Agile Practices for Waterfall Projects

Shifting Processes for Competitive Advantage

Barbee Davis, PMP, PMI-ACP, PHR

5

What Are Some More Agile Concepts?

Favor the Simple Over the Complex

Some critics of the Agile process say, "Well, wait a minute. If we don't list *all* the features we want at the very beginning and get a commitment from the team to do them, some things may take longer to complete than expected and we'll run out of time and won't get as many features as we want for our money." However, do they really need all of those features?

Another tip in the book *97 Things Every Project Manager Should Know* is entitled "Favor the Simple Over the Complex" and is contributed by Scott Davis, who humorously, but thought-provokingly considers the same question of the necessity of extensive lists of features, stating:

> "As far as I'm concerned, my microwave oven only has one button: 'add a minute.' To boil a cup of water for my coffee, I press the button three times. To melt cheese on my crackers, one click. To warm up a flour tortilla I press 'add a minute' and then open the door after 15 seconds.
>
> Would a one button microwave oven even make it out of the planning committee on a traditional project? Probably not. I can tell by the never-used features on my microwave that the committee who designed it favored complexity over simplicity. Of course, they probably cloaked "complexity" in the euphemism "feature-rich." No one ever starts out with the goal of making a product that is unnecessarily complex. The complexity comes along accidentally.

> Suppose I have a slice of cold pizza that I want to warm up. According to the manufacturer's directions, I should press the "menu" button. I am now faced with the options "speedcook" or "reheat." (Um, reheat I guess, although I'm kind of hungry. I wonder if speedcook is any faster than reheat?)
>
> "Beverage," "pasta," "pizza," "plate of food," "sauce," or "soup"? (I choose pizza, although it does have sauce on it, and it is on a plate.)
>
> "Deli-Fresh" or "Frozen"? (Neither, actually—it's leftover delivery pizza. I'll choose, Deli/Fresh, I guess.)
>
> "1 slice," "2 slices," "3 slices," or"' 4 slices"? I have no idea how much longer this interrogation will last, so I press "Cancel," and then the "add a minute" button.[1]

So, Scott's thinking is that although projects generally solve complex problems, they often erroneously do it by creating equally complex *products*. The question is, how much of that inherent complexity is really of value to the end user?

There is an interesting study done by the Standish Group, the research organization that is centered on collecting statistics of why projects succeed or fail. They found out that of all the successfully delivered features—all features that the customer put into long requirements documents of traditional Waterfall projects—45% of them were Never used by the end user anyway. An additional 19% of the features fell into the Rarely used category. Totaling those percentages, as shown in Figure 5.1, and you'll realize that 64% of the features could have been eliminated, and no one would have cared.[2]

The Always, Often, and Sometimes used features total 36%, just slightly over one third of the features created. So, if a customer has you produce features in the order of importance, they will get total usability at the end of the project with workable, well-tested software or products, even if you don't have time to deliver the 64% of the requested features no one will ever touch anyway. But, the ones you do deliver are the important ones, and they are ready-to-use to garner business value.

Scott also raises the question of whether your project deliverable is a complexity amplifier. He poses the idea that great software, products, or services are generally a complexity "sink." They bear the brunt of the problem on behalf of the user and provide a clean, streamlined, and simple solution rather than amplifying a problem. According to Davis, simplicity does not happen accidentally. Rather, it needs to be cultivated. Complexity is what happens when you are not paying attention.

As a project manager, you need to ask yourself, regardless of the type of project you work on, if you, personally, are a complexity sink or a complexity amplifier. The best project managers who work in an Agile manner assume a role whereby

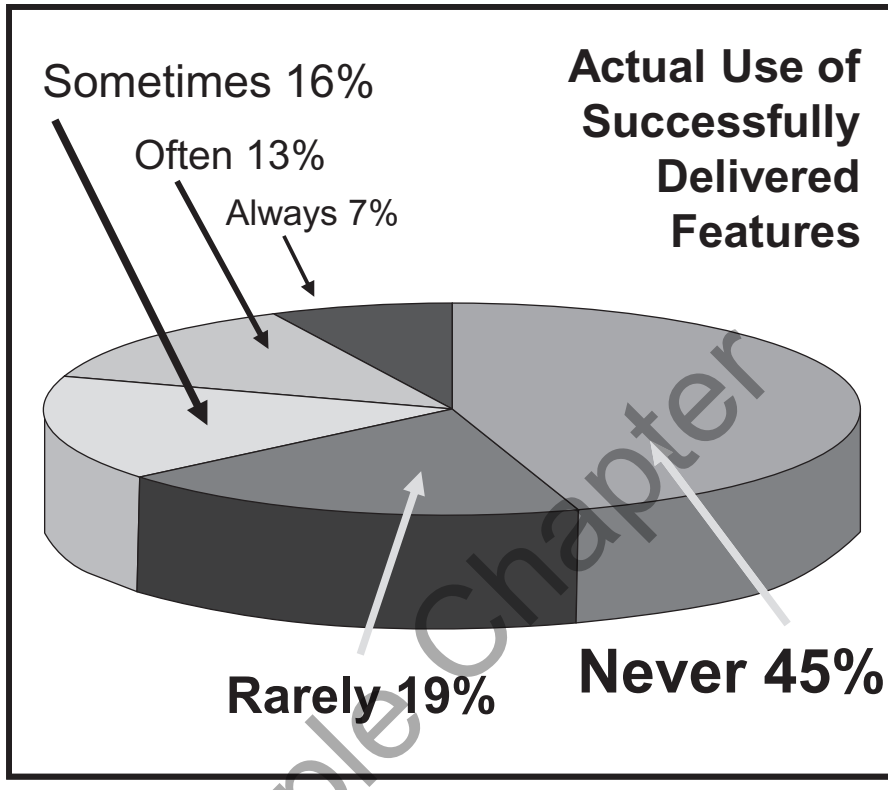

Figure 5.1 Seldom Used Features

they absorb complexity from all sides—from the project team, end users, product owners, customers, and the management of the organization. They take it upon themselves to remove the constraints and obstacles in the way of the team's performance and make the path smooth. They never amplify the problems facing them, even if it makes a great story over coffee.

The old traditional project management pattern is gathering requirements, designing the solution, developing or creating the solution, and then testing it, as shown in Figure 5.2. The newer Agile model suggests that you start with a list of features or user stories the customer has prioritized, let the team choose how far down the list they can commit to for each timebox, and move those items over to become iteration features. Once the team begins creating the actual project work, it is a cycle in which they plan and develop, evaluate their work, learn from their successes and errors, and keep the cycle going until at the end of the iteration they

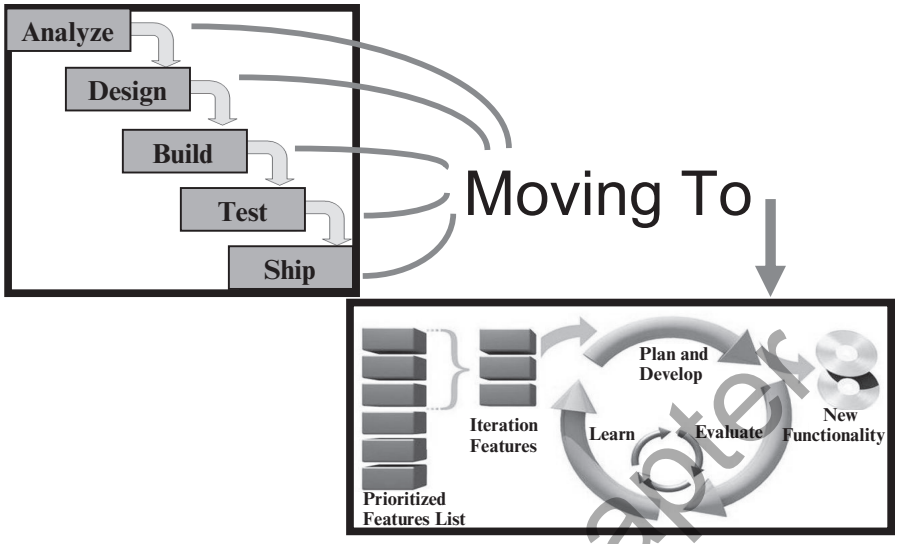

Figure 5.2 Traditional Processes Moving to Agile Processes

have a piece of product, service, or software that can be put to use for business value.

Another way to describe the cycle is to figure out the first thing the user wants, build a small portion of it, and make sure you are on the right track. Then, check with the user to see if he or she would like it altered or adjusted, and continue that process until the user is happy. Remember, it is not just doing actions or activities that "look" Agile; it is about embracing the philosophy behind Agile and using it in a way to achieve greater *business* value by creating greater *customer* value. It's about moving the software, product, or service, through the project team process flow with a minimum of impediments or constraints.

Phase-to-Phase Relationships

Many people continue to think Agile is new, rather than the more than 12 years old it really is. They may also think that these heretical processes are "against the rules" set out by PMI, despite the obvious misconception that PMI "sets out rules," rather than sharing effective practices found and contributed by their members. The acceptance of Agile project management has been in the *PMBOK® Guide* since the Third Edition was published in 2004. In the Fourth Edition, which came out in 2008, there were three basic types of phase-to-phase relationships:

- **Sequential**, where *one phase must finish before the next begins*. This is what we have come to know as Waterfall.
- **Overlapping**, in which *the next phase may start before the preceding one is completed*. If you speak project management, you will be familiar with this idea that we frequently use in attempts to bring projects back on track with the techniques of schedule compression or fast-tracking.
- **Iterative**, which sounds exactly like a description of Agile: "*… an iterative relationship, where only one phase is planned at any given time and the planning for the next is carried out as work progresses on the current phase (iteration) and deliverables.* The scope is then managed by continuously delivering increments (small parts) of the product and prioritizing requirements to minimize project risk and maximize product business value."[3]

At a high level, a Waterfall project might look like the top of Figure 5.3. In this example, the Design Phase is broken into several Sub-Phases. The length of time planned to complete the Design Phase is the total of the three Sub-Phases; however, the phases as the higher level such as Planning, Execution, and Test can vary in length from one to the next. One might be two weeks long and another three months. The Sub-Phases can also be quite different lengths in the time it takes to complete each one.

In the lower portion of Figure 5.3, you can see an Agile project sample. For purposes of comparison, you might like to think of a Release as being equated to a Phase in the Waterfall diagram. As you can see, the Release is also broken into smaller parts called Iterations. The difference is that the Iterations will all be equal in length. That length can be 1 week, 2 weeks, or up to 4 weeks, but if you choose 4 weeks for the iteration timebox, each and every Iteration must also be 4 weeks. So, in the example, if we plan a 4-week Iteration, the first Release will be 16 weeks (four months). The entire project would take a year, since there are three Releases.

The Non-Software Agile Process

There are many books on Agile for the software industry, but for those of us who are not software developers, they use many terms and processes that do not translate to the non- IT world. For our purposes, an overview of a non-software approach to Agile would be more useful. Almost all people retain information best through pictures, so in its most simple form, a non-software Agile process would look something like the bottom of Figure 5.3.

As with all good project initiations, a non-software Agile project starts with a company vision that can be achieved through the success of the project. *Someone who knows and understands that vision, and has some personal interest,*

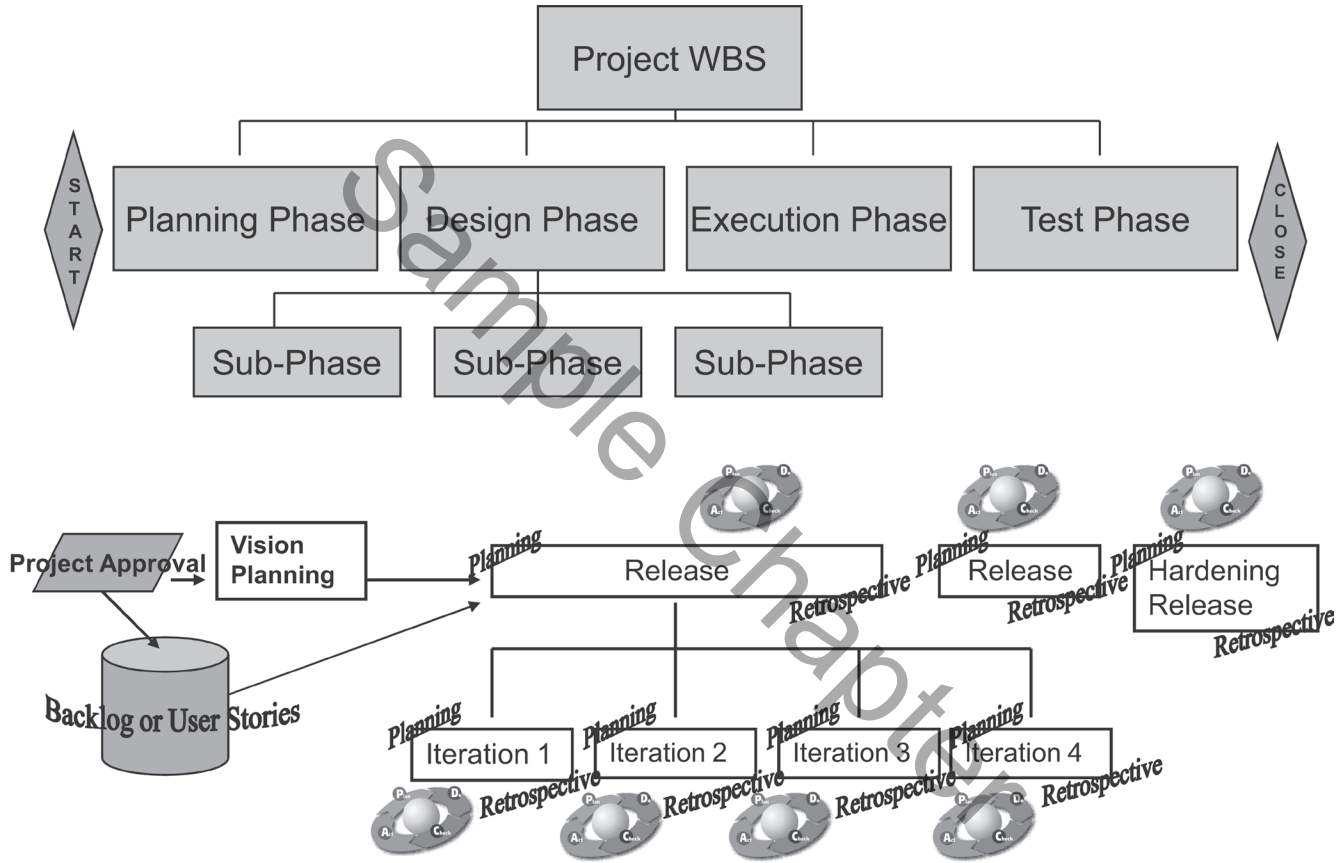

Figure 5.3 Waterfall and Agile Project Structures

involvement, or responsibility for the outcome, needs to commit to be the management link to the project team and the project manager. A common title to designate that role in the project is **product owner**. You may commonly see it written as Project Owner to show that this is a specific Agile role with distinct responsibilities.

In different organizations, the title of the person who will take on the role of the product owner will vary. For a small organization, it could be the owner. In a larger company, it might be a department manager, and in an even larger enterprise, it might be a program or portfolio manager. When the infrastructure has many levels, there may be an intermediate representative who reports to that person with the ultimate responsibility, who is chosen to be the project liaison. Perhaps it is a business analyst. If this project is for an external group, you may hear the person who performs the product owner responsibilities labeled the Customer, with a capital C.

Ideally, the product owner or customer must be prepared to be available to the team on a daily basis, and preferably they will be collocated with the team. The idea is that they are there to guide the team and answer questions that occur as the work of the project progresses. Having immediate feedback keeps the team from creating incorrect choices, which, as we saw in the bicycle hub flange example in Chapter 4, can add time and cost to the project.

At the start of the project, the product owner writes stories of what they want in this product or service, rather than create a long list of technical and functional requirements. This is typically done with the help of the project manager. Again, in its attempt to deal with the realities of the world, Agile approaches acknowledge that while the product owner knows best what he or she hopes to gain from creating this new deliverable, this may not be a person who also knows the details of *how* it can best be created. So, that is left to the project team.

When the scope of a project arrives on the desk of team members in the form of a 210 page, single-spaced technical and functional requirements document, it is hard for them to see the vision that was intended when the project was originally approved and funded. But if they attend a meeting where the product owner tells them in person what this product should do, why it is being proposed at this moment in the company's lifecycle, and how it needs to perform to make the work or life of the end user/final customer easier or better, the team member is better prepared to share this vision and bring it to life. A Vision Statement of what this project is intended to do should be posted on the wall of the team room to keep it constantly in focus.

User Stories

In the Agile world, technical and functional requirements are written in a way that is easy to understand. They are *descriptive, short explanations of a feature* called **user stories**. There is no one way to create user stories, but a successful sample is shown in Figure 5.4. It is written to give the team the knowledge of who the end user or beneficiary of the feature would be, and what need it would meet. A common way to write user stories is:

> As a <Insert the Role of the Ultimate Person Who Needs This Feature>, I want <Insert the Functionality/Feature/Service that the Ultimate User or Person Wants> to <Insert the Business Value or Benefit to the Customer/ Organization>.
>
> This is usually shortened to:
>
> As a <*Role*>, I want <*Functionality*> to <*Business Value or Benefit*>.

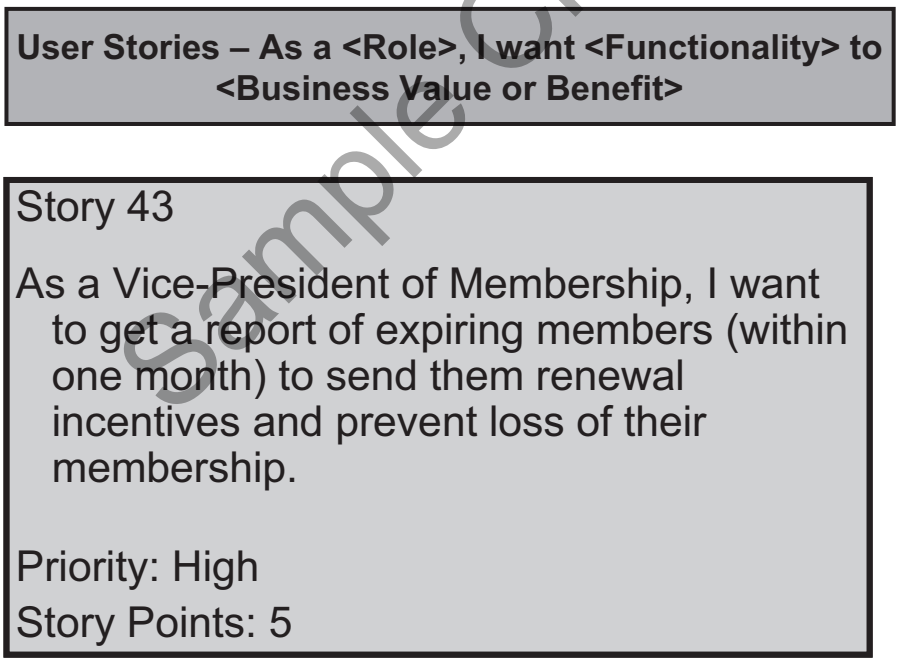

Figure 5.4 A Sample User Story

Imagine that you work for a professional organization that represents the security industry. Your funding is based on membership dues, and without a robust enrollment, you will not have enough money to provide the services that the security professionals have come to count on. As part of a project to fulfill the vision of the organization to support its members, you might fund a project to retain and increase memberships. One of many user stories you, as the product owner, might write to share with your project team, could read:

> As a <Vice President of Membership>, I want <to get a report of expiring members (within one month prior to expiration)> to <send them renewal incentives and prevent loss of their membership>.

If you were on this team, you would clearly understand what needed to be done. (You don't need to include < > marks. They just help to show where the template and the sample fit together.)

By the way, all joking aside, you might want to be sure that your team did not misunderstand the word "expiring" and think that you wanted a list of dead members. But that discussion might prompt the addition of another user story to read:

> As a <Vice President of Membership>, I want <to get a report of deceased members (within one month)> to <remove them from mailing lists or deduct that projected income from the balance sheet, or activate their insurance settlement paperwork>.

Each of those needs could also be a separate story. You can find a user story template on the Web Added Value section of the publisher's website (www.jrosspub.com).

Some teams prefer to get very specific with their descriptions of the end users, developing several pages for each **persona**, or *character they are going to use to represent the ultimate customers*. They tend to describe jobs, hobbies, families, interests, and other details for each person. Then, when discussing a feature, they have a communication shortcut to say, "How do you think Ethel would like this?" (a grandmother persona that has little experience with technology), or "Would these colors be an attraction for Jason?" (the sports minded professional persona with four children).

In order to know when each user story is done, it needs to have pre-planned approval or acceptance criteria. These will be written or stated by the product owner and captured by the project manager or the team. **Acceptance criteria** *allow the person working on the feature or service to know when this user story is finished, that is, when it meets the standards or requirements set out for its approval*. For example, acceptance criteria for a new printer being created might be that it prints two-sided documents at speeds of 7.4 pages per minute in black

and 5.4 pages per minute in color, based on the International Organization for Standardization (ISO) standards for measuring print speeds.

Traditionally, these criteria come in the technical requirements document. Now, we are going to find them out from the product owner, or perhaps the team has the authority to suggest some themselves. Regardless of the lighter form of documentation preferred in Agile approaches, there is no dishonor to having product requirement specifications on a list. In an Agile project, it is incredibly important for the product owner to spell out what he or she expects, so the team is always clear as to what they need to accomplish.

The difference is in how requirements are kept in front of the team as they work. One workable method is to have the acceptance criteria written on the back of the same sticky note that contains the user story. At any time, any of the stakeholders on the team can flip over this project artifact and see the technical goal for the item. So, when the printer meets the ISO standards for speed set out in the acceptance criteria, we can move this sticky note to the "Done" pile. Because there are two color speeds to achieve, perhaps the speed requirement will be broken into two, separate user stories.

Each project will have a number of stories, and *all of those stories that describe product, service, or software features are placed into an imaginary bucket* called a **product backlog**. Now the customer or product owner prioritizes them by their importance, business value, and risk. Those at the top of the list will be completed first. In the earlier membership example, we might decide that the need to contact existing people whose memberships are due to expire is a story that belongs at the top of the list, since it will be easier to retain members than attract new ones. It also brings money into the coffers earlier if we notify them ahead of time, rather than waiting for the membership to expire (reach its renewal date) before reminding current associates to rejoin.

Let's switch examples, and look at how this approach might compare to a traditional project. In Figure 5.5, we see that this project has a Company Vision to "Be the online bank of choice to small business customers." The Specific Company Goal is to "Increase customer retention on our website." This Company Vision and the Specific Company Goal could work for either a Waterfall or an Agile approach. At the third level, where, in a traditional project, the team could expect to receive technical and functional requirements; instead the Agile team gets an **Epic** story, *a large umbrella collection or group of related user stories*. In our example, the Epic story is: "Add a self-service Customer Center for frequent customer needs."

Now, the *Epic story is broken down into* **Features**, telling what the customer will be able to do that provides value to the customer. Using an online bank, a customer might find value in the ability to access past statements, stop payment on a check, and find a branch location to provide the services that cannot be provided by the online functionality. The Features are now divided into Stories

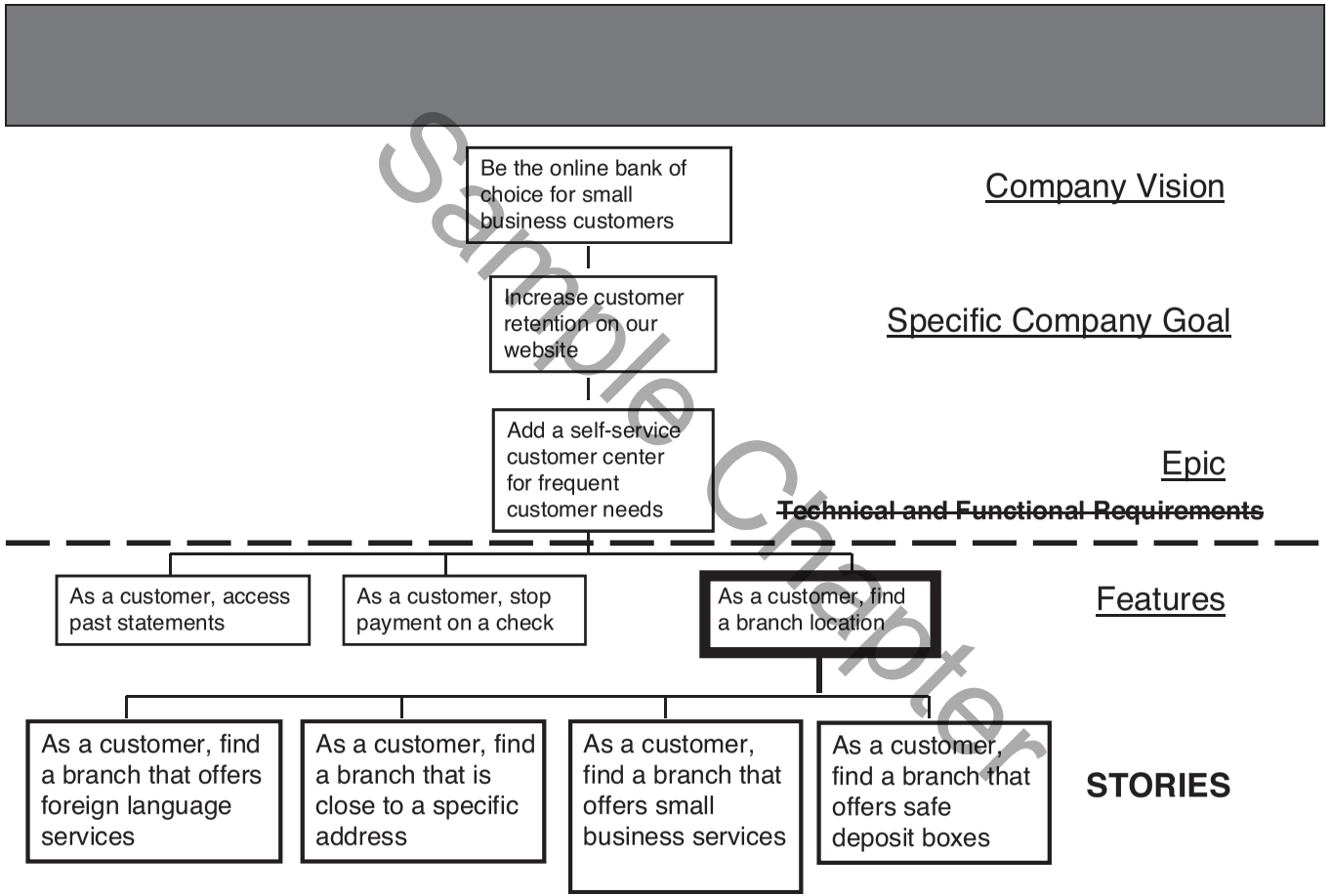

Figure 5.5 A Sample Bank Project

(user stories), showing that the customer may want to find a branch with foreign language services, close to their home or office, a location with small business services available (since all branches do not necessarily maintain identical offerings), and pinpoint which branches offer safe deposit boxes. If the Stories are too large to do in one iteration or as one unit, or some parts deliver more business value than others, or some parts have differing levels of importance, they can be further separated into **tasks**. At the task level, the work of the project is in *a small enough piece for the team members to assign story points, based on the difficulty and ability to complete in one iteration*.

Notice that we deliberately do not tell the team *how* they are to do what is asked of them. In an Agile approach, control of the stories of *what* is needed and *when* it will be done rests with the product owner or customer. *How* those needs are met is in the hands of the team (within reason). Obviously, if the need is for a faster route between Omaha and Sao Paulo, Brazil, an airline team will be expected to think in terms of airplanes, not dog sleds.

The subject matter experts and technical experts from the team can also add infrastructure stories that need to happen to support the other stories, or add any additional work they can see from their own detailed knowledge of what it will take to complete these stories. They work with the product owner to get the team stories integrated into the prioritized line of work in the backlog.

Here is an example that may help you visualize how an Agile project would work with user stories rather than the traditional WBS approach. The dotted line in Figure 5.6 shows the process flow, or the project value stream. It maps how the software, product, or service moves from Company Vision to completion, and out into the marketplace or into internal departments. Since we are looking for a value stream with the fewest possible constraints or impediments, the project manager is going to get those out of the way for the team, who is doing the work of the project.

Remember, ideally we have a team all working in one room and they have cross-functional (overlapping) skill sets. We do not want to have only one expert on anything because it would delay the process flow if that one team member was busy on another task. Also, this team is dedicated 100% to this one project. Ahead of time, the project manager has worked with the product owner or customer to figure out what features this project is to produce, broken them down into stories (Customer User Stories), and written the stories on "sticky notes". Usually, the notes are in a format like, "As a customer I want to be able to search for a product on the web." These stories make up the product Backlog, shown as the large bucket. It has been prioritized by the product owner/customer based on which items would provide the most business value, and any additional, necessary work has been added by the team and worked into the prioritization.

For the first **Planning meeting**, shown as a vertical pillar labeled Planning, the product owner or customer is right there with the team face-to-face to present

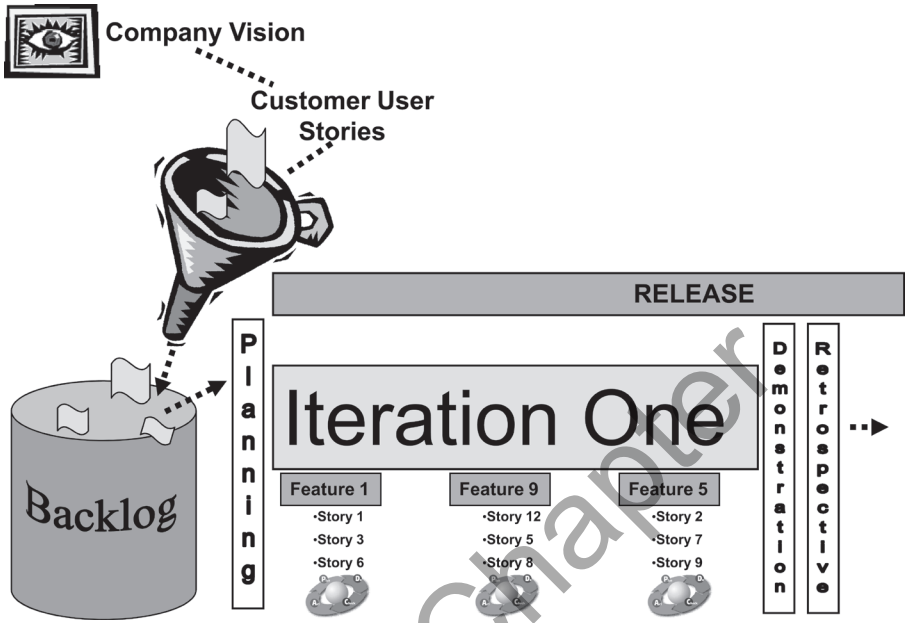

Figure 5.6 Sample Agile Process Flow

what he or she wants, tell the vision in person, and plan together roughly how long this project is going to take. This is without doing a WBS or getting into great detail. The group may say, it looks like we could get most of this done in about 9 weeks. So, we start out planning for 8 weeks and decide to work in 2 week increments.

Progressive Elaboration

Once there is a prioritized Backlog, we plan at a high level for a Release, the longest horizontal rectangle that you can note. There is a misconception regarding Agile that since it advocates lighter documentation, it is an *ad hoc* process that eliminates planning. Not true! It is just based on the reality that because of unknowns, or information that we do not know at this moment but may know more about later, it is best to *plan with broader strokes far out into the future and then add more detail just before doing the work.* This lowers the risk to the project and lowers the cost of change. This technique is called **progressive elaboration**.

A form of progressive elaboration, known as **rolling wave planning**, is also referenced in the *PMBOK® Guide* as, "*Planning where the work to be accomplished in the near term is planned in detail, and future work is planned at a higher level*" in the WBS.[4] Therefore, work can exist at various levels of detail, depending on where it is in the project life cycle. For example, during early strategic planning, when information is less defined, work packages may be decomposed to the milestone level. As more is known about the upcoming events in the near term it can be decompressed into activities. That exact philosophy is being followed in the Agile planning cycle. The Release shown in Figure 5.6, for example, might be eight weeks long. That is how long we anticipate the project will take, based on the high level look at the Customer User Stories in the Backlog. Now, we do some detailed planning for the next two weeks and decide how many features, divided into user stories we can create or produce during that timebox. In this iteration, we are going to do Feature 1, and Stories 1, 3, and 6, which will help complete that feature. We will also commit to complete Features 9 and 5, with the most important stories for each of them. Notice that we do this detailed planning for the iteration, which is set at two weeks just before we begin to do the work. You can see where this occurs by locating the vertical Planning pillar before Iteration One.

We then do as much of the work as we can during Iteration One, and put any leftover stories we have yet to complete back in the Backlog bucket. At the end of the timebox, we *demonstrate what we have created to the product owner/customer* (**Demonstration**), get his or her feedback, and hopefully get approval. This is sometimes referred to as, "Demo, not memo."

Now the product owner/customer chooses the next set of stories in importance or business value and the team gets to make a case to add any architectural or other supporting technical stories that need to be added to the backlog. We then begin the detailed planning for the next iteration, Iteration Two, which includes any changes we agreed upon during the Retrospective on Iteration One.

The final part of an iteration is to take time for the team to talk together about how the work went the last two weeks. This talk is called a **Retrospective**. Through this meeting, we get feedback from the team as to what we need to change for the next iteration. *Is there a better way for the team to do things in the way that we create this product or service*? *Also, teams talk about their processes*. Is there a better way to run the daily stand-ups, communicate, and make our progress visible? Do we need to change our team rules? This is continuous improvement in action, because we are going to improve both the product and the team's processes, or rules, at the end of each iteration. Retrospectives are far superior to capturing people's grumblings in a lessons learned meeting after 6 months or a year, and then filing them away, never again to see the light of day.

Be sure to review the project's vision statement and other organizational goals to check that what you are creating is aligned with what is needed. Think of the

retrospective as if you are a powerful, up-and-coming sports team. Each Monday, whether you won or lost, you watch the game tapes from the weekend's game together as a team to enable you to clearly spot what you need to change in your own performance and what needs to be adjusted in the team's playbook to improve the chances of winning. The same self-awareness and openness to personal and group improvement will also work to hone a winning project team.

The issues that come out of the retrospective are added to the backlog and ranked. That way, they get addressed in the following two weeks. If there are impediments, the project manager puts his or her name on that sticky note and it is placed with the committed stories that will be done this week. However, the project manager tasks do not get counted for the team velocity figures. (You'll understand velocity later.)

If this is an IT team, or a team from any small department or small company, you may not have the luxury of being totally dedicated. Perhaps the team includes all the resources available to the company, and you must do both operational and project activities on a daily basis. In these situations, some teams have had success by placing the operational issues in the backlog as well, and have the product owner, usually a functional manager, rank them, too.

The product owner helps to decide what will be done in the next two weeks, and if he or she is not the functional manager of the people on the team, you can invite the functional manager into the meeting to help prioritize the backlog with the product owner. Let the two of them work out which tasks have priority, and after they have decided what to do, the team is left with a clear direction of the work they should complete, and left out of the middle of any conflicts. Both managing parties of this matrix organization, who have now been involved in the decision making process, also know about and have agreed to any delays in the work of the project or the operational work.

If we pull back for a higher, birds-eye view in Figure 5.7, we can see the whole project. This view of an Agile project is called a **Project Roadmap**. There are four Iterations in each Release. If we know that each Iteration is 2 weeks, in this example, the Release will be 8 weeks in length. There are three Releases, so the entire project should take 24 weeks (6 months) to complete. The other two Releases will eventually have their own 4 Iterations each, but they are not yet planned this early in the timeline, as there may be changes that would cause them to be reworked at an added cost to the project if they are planned in detail too early.

Hopefully, this brief overview gives you confidence that there is nothing in this approach to managing projects that could not be adapted for your own project environment, regardless of your product, service, or industry. Remember, the idea is *not* to slavishly follow every detail, but to use the spirit of it to find ways to ease project pain points and enhance customer satisfaction, speed to market, and profitability for your organization.

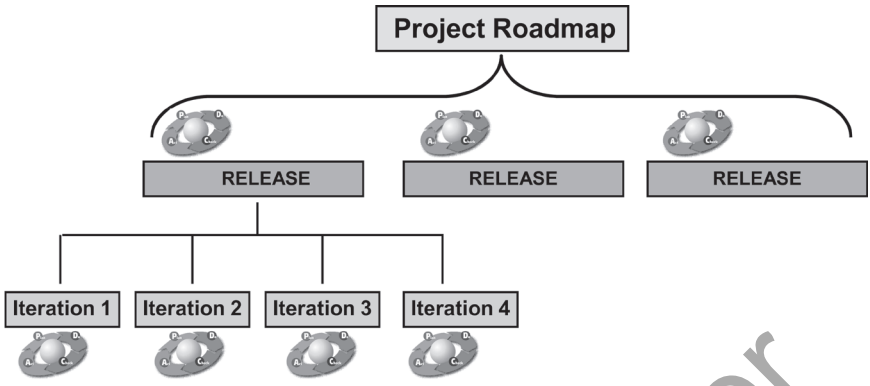

Figure 5.7 Sample Agile Roadmap

References

[1]Davis, B. 2009. *97 Things Every Project Manager Should Know*. New York: O'Reilly Media.

[2]The CHAOS Report. 2002. The Standish Group International, Inc..

[3]Project Management Institute. 2012. *Project Management Body of Knowledge (PMBOK® Guide)*. Fourth Edition. Newtown Square, PA.

[4]Project Management Institute. 2012. *Project Management Body of Knowledge (PMBOK® Guide)*. Fourth Edition. Newtown Square, PA.