the results in a window off to the side of the invoice display. The team divides this work into four different coding tasks, and each is estimated. The coding tasks are assumed to include unit testing activities, but the entire feature must be integrated and tested (including PBI acceptance criteria testing—see below), which the team records and estimates as another task. The team inspects all its code, so the inspection task is estimated and included. Finally, modifying the user documentation to include this feature is estimated and added as a final task. Thus the original PBI, with its size estimate in ideal hours or story points, has been augmented with seven tasks needed to complete the PBI, each with a more precise estimate in ideal hours. The team can then be much more confident about how much effort completion of this PBI will entail, and it can factor this information into its decision-making about what will go into the sprint backlog.

In summary then, during sprint planning the team and the PO make an initial stab at the contents of the sprint backlog based on the sizes of the PBIs at the top of the product backlog. The team then analyzes the PBIs in more detail, generating tasks and task effort estimates for each PBI. Based on this analysis, the team reconvenes with the PO and comes to final agreement on the contents of the sprint backlog. The sprint backlog then consists of (a) PBIs, (b) the tasks necessary to complete them, and (c) effort estimates for all task, usually in ideal hours. The team is then ready to execute the sprint.

Sprinting

When a person has some work to do, he or she may either set the scope of the work and then expend as much time as necessary to do the job, or he or she may set the time to be spent and adjust the scope of the work to fit the time. The latter approach is called **time-boxing**. Sprints are time-boxed, meaning that they are of fixed duration, typically one week to one month long. Furthermore, sprints should have a consistent duration, except for occasional special sprints, typically at the start or beginning of a project or a release.

These characteristics have several advantages. Short, consistent, time-boxed sprints are easier to plan and track. They provide rapid feedback about development so that mistakes and misunderstandings are caught quickly. They provide lots of information about the current state of development and lots of opportunities for feedback about the developing product.

Sprints should not be shortened or extended. Furthermore, the PO should never try to alter the sprint backlog during a sprint. If a team cannot finish the items in its sprint backlog, then the unfinished items go back in the product backlog. If a team finishes all PBIs before the end of the sprint, then they can consult with the PO for another PBI to work on. Inaccuracies in time estimates should be identified during the retrospective, and steps should be taken to improve their accuracy in the future.

Tracking Progress with Burn Charts and Task Boards

Any project, even an agile project, must have some way to monitor whether work is progressing as expected. Agile methods, being light-weight, use techniques that do not require a lot of effort to collect data or produce management artifacts. One of the main tools for tracking progress in Scrum projects is a **burn chart**, which is a graph displaying work on the vertical axis and time on the horizontal axis. If a burn chart shows how much work remains against time, then it is a **burn-down** chart; if it shows how much work has been accomplished, then it is a **burn-up** chart. Both kinds are used in Scrum projects, but burn-down charts are usually preferred for sprints.

A burn-down chart for a sprint has the total effort (usually expressed in story points) for the sprint marked on the vertical axis, and the horizontal axis marked with work days in the sprint. On the day that one or more PBIs are finished, there is that much less work to do in the sprint, so a point is marked showing the story points remaining on that day. Connecting these points produces a line showing progress in the sprint. A straight diagonal line from the starting point on the vertical axis to the last day of the sprint on the horizontal axis serves as a reference line showing ideal progress.

Consider the burn-down chart in the figure below. This burn-down chart shows the progress of work in a three-week sprint (with 15 work days) that began with PBIs totaling 28 story points of effort. On day two of the sprint, one PBI worth three story points was completed, so a point was plotted at that spot. Further points were plotted as the sprint progressed. The last PBI was completed on day 14; one PBI worth three story points was not completed in the sprint, so the progress line does not meet the horizontal axis. Comparing the solid progress line with the dashed ideal progress line shows that work in the sprint was a bit behind right from the start, though progress was steady. It may be that the team simply took on a bit too much in this sprint.

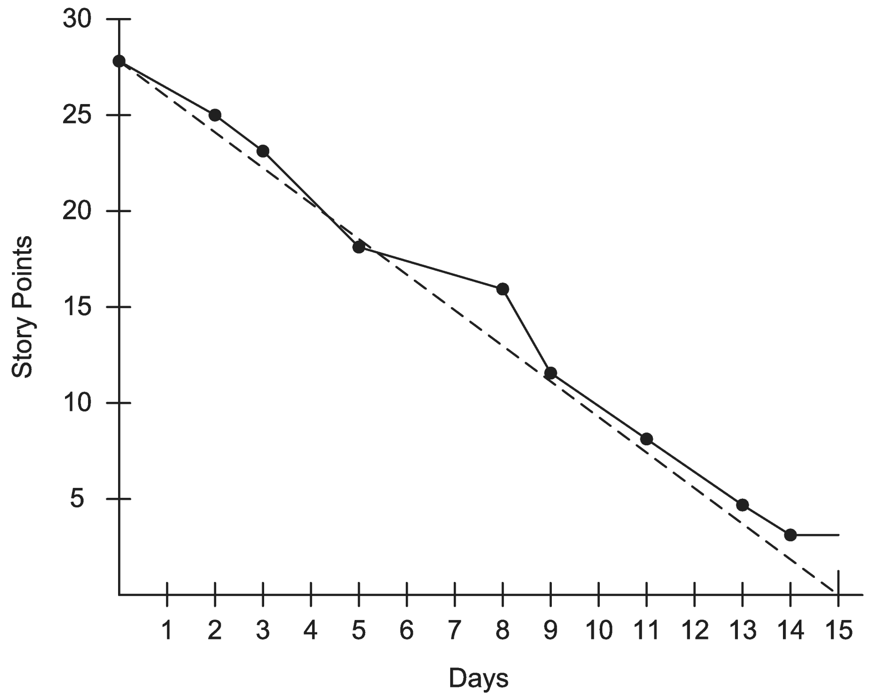

Figure 2: Sprint Burn-Down Chart

Burn-down charts are easy to make: once they are laid out, a point is plotted at the end of any day that some PBIs are completed. Teams usually post them in a prominent spot in their work area so that everyone can see how work in the sprint is going at any time.

Burn-down charts show the progress of work during a sprint in terms of story points, but they can only show progress at the level of PBIs. During a sprint, work has been planned in smaller units of tasks, so it is useful to be able to see the status of all the tasks in the sprint. A **task board** does this. It is laid out as a table whose rows are the sprint backlog PBIs and whose columns are areas showing tasks that have not yet been started, are being worked on, or are done. Task board column headers are "PBI," "To Do," "In Progress," and "Done." Cards or sticky notes are filled out for each PBI in the sprint backlog and for all the tasks for each PBI. The PBI cards serve as row headers, and the task cards all start in the To Do column at the start of the sprint. When a team-member is ready to work on a task, he or she chooses one from the To Do column and moves it into the In Progress column. When that task is finished, it is moved into the Done column.

Figure 3 below shows the layout of a task board. In this particular task board there are four PBIs in the sprint backlog. Each has several tasks associated with it, all appearing in that PBI's row. The task board shows that Story4 is nearly finished because all its tasks are either done or in progress. Work on Story3 has clearly just gotten started, however, because only one of its five task cards is in progress.

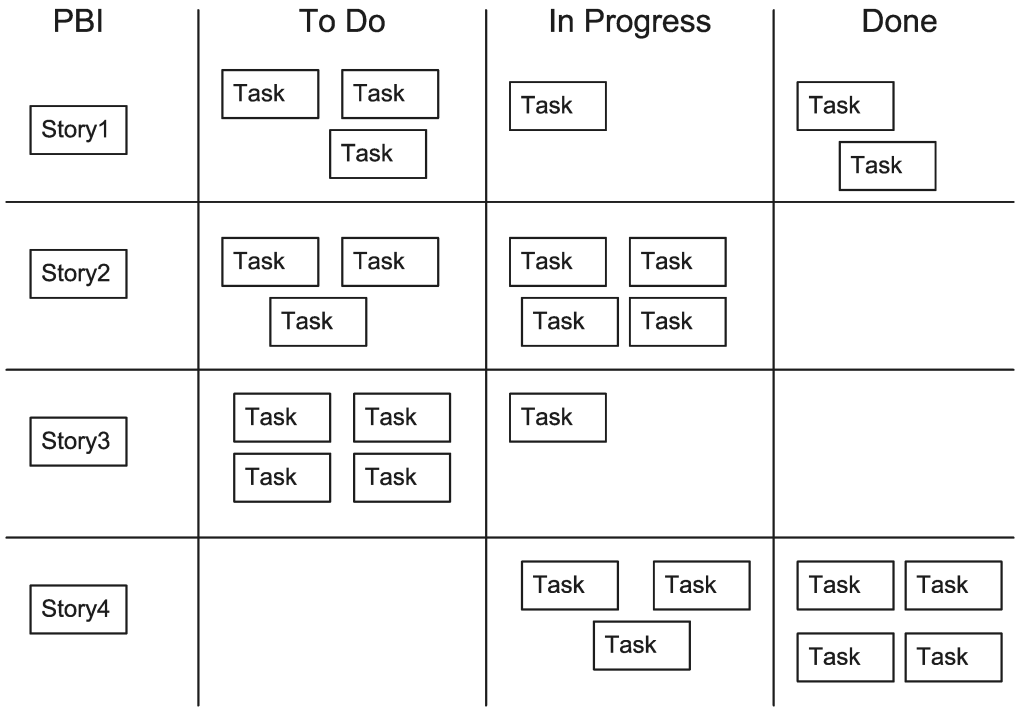

Figure 3: A Task Board Layout

As this schematic example shows, it is very easy to use task board, and easy to tell by looking at them the state of the tasks in the sprint. Teams can use this tool to decide how to concentrate their effort, to detect when they are getting behind in their work, to identify tasks that are giving a team-member trouble, and so forth.

Making and Using a Definition of Done

One pitfall that teams sometimes fall into is to count a PBI as finished during a sprint when really it is not. The goal is to have a potentially shippable product after every sprint, so if a PBI is not really finished, then it should not be counted as part of a potentially shippable product. More pragmatically, counting a PBI as done when it really is not gives a misleading impression about the state of the project, and leaves work to be done that does not fit in anywhere—later sprints will be concerned with other PBIs, so when will the unfinished PBI be completed?

To help keep this from happening, each team should devise its own definition of done. A **definition of done** is a checklist of what needs to be accomplished for a BPI to be considered completed. Typical checklists include the following items.

- The design is complete and reviewed.
- The code has been formatted and commented.
- The code has passed inspection.
- The code has passed all PBI acceptance criteria (usually embodied in tests).
- The code has passed all unit tests (and regression tests).
- User documentation has been updated.
- The code has been integrated and passed all integration and system tests (and regression tests).

A team adopts a definition of done for all PBIs and a PBI is not considered done until it meets *all* items in the checklist. The best way to manage this is to create a definition of done checklist for each PBI and use it as a guide during the sprint to ensure that all work needed to complete each PBI is actually performed.

Finishing a Sprint

The two activities at the conclusion of a sprint are both aimed at reflection and improvement: the sprint review reflects on the product to improve it, and the sprint retrospective reflects on the process to improve it.

At a sprint review the team presents what it accomplished in the sprint. Only completed aspects of the product should be presented. All stakeholders should be invited to sprint reviews. Ideally, sprint reviews are held at a regular time so that stakeholders can put them on their calendars as recurring meetings, thus making it easier for all interested parties to attend.

Sprint reviews should be informal but follow more or less the following agenda.

1. The meeting begins with a team member (usually the PO) presenting the overall sprint goal (if any) and the PBIs in the sprint backlog. The team also lists the PBIs completed and explains any discrepancy between the planned and actual work accomplished. This is for informational purposes only, not as a basis for a discussion about team performance.
2. One or more team members demonstrate the new aspects of the product. This demonstration can include tests showing that parts of the product work better or correctly even if improvement don't result in anything evident to users. In this way there will be something to demonstrate even if most changes are improvements without much obvious change in product behavior.
3. All participants discuss the product and its development path. For example, the stakeholders may say whether they are happy with the product, whether its features are what they need and expect, what features might need to be changed, what features might still be needed, and so forth. The team might suggest changes or propose ideas they may have had for the product. The goal is to decide on the next steps in development to ensure that the team is always working on the most important and valuable changes that can be made to the product.

The results of the sprint review are fed forward into planning for the next sprint and should be reflected in the product backlog.

The sprint retrospective is a meeting for the team only, but it should include the entire team, including the PO and the SM. Its purpose is to devote time to analyzing the way the team works, identifying ways to improve, and making plans to incorporate the improvements into the way the team works.

When teams start using Scrum, enough problems come up that there is always at least one thing from the most recent sprint that the team wants to improve. Usually it is a good idea to work on only one improvement at a time. It may take several sprints for an improvement to be fully incorporated into a team's practice. After a team has been sprinting together for some time, however, they may become quite comfortable with what they are doing, and obvious improvements may not present themselves. At that point, the team may need to start working harder to brainstorm improvement opportunities, collect and analyze data about what they are doing, research possible improvements, and figure out how to make them. Improvement activities may even be incorporated as team goals with time allocated for working on them outside regular development activities.

Other Scrum Practices

There are several other practices that are common in Scrum. Many of them come from other agile methods and have been adopted by Scrum teams.

Daily scrum—A short (maximum 15 minute), often stand-up (no chairs) meeting in which each member states three things: (a) what did I do since the last meeting, (b) what will I do today, (c) what is impeding my progress. These meetings allow team members to decide how to spend their time that day. For example, if all is well, a team member may choose a task that still needs to be done in the sprint and work on it. But if a team-mate is struggling with a problem, a team member may choose to help them, or if there is some impediment to progress, several team members may meet after the daily scrum to figure out how to resolve it. The daily scrum is thus a just-in-time technique to help a team organize its daily activities.

Story time—This is a regularly scheduled meeting to groom the product backlog. As noted, although the PO is responsible for the product backlog, grooming often requires input from the team; story time provides an opportunity for the PO and the team to meet.

Cross-functional teams—Scrum teams usually include people with all the skills necessary to carry a product increment from requirements to deployment, including designers, testers, coders, documenters, and so forth. But if there is too much of one kind of work for specialists to accomplish in a sprint, then everyone else helps out. This is viewed as a way for team members to increase their skills, to keep growing, and to stay interested.

Sustainable pace—Although this idea comes from XP (XP is short for *Extreme Programming*, one of the first and most influential agile methods), it is common to all agile methods. Each sprint should be planned to keep everyone busy until the end, but without a mad rush to finish. Furthermore, developers are not expected to work overtime. Sprints are not extended and only shortened if the team asks them to be. Sprint backlogs cannot be changed during a sprint. If something is not complete, at the end of the sprint it goes back into the product backlog. This helps avoid pressure to over-work.

Planning poker—This is a technique that a team can use for jointly estimating PBIs. It uses story point cards in a format resembling poker. After describing a PBI, each team member throws an estimate card on the table. If there is disagreement, each team member explains the basis for their estimate, and the cards are collected and then thrown again, presumably reflecting estimates modified in light of the discussion. This continues until there is consensus. Planning poker is discussed in more detail in another chapter.

Bidding—This is also a technique for estimating the effort required to complete PBIs. In essence, team members take ownership of particular tasks by bidding on them in ideal hours. If one team member places a bid that is too high, another can come along and offer to complete it in less time.

Pair programming—This technique also comes from XP. Two team members sit together at one terminal, with one person typing. The two collaborate writing code (or whatever else they are doing), with the non-typing person checking what the typing person is doing. The idea is to incorporate instantaneous inspection and correction into the development process.

Thanks to its flexibility as a process framework rather than a strict process specification, Scrum can incorporate any technique that a team feels would benefit their work. Consequently there is a wide range of practices used by Scrum teams including and going beyond those listed above.

References

1. Takeuchi, Hirotaka, and Ikujiro Nonaka. 1986. "The New New Product Development Game." *Harvard Business Review*, January, 137–146.
2. Schwaber, Ken. 1995. "Scrum Development Process." In *OOPSLA Business Object Design and Implementation Workshop*, ed. J. Sutherland et al. Springer.
3. Schwaber, Ken, and Mike Beedle. 2001. *Agile Software Development with Scrum*. Prentice Hall.
4. *The Scrum Guide*. http://www.scrumguides.org/.
5. Rubin, Kenneth S. *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. Addison-Wesley, 2012.