Chapter 5
Scope Management

Project Scope Management includes the processes required to ensure that the project includes all the work required, and only the work required, to complete the project successfully.

—PMBOK® Guide

It is not the strongest of the species that survive, nor the most intelligent, but the ones most responsive to change.

—*Charles Darwin,* The Origin of Species

Next week there can't be any crisis. My schedule is already full.

—*Henry Kissinger*

"Scope creep" has always been the bane of traditional project managers, as requirements continue to change in response to customer business needs, changes in the industry, changes in technology, and things that were learned during the development process. Scope planning, scope definition, scope verification, and scope control are all processes that are defined in the *PMBOK® Guide* to prevent scope creep, and these areas earn great attention from project managers. Those who use agile methods believe these deserve great attention as well, but their philosophy on managing scope is completely different. Plan-driven approaches work hard to prevent changes in scope, whereas agile approaches expect and embrace scope change. The agile strategy is to fix resources and schedule, and then work to implement the highest value features as defined by the customer. Thus, the scope

remains flexible. This is in contrast to a typical waterfall approach, as shown in Figure 5-1, where features (scope) are first defined in detail, driving the cost and schedule estimates. Agile has simply flipped the triangle.

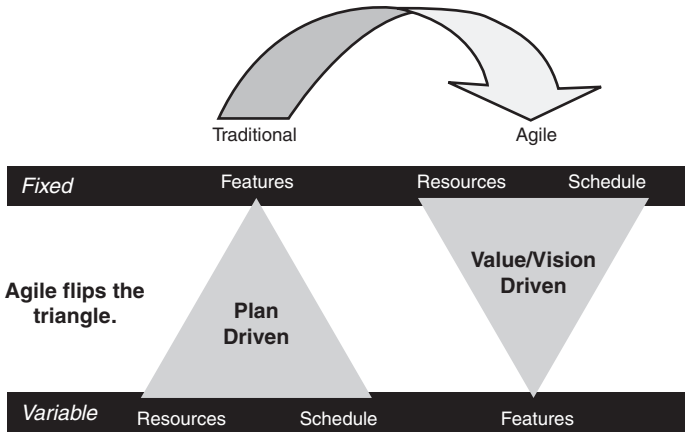


Figure 5-1
Waterfall vs. Agile: The paradigm shift (original concept courtesy of the DSDM Consortium)

Scope Planning

The *PMBOK® Guide* defines the Project Scope Management Plan as the output of the scope planning process.[1] This document defines the processes that will be followed in defining scope, documenting scope, verifying and accepting scope and completed deliverables, and controlling and managing requests for changes to the scope. In agile, the iterative and incremental process itself is what manages scope. Unless documentation is required for auditing purposes, no additional document outlining procedures for scope management is needed. Scope is defined and redefined constantly in agile, as part of the planning meetings—in particular, release planning and iteration planning—and by the management of the product backlog. Remember, resources and time are typically fixed in agile approaches, and it's the scope that is allowed to change. However, when fixed-scope projects are required, it is the number of iterations that will change, in order to accommodate the need for a full feature set prior to release. Additionally, one of the success criteria in traditional projects is the extent to which we can "stick to the scope"; in agile, it is more important to be able to efficiently and effectively respond to change. The success criteria in agile thus changes to "Are we providing value to our customer?" The primary measure of progress is working code.

Table 5-1 provides a summary comparison of scope planning from the traditional and agile perspectives. In agile projects, scope planning is referred to as "managing the product backlog."

Table 5-1
Scope Planning

Traditional	Agile
Prepare a Project Scope Management Plan document.	Commit to following the framework as outlined in the chosen agile process.

Scope Definition

The *PMBOK® Guide* practices of scope definition, work breakdown structure (WBS) creation, and scope verification occur iteratively in agile. A traditional WBS for software projects is usually divided at its highest level into phases of analysis, design, coding, testing, and deployment activities. Each of these phases is then decomposed into tasks or groups of tasks, referred to as work packages in the *PMBOK® Guide*. Traditional project planning begins top-down and relies on the elaboration of detailed tasks with estimates and dependencies to drive the project schedule via use of critical path analysis. Even though the *PMBOK® Guide* goes into great detail about scope decomposition by way of WBS (work breakdown structure), it also warns that "excessive decomposition can lead to nonproductive management effort, inefficient use of resources, and decreased efficiency in performing the work."[2]

In agile, we approach these practices differently in that we define features at a high level in the product backlog and then place features into iterations during release planning. One can think of the iteration—or even the feature itself—as the agile equivalent of work packages. The features are estimated at a gross level in the product backlog—no detailed tasks or resources are defined at this point in time. Once the iteration begins, the features slated for that iteration—and only that iteration—are then elaborated into tasks that represent a development plan for the feature. Think of it as just-in-time elaboration, preventing a wasteful buildup of requirements inventory that may never be processed. The *PMBOK® Guide* supports this idea of "rolling wave planning":[3] As the work is decomposed to lower levels

of detail, the ability to plan, manage, and control the work is enhanced because the short timeframe of the iteration reduces the amount of detail and the complexity of estimating. The agile approach assumes that because things change so often, you shouldn't spend the time doing "excessive decomposition" until you're ready to do the work.

Let's look at how scope is defined throughout an agile project by examining five levels of planning common to most agile projects: the product vision, the product roadmap, the release plan, the iteration plan, and the daily plan.[4]

Product Vision

At the outset of a project, it is typical to hold a kickoff meeting. Agile is no different; however, the way the agile vision meeting is conducted is unlike what a traditional project manager might be accustomed to. Although the vision is defined and presented by the customer or business representative, it is the team that clarifies the vision during the discussions and subsequent exercises. Therefore, the team is heavily involved, and group exercises are a big part of determining the final outcomes. See Chapter 4, "Integration Management," for more detail on vision meetings.

The vision meeting is designed to present the big picture, get all team members on the same page, and ensure a clear understanding of what it is that they've been brought together to do. The vision defines the mission of the project team and the boundaries within which they will work to achieve the desired results. The project's goal should be directly traceable to a corporate strategic objective.

Here the scope is defined at a very high level. It is not uncommon to leave the vision meeting with only a dozen or so features identified, such as "provide online order capabilities," "enable international ordering and delivery," "create data warehouse of customer orders to use for marketing purposes," and "integrate with our current brick-and-mortar inventory system." Clearly these are all very large pieces of functionality with little-to-no detail—and this is what is appropriate at this stage of the project. The farther away the delivery date, the broader the stroke given to feature details.

Product Roadmap

A product roadmap shows how the product will evolve over the next three to four releases or some period of calendar time, typically quarters. The

product roadmap is a high-level representation of what features or themes are to be delivered in each release, the customer targeted, the architecture needed to support the features, and the business value the release is expected to meet. The customer or product manager, agile project manager, architect, and executive management should meet on average two to three times a year to collaborate on the development and revision of the product roadmap. Figure 5-2 shows a sample roadmap template made popular by Luke Hohmann in his book *Beyond Software Architecture.*[5]

> **Note**
>
> In agile, the word "release" does not solely mean a product release to the end customer—it can also mean an internal release to fulfill integration milestones and continue to confirm that the product is "potentially shippable."

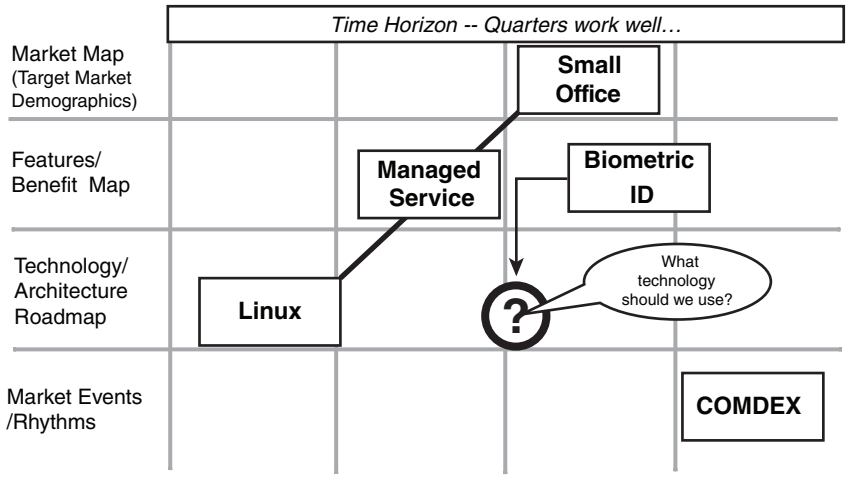


Figure 5-2
Product roadmap template, courtesy of Enthiosys and Luke Hohmann, from his book *Beyond Software Architecture*

Because the customer is responsible for maintaining and prioritizing the backlog of work, the customer also owns the product roadmap. In large corporations or on projects with multiple customers or product owners, the customer assigned to the project will often first work with others in his business unit to create a roadmap straw man as part of working out the priorities of deliverables with the business. Then this straw man is presented to key project team members (agile project manager, architect, and so on) for further revision. Finally, the roadmap is presented to the entire team and interested stakeholders, usually as part of the vision meeting and/or release

planning meeting. Feedback is encouraged at all sessions because it helps to better define a reasonable approach to product deliverables.

In addition to the vision plan and product roadmap, the end result of the product vision and product roadmap discussions should be the prioritized product backlog. These are all inputs into the next level of planning: release (or quarterly) planning.

Release (or Quarterly) Planning

In a release planning meeting, the team reviews the strategies and vision shared by the customer and determines how to map the work from the prioritized backlog into the iterations that make up a release or that make up a period of time such as a quarter. Figure 5-3 shows a typical release plan agenda, and Figure 5-4 shows the release plan done using a whiteboard and sticky notes, as is common in agile meetings when the team is co-located. The release plan is divided up into iterations (usually one flipchart page per iteration), with associated high-level features. The release plan also includes any assumptions, dependencies, constraints, decisions made, concerns, risks, or other issues that may affect the release. Again, documentation of these additional items can be as simple as posting the flipchart that they were originally recorded on or taking a picture of it and posting it on a shared website.

Last Responsible Moment Decision Points

Note that one of the items on the release planning meeting agenda is the identification of "Last Responsible Moment (LRM) decision points." LRM decision points identify points in the release where a decision must be made on an issue so as not to allow a default decision to occur. In other words, they identify "the moment at which failing to make a decision eliminates an important alternative".[6] Up until this point, the team can continue its momentum and gather additional information that will help in the decision-making. For example, one team knew it would have to make a decision between going with a Sybase database and an Oracle database. But the team did not have to decide this before they could start on the project—indeed, the team realized that it could develop code that was database-independent until the third iteration, when integration and reporting were required. Therefore, the team set the end of the second iteration as its LRM on the database decision, giving the architect and the DBA time to experiment with the work being developed concurrently.

Release Planning Meeting Agenda

- *Introductions, ground rules, review of purpose and agenda (Project manager)*
- *Do we need to review our current situation and/or existing product roadmap? (Project manager, architect, customer/product owner)*
- *Do we remember the product vision? Has it changed? (Customer/product owner)*
- *What is the release date? How many iterations make up this release? (Project Manager)*
- *What is the theme for this release? (Customer/product owner)*
- *What are the features we need for this release? (Customer/product owner)*
- *What assumptions are we making? What constraints are we dealing with? (Team)*
- *What are the milestones/deliverables expected? Do we have any LRM decision points? (Team)*
- *What is the capacity of the team (iteration velocity)? (Team)*
- *Can we move the features into the iterations? Do we need to break them into smaller features so that they can be completed in a single iteration? (Team)*
- *What issues/concerns do we have? (Team)*
- *Can we commit to this release as a team, given what we know today? (Team)*
- *Close: empty parking lot, action items, next steps (Project manager)*

Figure 5-3
Release planning meeting agenda

Figure 5-4
Release plan

Coordinated Release Planning

A colleague of ours once ran a release planning meeting with teams located in the U.S. and in London. Because of the size of the team and the budget constraints, not everyone could attend the day-long event. So the meeting was broken out into three days. Day 1 was focused on the U.S. team's release plan and all its assumptions about and dependencies on the London team. Due to time zone issues, the London team listened in on the phone for the first part of the meeting as the vision and the high-level detail and expectations around the features were discussed, then dropped off the call once the U.S. team started on the work of moving the features into the iterations. On Day 2, the London team did its work of moving the features into the iterations after reviewing the results of the U.S. team's release plan (photos and notes were made available on their shared wiki). At the end of Day 2, the London team posted its release plan. Day 3 was devoted to the coordination of the two plans, making sure all assumptions had been addressed and understood, all dependencies accounted for, and proper prioritizations had been made reflecting the teams' constraints. Both groups committed to the release plan on the third day after some final tweaking.

Teams that are not co-located should make every effort to bring everyone together for this meeting. Agile emphasizes face-to-face communication because of its benefits. However, balancing this with the realities of geographically dispersed teams means that budget constraints force teams to be selective about when they can gather together as a group. The vision and release planning meetings should receive high priority, because the information shared and decisions made in these meetings guide the team throughout the remainder of the release.

Iteration Planning

Traditional scope definition and many of the practices defined in the *PMBOK® Guide* knowledge area of Project Time Management are done as part of iteration planning. Here, features are elaborated (creating the equivalent of *PMBOK® Guide* work packages), tasks are identified, and the time needed to accomplish the tasks is estimated (see Figures 5-5 and 5-8). At the beginning of each iteration, the team should hold an iteration planning meeting to conduct this work. The team reviews the release plan and the prioritized items in the backlog, reviews the features requested for the current

iteration, and tasks out and estimates those features. See Figure 5-6 for a typical iteration planning meeting agenda. In keeping with the agile practice of just-in-time design, it is here that the details of the features are discussed and negotiated.

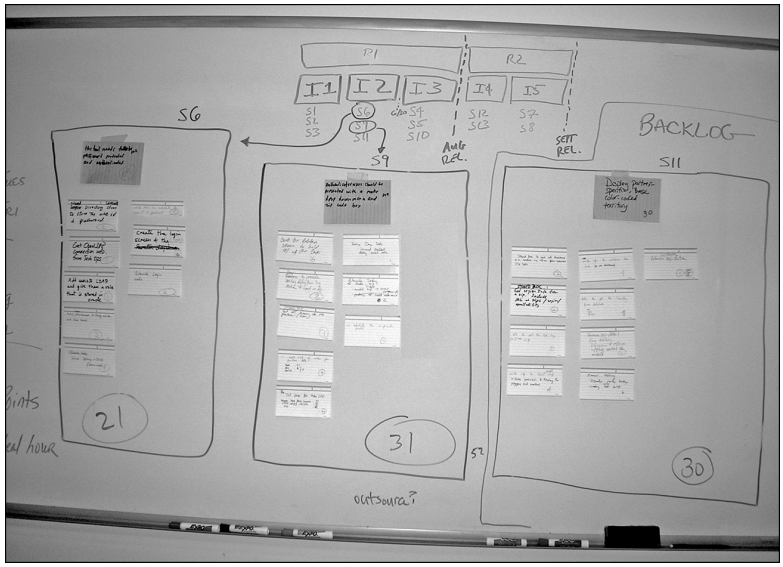


Figure 5-5
Iteration plan

Iteration Planning Meeting Agenda

- *Introductions, ground rules, review of purpose and agenda (Project manager)*
- *Do we know our iteration start and end dates? (Project manager)*
- *Do we know the team's velocity? (Team)*
- *Do we know what "done" means? (Team)*
- *What are the features we need for this iteration? What is the acceptance criteria for each feature? (Customer/product owner)*
- *Do we have enough information about the features so that we can task them out? (Team)*
- *Can we estimate the time it takes to complete the tasks? (Team)*
- *What assumptions are we making? What constraints are we dealing with? Are there dependencies that affect our prioritization? (Team)*
- *Are we within our velocity limits? (Team)*
- *What issues/concerns do we have? (Team)*
- *Can we commit to this iteration as a team, given what we know today? (Team)*
- *Close: empty parking lot, action items, next steps (Project manager)*

Figure 5-6
Iteration planning meeting agenda

Again, planning and design work is done only for the pieces that are being readied to code in that iteration, not for the entire system. It's often discovered during iteration planning that the sum of the task efforts exceeds the size of the iteration timebox. When this occurs, some of the work needs to be shifted either into the next iteration or back into the backlog. Similarly, if a team discovers that it has chosen too little work for the iteration, it will consult with the customer, who can then give the team an additional feature or two to make up the difference. This allows the team to make a realistic commitment to the scope of the work being defined.

Daily Stand-Up

One of the key heartbeats of agile development involves the practice of daily stand-up meetings. It is just what it sounds like: a daily meeting, where all team members attend, and while remaining standing, they each relate their status to the other team members and their plan for the day based on the progress that they've made. Standing helps keep the meetings short—stand-ups should run only 5 to 15 minutes. Its primary purpose is for the team members to inspect and adapt its work plan (iteration backlog) by quickly sharing information about the progress (or lack of) being made by each individual regarding the tasks that were committed to during the iteration planning meeting. These stand-ups help the team to remain focused on the agreed-to scope and goals of the iteration.

Summary Comparison

Table 5-2 provides a summary comparison of traditional and agile approaches to scope definition. In agile projects this is called "multilevel planning."

Table 5-2
Scope Definition

Traditional	Agile
Prepare a Project Scope Statement document that includes items such as the following: Project boundaries and objectives, product scope description…	Conduct a vision meeting to share the product vision; confirm and clarify the boundaries, objectives, and product scope description using exercises such as the elevator statement and design the box.

Traditional	Agile
And major milestones and project deliverables…	Conduct a planning meeting to prepare the product roadmap, as well as release or quarterly planning meetings that also include milestones and deliverables at an iteration level.
And product specifications and acceptance criteria…	Conduct an iteration planning meeting that results in the detail around each feature, and the tasks needed to complete the feature according to the team's definition of "done" and the acceptance criteria defined by the customer.
And assumptions and constraints.	All planning meetings identify and/or review assumptions and constraints.

Create a WBS

Agile teams do not tend to create formal WBSs (work breakdown structures). Instead, flipcharts and whiteboards are used to capture the breakdown of work. You've seen examples of these in Figures 5-4 and 5-5. So at the end of release planning, the agile equivalent of a WBS—a feature breakdown structure—would look like the sample release plan feature breakdown structure in Figure 5-7. If having iterations as work packages is not sufficient for your organization/billing needs, then breaking the work down further into smaller work packages would look like the results of an iteration planning meeting, as illustrated in Figure 5-8.

Table 5-3 compares the traditional and agile approaches to work breakdown. In agile projects, the work breakdown structure is captured in the release plan and the iteration plan.

Table 5-3
WBS Creation

Traditional	Agile
Create a work breakdown structure diagram.	Conduct planning meetings and give the team the responsibility for breaking down the work into smaller work packages (features and tasks), displayed as the release plan at the high level, and the iteration plan at the more detailed level.

Figure 5-7
Release plan feature breakdown structure

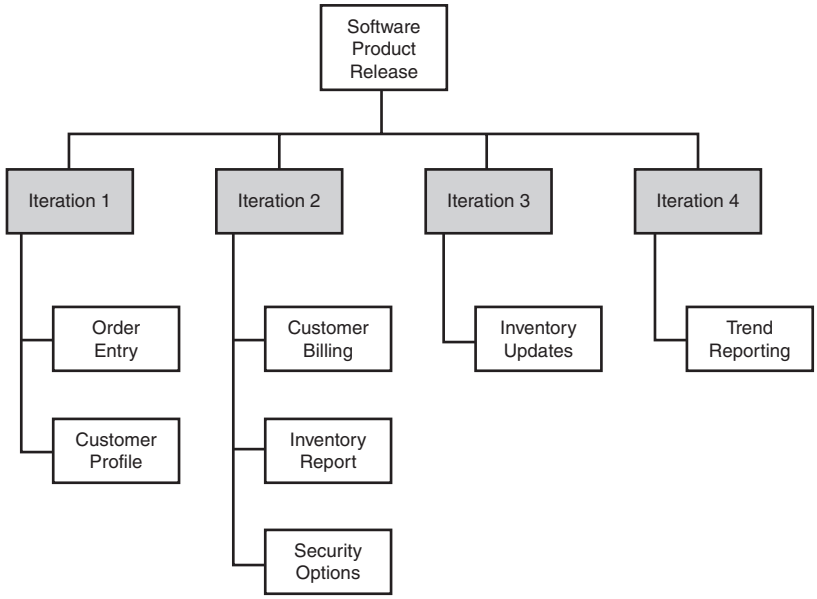
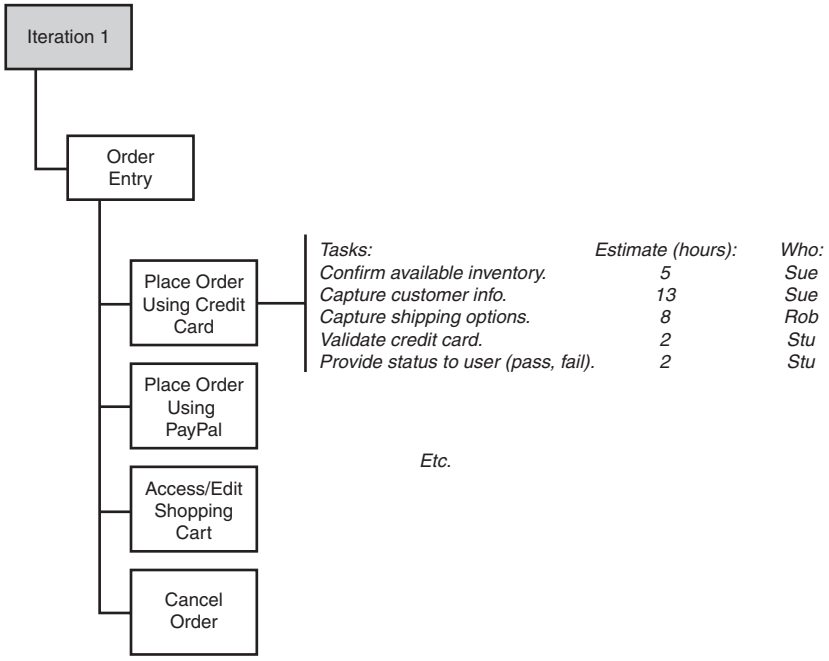


Figure 5-8
Iteration plan (partial)



Scope Verification

Scope verification is accomplished within the iteration, as the customer gets to review, test, and accept the implemented features. Ideally this happens throughout the iteration, but it can also happen at the end of the iteration, during the demo of the working code. Those features that were not accepted (either because they weren't ready or weren't right) move back into the backlog or into the next iteration at the discretion of the customer. Scope change control is handled by the management of this backlog, as discussed in the previous chapter on integration.

Table 5-4 makes the comparison between the traditional and agile approaches to scope verification. Scope verification is captured by the agile practices of acceptance testing and customer acceptance.

Table 5-4
Scope Verification

Traditional	Agile
Document those completed deliverables that have been accepted and those that have not been accepted, along with the reason.	Documentation of accepted features may be done informally (by moving the sticky notes to the "done" pile) or formally.
Document change requests.	Customer updates the backlog.

Scope Control

Controlling scope in agile projects consists of two things: managing the product backlog and protecting the iteration. Whereas the customer maintains the backlog, it is the agile project manager who protects the team and helps prevent scope changes from occurring during the iteration.

When a team commits to the iteration at the end of the iteration planning meeting, the delivery team is effectively saying, "Given what we know today, we believe we can deliver this work using our definition of 'done' within this iteration," and the customer is effectively saying, "Given what I

know today, this is the work that I am expecting by the end of the iteration, and during that time I will not mess with the iteration backlog" (that is, scope). The iteration backlog is thus locked in.

It is important to set the length of your iteration accordingly, because the customer must wait until the next iteration to make changes. If there happens to be lots of "requirements churn" (that is, requests for changes are coming in very frequently), you may want to discuss shorter iteration cycles with the team in order to enable more frequent changes. Maintenance teams may have iteration lengths of only one week, whereas larger system developments with known requirements may have an iteration length of four to six weeks. If the customer keeps trying to interrupt the team with changes, the iteration length may be too long.

There will always be exceptions, and in those cases a discussion between the customer and the agile project manager should help identify potential resolutions. Iterations can be aborted and restarted, but this should be the rare exception.

Given the short duration of iterations, it is easy to protect the iteration backlog from change. However, changes in the product roadmap and the release plan are expected and therefore should be reviewed regularly.

Table 5-5 lists out the differences between the traditional and agile approaches to scope control. Agile users refer to scope control as "managing the product backlog."

Table 5-5
Scope Control

Traditional	Agile
Use a change control system to manage change.	The customer manages the product backlog; once the team commits to the work to be done in an iteration, the scope is protected for that duration.
Update all documents as appropriate with the approved changes.	The team revisits release plans and product roadmaps regularly, making changes as needed to better reflect the team's progress and changes requested by the customer.

Summary

The main points of this chapter can be summarized as follows:

- "Scope creep" doesn't exist in agile projects, because scope is expected to change.
- Scope management in agile is primarily a function of "rolling wave" planning and the management of the product backlog.
- Scope is defined and redefined using five different levels of planning that take the team from the broad vision down to what team members plan to complete today.
- WBSs are not created per se; instead, release/quarterly plans and iteration plans serve to break down the work into smaller work packages, referred to as "features and tasks."
- Scope is verified by the customer, who is responsible for accepting or rejecting the features completed each iteration.
- Scope is controlled through the use of the backlog, rolling wave planning, and the protection of the iteration.

Table 5-6 presents the differences in project management behavior regarding scope management in traditional and agile projects.

Table 5-6
Agile Project Manager's Change List for Scope Management

I used to do this:	Now I do this:
Prepare a formal Project Scope Management plan.	Make sure the team understands the framework and process structure of the chosen agile approach.
Prepare a formal Project Scope Statement document.	Facilitate planning meetings—vision, release, iteration, daily stand-up—and arrange for the informally documented plans to be highly visible to all stakeholders.
Create the WBS.	Facilitate the release planning meeting so that the team can create the plan showing the breakdown of work across several iterations.

(continued)

Table 5-6
Agile Project Manager's Change List for Scope Management *(continued)*

I used to do this:	Now I do this:
Manage the change control system and try to prevent scope creep.	Step away from the backlog; it is owned by the customer. If needed, remind the customer that during the iteration, the team is protected from scope changes.
Manage the delivery of tasks to prevent or correct scope creep at the task level.	Allow team members to manage their daily tasks and facilitate conversations with the customer to avoid unnecessary work or "gold plating."

Endnotes

1. *PMBOK® Guide*, 107.
2. Ibid, 114.
3. Ibid.
4. Mike Cohn. *Agile Estimating and Planning* (Upper Saddle River, NJ: Pearson Education, Inc., 2006), 28.
5. Luke Hohmann. *Beyond Software Architecture* (Boston: Addison-Wesley, 2003), 287.
6. Poppendieck. *Lean Software Development*, 57.