4 Agile Planning, Requirements, and Product Backlog

AGILE PLANNING PRACTICES

PROBABLY THE BIGGEST DIFFERENCE between agile and more traditional project management approaches is in the area of planning. There's a misconception that agile projects do not require planning. That is not the case—they require just as much or more planning; it is just done very differently.

Rolling-wave planning

Traditional, plan-driven project management approaches typically attempt to do more of the planning prior to the start of the project, while agile project management approaches typically defers planning decisions to "the last responsible moment." By *the last responsible moment*, we mean the latest point in time that a decision can be made without impacting the outcome of the overall project.

That is what is called *rolling wave planning*:

- You typically start with a high-level plan that is sufficient for defining at least the vision, scope, and objectives of the project to whatever level of detail is needed at that point to support whatever level of planning and estimation is required for the project.
- The details of the plan and the requirements are further elaborated as the project progresses.

The thinking behind that strategy is that attempting to plan too far in advance naturally involves some amount of guesswork and speculation. Quite often, that speculation is wrong and will result in wasted effort in replanning later, and it might also require reworking any work that has been done based on erroneous assumptions. Deferring planning decisions as long as possible should result in better decisions because more information will be available at that point in time to make those decisions with less speculation.

Rolling-wave planning is not really new to agile. The general concept has been around for a long time and has been in PMBOK® for some time. However, in actual practice, the plan-driven or waterfall approach typically requires doing more of the planning up front. Prior to agile, iterative approaches have recognized the need for taking a more incremental approach to planning and building a solution. Iterative approaches also deferred detailed planning of each iteration until when that level of detail is needed.

Planning strategies

Planning is very much related to management of uncertainty in a project, and the planning approach generally attempts to remove uncertainty by more clearly defining the goals and requirements for the project. A plan-driven or waterfall project attempts to reduce the level of uncertainty associated with the project to a very low level before the project starts, as shown in Figure 4.1. That might be a reasonable approach with some projects, but it's not reasonable to force that kind of approach on a project that has very high levels of uncertainty that can't be easily reduced.

Attempting to do that on a project with very high levels of uncertainty forces you to make a number of assumptions about the requirements for the project based on very sketchy and incomplete information, and the quality of those assumptions may be very questionable. Sometimes, project managers make assumptions like that, lock them into a requirements document to define the project, and then make it difficult to change those assumptions through a formal method of change control later. That creates an *illusion of control*; on the surface, it looks like the project is very well controlled,

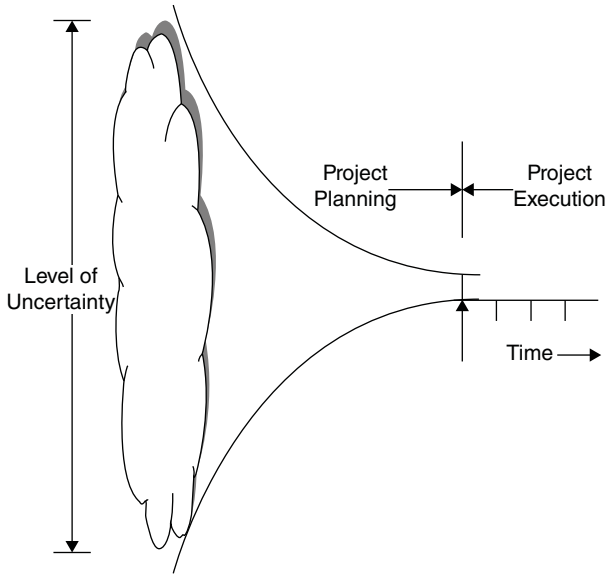

FIGURE 4.1 Typical plan-driven or waterfall planning approach

but a project can only be as well controlled as the requirements are certain. As a result, you might start the project with what appears to be a very detailed plan that is designed to control the project, and after some large number of change requests later, discover that it wasn't so well-controlled after all. An agile approach recognizes that and doesn't attempt to resolve all of the uncertainty associated with a project prior to the start of the project.

A key point I want to make is that the level of planning in an agile project can be very different, depending on the nature of the project. It's a difference between taking a highly adaptive approach and a totally plan-driven approach. The approach should be adapted to fit the nature of the project. There are a number of factors that influence which approach is most appropriate for a given project, but probably the biggest factor is the level of uncertainty in the project.

For example, if you were building a bridge across a river, it would be ridiculous to say something like, "We're going to build the first span, see how that comes out, and then we'll decide how to build the remaining spans." A typical bridge-building project would be very plan-driven and all the requirements, as well as the design of the bridge, would typically be well-defined to a fair amount of detail prior to beginning to build the bridge. On the other hand, what if there was some uncertainty in the project? Suppose that the river called for building a new kind of bridge that had never been built before, or there was some uncertainty about how stable the foundation for the bridge is? You might want to prototype and test the new design on a smaller scale somehow to remove some of the uncertainty and risk before committing to the full-scale development.

The key things to remember:

- You should always fit the methodology to the project based on the characteristics of the project and other factors such as the training and capabilities of the project team
- One of the biggest factors that influences the selection of an approach is the level of uncertainty in the project.
- It is not a black-and-white decision to be totally plan-driven or totally adaptive. There are plenty of alternatives between those two extremes.

Figure 4.2 shows the general way that an agile rolling-wave planning approach progressively reduces the uncertainty in a project as the project progresses rather than attempting to remove all the uncertainty up front. Breaking up the project into releases and iterations means that a lot of the detailed planning can be deferred until that release or iteration is ready for development (as I've mentioned, the approach need not be this sophisticated).

Spikes

Many times, there is a lot of uncertainty associated with requirements and/or what the most likely solution to those requirements should be. It's a good technique to isolate and resolve this uncertainty separately from the development effort. Letting too much uncertainty get into the development process can seriously undermine the efficiency of the team.

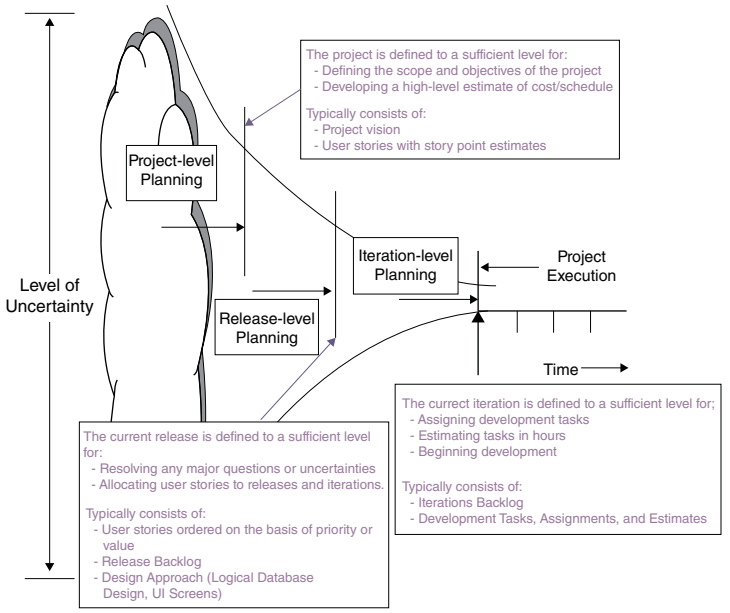

FIGURE 4.2 Agile planning approach

For that reason, a technique called a *spike* is often used in an agile project to resolve uncertainty. A spike is a special kind of iteration that is used to do research, possibly prototype a solution, and/or evaluate alternative approaches for resolving uncertainty associated with developing a solution.

Isolating and time-boxing the amount of time dedicated to resolving this kind of uncertainty can help prevent it from significantly slowing down the mainstream development effort.

Progressive elaboration

The concept of *rolling-wave planning* is very much related to the concept of *progressive elaboration,* which is an important concept in planning a strategy for the definition of requirements in an agile project. However, it is not really new to agile. Progressive elaboration has been a traditional project management practice for a long time, but agile emphasizes its use much more heavily.

The key point is that you shouldn't get too bogged down in planning requirements too far in advance because it involves too much speculation and may lead to unnecessary rework if that speculation is wrong. Requirements should only be defined to the extent needed to support whatever decisions or action is required at that particular point in time. For example:

- At the project-level, there may be a need to understand the overall scope of the project to evaluate the resources, costs, and schedule required; however, that can typically be done on the basis of a very high-level understanding of the requirements without details.
- At the release-level, there may be a need for a little more accuracy in the estimated time and effort to complete a release. That might call for understanding the requirements in a little more detail, but that also should not require a significant amount of detail.

Finally, at the sprint level, there are typically two needs that should be satisfied prior to the start of a sprint:

- The team needs to have a reasonable estimate of the level of effort associated with the requirements to determine if they can fit into the capacity for a sprint.
- The team should not take any requirements into a sprint that have major uncertainties or issues associated with them that might block or delay the development effort.

However, even at the sprint level, many details of how the requirements will be implemented can be worked out while the sprint is in progress without any significant impact on the sprint-level estimates.

Value-based functional decomposition

An important benefit of an agile approach is increased focus on business value. A good approach is to start with a vision statement that clearly defines the business value that the solution will provide. A vision statement should be short and succinct. This is an example of a format that can be used for a vision statement to keep it simple:

- For (target customer)
- Who (statement of the need or opportunity)
- The (product name) is a (product category)
- That (key benefit, compelling reason to buy)
- Unlike (primary competitive alternative)
- Our product (statement of primary differentiation)

Once a high-level vision statement has been defined, it is a good technique to use functional decomposition to break down that vision statement into the functionality that will be needed to achieve that overall vision. Functional decomposition becomes particularly important on large projects where there could be hundreds of user stories. It provides a hierarchical approach for organizing requirements, which can be an essential technique for prioritizing requirements. Without an effective approach for functional decomposition that aligns with an overall value statement, it is very easy to "get lost in the weeds" of individual requirements or user stories and lose sight of the big picture of the value you're trying to deliver.

AGILE REQUIREMENTS PRACTICES

The role of a business analyst in an agile project

A business analyst (BA) many times plays a key role in a traditional project in working with the users to analyze and document requirements prior to development. However, since agile emphasizes direct

face-to-face communications between the project team and the users as much as possible with less emphasis on requirements that are documented in detail, there is less of a need for a business analyst and the role, if any, is very different.

In an ideal agile environment, there is no role for a business analyst; the product owner is directly responsible for defining and communicating the requirements to the developers in the form of user stories. However, in the real world that is not always possible:

- For large projects, the workload on the product owner could be very large.
- For complex projects, additional analysis work may be needed to further analyze requirements and to evaluate alternative approaches.

The important points are that:

- The business analyst should not become too much of an intermediary, isolating the project team from the product owner and the business users and inhibiting communications.
- The business analyst should play a supporting role to enable and facilitate more effective communications with the project team.

Frequently, the business analyst will facilitate discussions between the developers, the product owner, and the business users to perform product backlog grooming and to further elaborate and define requirements as the project progresses.

In summary, if a business analyst is used in an agile project, the primary functions are to support the product owner by doing the following:

- *Analyze a broadly defined area and use functional decomposition to define high-level epics and stories to create a well-organized, value-driven framework to provide the required business value in the product backlog*. If the stories follow a logical functional hierarchy, it provides a mechanism for better understanding the relationship of the stories and epics to each other and for satisfying overall business goals.
- *Write individual stories that are very clear and concise and are easy to understand and implement by the project team*. Writing effective user stories is a skill that is often taken for granted. What is often overlooked in good stories is the "why" or the "so that" clause that expresses the business value the story is intended to provide. A good BA can provide that perspective that is difficult for a developer to provide.
- *Identify related user stories and epics, grouping them into a logical structure as necessary and documenting the interrelationship and associated business process flows as necessary*. The interrelationship of user stories and epics should be well-understood and the implementation of stories across different functional areas may require some planning and coordination so that they are consistently implemented. This overall framework can provide a mechanism for easily identifying any inconsistencies and/or missing functionality.

- *Integrate the needs of various stakeholders to produce an overall solution*. This is more crucial on large projects or programs, where there might be a need to integrate the needs of related projects as well as the needs of a number of different stakeholders.

"Just barely good enough"

A common problem in many traditional projects is that requirements can become bloated. One of the big reasons that happens is that users feel that if they don't ask for everything that they could possibly need and get it into the requirements, they might not get it at all. Also, we all know that in many situations in the past, project teams have gone beyond the basic requirements and "gold-plated" a solution. Excellence was perceived as going above and beyond the user requirements to develop a more complete solution, but going far beyond the user needs to develop a solution isn't necessarily a good thing and can significantly increase the scope and complexity of the development effort.

It may seem to be counter-intuitive to traditional project management thinking, but a very important principle in agile is simplicity and limiting a solution to what is "just barely good enough" to solve the problem. There is an optimum point, and beyond that point, adding additional features has diminishing value. It requires close collaboration with the users to find where that optimum point is. A good technique is to start with the simplest and most basic solution possible and then add to it incrementally and iteratively *only* to the extent that it adds value to the user.

An important element in making this work is a spirit of trust and partnership between the users and the project team. The users have to trust that if they don't ask for something specific and it is discovered to be important as the project progresses that it can be easily added at that point if necessary.

Differentiating wants from needs and the "five whys"

Another very good technique in developing requirements for agile projects is to differentiate wants from needs.

- Wants tend to be associated with a solution that a client envisions.
- Needs tend to be associated with the problem.

It is typical in many instances for a user to tell you what they think they *want* for a solution before the problem the solution is intended to solve is clearly defined. You should never lose sight of the problem to be solved before you go too far into implementing a solution. It is also a good practice to dig a little deeper into the problem to be solved to get to the root cause of the problem to be solved.

The *five whys* method is a good approach for digging into a customer need to get to the root of the problem. The idea is that by progressively asking "why" over-and-over again, you eventually get to the root cause of the problem. The following shows an example of how the "Five Why's" technique can be used to get to the root of a problem:[1]

[1] "5 Whys: Getting to the Root of a Problem Quickly," MindTools.com, http://www.mindtools.com/pages/article/newTMC_5W.htm

1. *Why is our client, Hinson Corp., unhappy?* Because we did not deliver our services when we said we would.
2. *Why were we unable to meet the agreed-upon timeline or schedule for delivery?* The job took much longer than we thought it would.
3. *Why did it take so much longer?* Because we underestimated the complexity of the job.
4. *Why did we underestimate the complexity of the job?* Because we made a quick estimate of the time needed to complete it, and did not list the individual stages needed to complete the project.
5. *Why didn't we do this?* Because we were running behind on other projects. We clearly need to review our time estimation and specification procedures.

MoSCoW technique

Another very good technique for prioritizing requirements is called *MoSCoW,* which is defined as follows:[2]

- *Must have*—Requirements labeled as "*MUST*" have to be included in the current delivery time box in order for it to be a success. If even one "*MUST*" requirement is not included, the project delivery should be considered a failure.
- *Should have*—"*SHOULD*" requirements are also critical to the success of the project, but are not necessary for delivery in the current delivery time box.
- *Could have*—Requirements labeled as "*COULD*" are less critical and often seen as *nice to have*.
- *Won't have*—"*WON'T*" requirements are either the least-critical, lowest-payback items, or not appropriate at that time.

An example of how this approach has been used successfully is the case study of General Dynamics, UK, which is discussed in Chapter 21 of this book. In that situation, General Dynamics, UK, was faced with the challenge of managing a large fixed-price government contract, and the only way to do it successfully was to work with the government customer to prioritize the requirements and eliminate some of the requirements that weren't absolutely essential to the solution using the MoSCoW method.

USER PERSONAS AND STORIES

User personas

A *user persona* is used in agile for personalizing user requirements; however, it really is a general process that can be used in any kind of development effort. The idea behind a *user persona* is that many

[2] MoSCoW Method, http://dsdm.org/content/10-moscow-prioritisation

traditional requirements management practices are impersonal; they result in a large documentation with lots of specific requirements that the system must meet, but it is not clear who those requirements are designed to satisfy. Agile puts a strong emphasis on providing value to the user and also has a strong emphasis on engaging the user directly in the project to provide feedback and inputs as the project progresses. Organizing the project requirements around specific users and their needs puts more focus on really understanding the value that the project provides and *who* the recipient of that value is.

To facilitate that process, it is useful to identify *user personas* as specifically as possible so that the project team can target their development efforts at the needs of those specific users. A user persona could be a specific category of user or it could even be a specific user, but in either case, it is useful to model that user's personality and specific interests as a hypothetical user persona. A user persona helps the team visualize that user and focus its efforts around the user's needs.

The following is an example of a user persona:[3]

User: Fred Fish, Director of Food Services "Get me out of the office and into the kitchen"

Background: Fred is director of foodservices for Boise Controls, a mid-sized manufacturer of electronic devices used in home security systems. He uses a computer, but he's a chef by trade and not so *computer-savvy*.

Key Goals: As a manager, Fred doesn't get his hands (literally) dirty the way he used to. He stops in at all the Boise Controls sites and sticks his fingers into things once in a while to stay in touch with cooks and cooking. He wants to learn computer tools but not at the expense of managing his kitchens. A computer is just another tool for getting his administrative tasks done.

A Usage Scenario: At the start of every quarter, he meets with the head chefs and plans out the next quarter's menus. That's one of his favorite things, because each chef gets to demonstrate a new meal. The chefs spend time in the kitchen exploring each new dish. When they're done, he sends the food to his staff and his manager.

He's not a computer whiz. On a good day, he can drag in some clip art and do some formatting with fonts. Once in a while, he'll format menus with the new editor on his MacBook Pro.

User stories

User stories are a succinct way of defining requirements in agile. Telling user stories is a way of simplifying the definition of the requirements in a language that can be easily understood by both developers and users. It breaks the requirements into small chunks of functionality that can be built incrementally. User stories follow the general format shown below:

As a <role> I want <to be able to do something> so that <benefit>

[3]A Sample Persona, http://creativebeatle.files.wordpress.com/2010/12/sample-persona-from-interaction-design.pdf

A user story consists of some standard parts:

- The first part identifies *who* the actor (or role) is that needs to do something. This should be as specific as possible to clearly identify who the story needs to satisfy.
- The next part identifies succinctly *what* the user (role) needs to do so that the end result is very clear.
- The third part is the "so that" clause. This clause provides some insight into *why* this functionality is needed. Knowing this additional insight should help the developers get a deeper understanding of the user's need. (What business value does it provide?)

Here are two examples:

As a student I want to purchase my monthly parking passes online so that I can save time.

As a student, I want to be able to pay for my parking passes via a credit card to avoid using cash.

There are several key advantages of a user story:

- It provides a standardized and concise way of defining a requirement in simple English that is easy for everyone to understand (both developers and users).
- It encourages defining requirements in small, bite-sized chunks where the functionality to be developed is clearly defined and can be completed within one sprint.

An important aspect of user stories is that they are written in functional terms—they define an expected result and don't tell the developer *how* to design the software to achieve that result.

- User stories are intentionally designed to be brief and don't typically provide a detailed description of what is required. They are intended to be a "placeholder for a conversation."
- That detail should take place through face-to-face communications; however, many times acceptance criteria are included in user stories to more clearly define the expected result.

There's a mnemonic that is well-known in agile called "INVEST" that is a useful way of defining the essential characteristics of good user stories.[4]

Independent: Stories should be as "independent" as possible so that they can be worked on in any order. That will simplify the flow of stories through development and avoid bottlenecks that can be caused by having too many dependencies among stories.

Negotiable: Stories should be "negotiable"—a story is a placeholder for conversation, and some dialog is expected to take place to explore trade-offs associated with developing the story as efficiently and as effectively as possible.

[4] Bob Hartman, "New to Agile? INVEST in Good User Stories," Agile for All, May 14, 2009, http://www.Agileforall.com/2009/05/14/new-to-Agile-invest-in-good-user-stories/.

Valuable: Stories should be "valuable"—the value that a story is intended to produce should be clearly-defined so that the product owner can make an objective evaluation of the level of effort required versus the value to be gained from the story.

Estimable—"Estimable" is the next important characteristic. A story needs to be sufficiently defined so that the team can develop a high-level estimate of the effort required for the story in story points.

Small—Stories should be relatively "small" so that functionality can be developed and tested as incrementally as possible. Breaking the work into small chunks allows it to flow much more smoothly and allows the work to be distributed more evenly among the team while large efforts can easily lead to bottlenecks and inefficiencies in distributing work among the team.

Testable: Finally, stories should be "testable" to determine if they have successfully fulfilled the value proposition that they are intended to fulfill. It's a good practice with agile stories to write acceptance test criteria, along with the stories to define the tests that they need to fulfill. The test criteria essentially take the place of more detailed specifications for what the story must do.

Epics

An *epic* is basically a very large user story. An epic serves the purpose of associating related individual user stories with a higher-level purpose that they are collectively intended to fulfill, but an epic is normally too large for the project team to work on directly without breaking it down into individual user stories.

It's a useful technique on large, complex projects for organizing user stories into some kind of structure so that the interrelationship of user stories is well-understood. The following shows an example of a large epic and how it can be broken down into smaller stories.

Epic:

> *As an electronic banking customer, I want to be able to easily make an online deposit of a check into my bank account through my IPhone® so that I can save the time required to send a check for deposit through the mail and I can have the money immediately credited to my checking account as soon as the deposit is completed electronically.*

Stories:

- As an electronic banking customer, I want to be able to scan an image of the front and back of a check into the IPhone® so that it can be deposited electronically.
- As an electronic banking customer, I want to be able to enter the deposit information associated with an electronic deposit so that the correct amount will be deposited into the correct bank account when the electronic deposit is processed.

- As an electronic banking customer, I want to be able to electronically submit a scanned check and deposit information to the bank for deposit so that I can save the time associated with sending deposits by mail.
- As an electronic banking customer, I want to be able to receive confirmation of a completed electronic deposit so that I will know that the deposit was successfully processed.

PRODUCT BACKLOG

What is a product backlog?

The product backlog was previously discussed in Chapter 3. It typically consists of user stories, and it is dynamic. The user stories are continuously groomed and prioritized over the course of a project. It is essentially a queue of work to be done. There are different ways to implement a product backlog and there are different schools of thought on how it should be done.

At one extreme, there is a view that the project should be totally adaptive and you shouldn't try to define the product backlog any more than two to three sprints into the future. That approach might work fine for some projects, but a problem with that approach is that it doesn't provide a basis for planning the overall scope, costs, and schedule of a project. If there are some expectations about the cost and schedule of the project, you may have to define the product backlog more completely, at least at a high level to evaluate the scope of the project to support those estimates. However, the ability to do that is obviously related to the level of uncertainty in the project.

You should develop an approach for planning the requirements and managing the scope of the project that is appropriate for the project. I would say that the most significant factors in selecting an approach are:

1. The level of uncertainty in the project and the difficulty of defining the requirements for the project upfront, and
2. The need for estimating some kind of schedule and costs for the project.

Figure 4.3 shows the flow of information into the product backlog. Product owners are responsible for defining and maintaining the product backlog; however, they might be assisted in that effort by the Scrum Master or others.

Product backlog grooming

There is a very general approach for thinking about how to organize the product backlog. Think of it as an "iceberg" in which the items in the iceberg gradually make their way to the top as they become ready for development, as shown in Figure 4.4.

The principles of *rolling-wave planning* are used and items start out at the bottom of the backlog as being very roughly defined and the stories are progressively "groomed" as they move closer

FIGURE 4.3 Product backlog flow

Courtesy of Rally

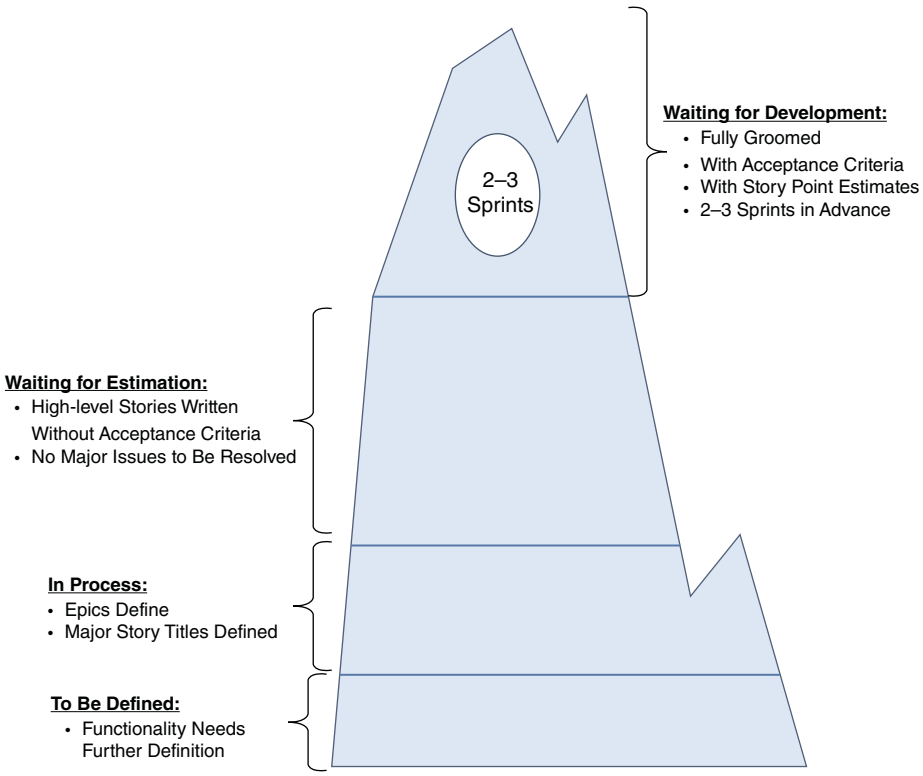

FIGURE 4.4 Product backlog grooming flow

to development. Grooming is an important part of any agile project but it is often overlooked and not planned for as much as it should be. A good technique is for the project team to allocate some time in each sprint so that the queue of stories to be developed never runs empty.

Agile is based heavily on "just-in-time" planning—obviously, you never want a project team waiting around for stories to be developed so you have to manage the grooming process to get stories ready for development as needed, but on the other hand, you don't want to work any farther in advance than necessary, because some of that effort might only need to be repeated once the stories are closer to being ready for development.

It is perfectly reasonable to develop a high-level backlog far enough out in time to develop an estimate of the scope, costs, and schedule of the project. It is a judgment call of how far it is reasonable to try to plan into the future. Some will say that it's a waste of time to plan any more than two to three sprints in advance, but if you have a need to estimate the costs and schedule of a project, you have to define the product backlog further out in time at a sufficient level to support those estimates and then elaborate the details later.

SUMMARY OF KEY POINTS

1. Agile Planning Practices

There is a common misconception that agile projects are not planned at all. That is not the case, it is just a different kind of planning, and the planning is spread out over the project rather than attempting to do the majority of the planning upfront prior to the start of the project. Rolling-wave planning and progressive elaboration of requirements are both used to plan agile projects.

The planning approach is very much related to the level of uncertainty in a project and the planning approach attempts to reduce the level of uncertainty to an acceptable level based on the nature and complexity of the project. The level of planning in an agile project can vary significantly depending on the project and it is always best to fit the planning approach to the nature of the project and the level of uncertainty associated with it.

2. Agile Requirements Practices

There is no defined role for a business analyst in an agile project as there is in a plan-driven or waterfall project and the product owner is ultimately responsible for defining and prioritizing the project requirements; however, the product owner might be assisted in that role by a business analyst. If a business analyst is engaged in an agile project, the BA should not become too much of an intermediary and isolate the project team from the product owner and users and inhibit communications—the BA should play a supporting role and facilitate more effective communications with the project team.

Prioritization of requirements is extremely important in an agile project and requires a spirit of trust and collaboration between the business users and the project team to work together to deliver

the highest-value items first and to preserve simplicity of the solution and avoid going beyond what is needed to satisfy the fundamental business need.

3. User Personas and Stories

   A user persona is used in agile for personalizing user requirements; however, it really is a general process that can be used in any kind of development effort. User stories are a succinct way of defining requirements in agile. Telling user stories simplifies the definition of the requirements in a language that can be easily understood by both developers and users. It breaks the requirements into small chunks of functionality that can be built incrementally. User stories follow the general format: *As a <role> I want <to be able to do something> so that <benefit>*. An epic is a large user story.

4. Product Backlog

   The product backlog is a very important tool in an agile project for maintaining a prioritized queue of project requirements. It should be dynamic and should be continuously groomed as the project progresses. Agile projects generally use an iceberg strategy for grooming the product backlog. The items that are near the top of the iceberg and are closest to going into development should get the most attention. There should typically be about two to three sprints worth of stories at the top of the backlog that are well-groomed and ready to go into development in order to avoid a situation where the project team is waiting for work to do. However, the product backlog grooming does not need to be limited to two to three sprints in advance, and it is a judgment call of how far it is reasonable to plan into the future.

DISCUSSION TOPICS

Agile Planning Practices

1. What is rolling-wave planning? How is it different from a traditional project to an agile project? Why would it make sense to use rolling-wave planning?
2. What are the levels of planning that you might typically find in an agile project? What is the role of each?
3. What is progressive elaboration, and why does it make sense?
4. What is a spike, and how is it used? Provide an example.

Agile Requirements Practices

5. What is the role of a business analyst in an agile project? When would a BA typically be needed and why?
6. Explain the concept of value-based functional decomposition and how it would be used. Provide an example of a real world project and how that concept would be used in the project.

7. What is the "five why's" technique, and how would it be used? Provide an example of how it might be used from a real world project.
8. What is the MoSCoW technique, and how is it used? Provide an example of how it might be used.

User Personas and Stories

9. What is a user persona? Give an example of a user persona in a project. How does it help to define the requirements?
10. What are the advantages of using user stories to define requirements in an agile project? Write an example user story for a project that you have been engaged in.
11. What are the characteristics of well-written user stories? Provide an example of a user story that is not well written and that violates these characteristics.

Product Backlog

12. What purpose does the product backlog serve? What are the characteristics of a well-organized product backlog?
13. Who is responsible for the product backlog in an agile/Scrum project?