Chapter 3

An Agile Approach

> *"A good plan violently executed now is better*
> *than a perfect plan executed next week."*
> –General George S. Patton

Although it started years before, the agile movement officially began with the creation of the Agile Manifesto in February 2001 (Beck 2001). This manifesto was written and signed by seventeen "lightweight methodologists," as they were called at the time. Their document both gave a name to what they were doing and provided a list of value statements. The authors of the Agile Manifesto wrote that they value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Agile teams value individuals and interactions over processes and tools because they know that a well-functioning team of great individuals with mediocre tools will always outperform a dysfunctional team of mediocre individuals with great tools and processes. Great software is made by great individuals and as an industry we have tried too long with too little success to define a development process that relegates individuals to replaceable cogs in the machinery. Agile processes acknowledge the unique strengths (and weaknesses) of individuals and capitalize on these rather than attempting to make everyone homogeneous.

Agile teams value working software over comprehensive documentation because it leads them to have a stable, incrementally enhanced version of the product at the end of each iteration. This makes it possible to collect early, frequent feedback on both the product and the process. As the developed software grows each iteration it can be shown to likely or actual users. Feedback from these users is fed back into the development process to make sure that the team is always working on the highest-valued features and that those features will satisfy user expectations.

Customer collaboration is valued over contract negotiation because agile teams would like all parties to the project to be working toward the same set of goals. Contract negotiation sometimes sets the development team and the project customer at odds right from the start. I enjoy playing most games and when my oldest daughter was four, I bought her a "cooperative game" because it looked like a game she'd enjoy and because I had no idea how a cooperative game could be fun. In the game I bought her, a princess is placed under a spell and players need to remove obstacles (a moat, a locked door, and so on) that are between them and the princess. Players take turns as in most games but the goal is to collaboratively remove obstacles and save the princess. All players win or all players lose. The game is surprisingly fun and we'd like software teams and customers to approach projects with this same attitude of collaboration and shared goals. Yes, contracts are often necessary but the terms and details in a contract can exert great influence on whether the different parties are set on a collaborative or a competitive effort.

Agile teams value responding to change over following a plan because their ultimate focus is on delivering as much value as possible to the project's customer and users. For all but the simplest projects, it is impossible for users to know every detail of every feature they want. It is inevitable that users will come up with new ideas and almost as inevitable that they will decide that some features desired today will become lower priorities tomorrow. To an agile team, a plan is one view of the future but many views are possible. As a team gains knowledge and experience they will factor these into the plan. Perhaps the team is progressing faster or slower than initially expected, perhaps users like one set of features more than expected but don't like another feature that was initially considered critical.

With these four value statements in mind, in this chapter we consider what it means to have an agile approach to a project as well as what it means to have an agile approach to estimating and planning.

An Agile Approach To Projects

With an understanding of the four primary agile value statements, we can turn our attention to what an agile team looks like in practice. Taken collectively, the

four value statements lead to software development processes that are highly iterative and incremental and that deliver coded and tested software at the end of each iteration. The following sections cover some of the main ways in which agile teams work. Including that they:

♦ Work as one team
♦ Work in short iterations
♦ Deliver something each iteration
♦ Focus on business priorities
♦ Inspect and adapt

An Agile Team Works As One

Critical to the success of a project is that all project participants view themselves as one team aimed at a common goal. There is no room for a "throw it over the wall" mentality on an agile project. Analysts do not throw requirements over the wall to designers. Designers and architects do not throw designs over a wall to coders; coders do not throw half-tested code over a wall to testers. A successful agile team must have a we're-all-in-this-together mindset. While an agile team should work together as one whole team, there are a number of specific roles on the team. It was worth identifying and clarifying those roles that play a part in agile estimating and planning.

The first role is the *product owner*. The primary duties of the product owner include making sure that all team members are purusing a common vision for the project, establishing priorities so that the highest-valued functionality is always being worked on, and making decisions that lead to a good return on the investment in the project. When developing commercial software, the product owner is often someone from the marketing or product management side of the company. When developing software for internal use, the product owner may instead be a user, the users' manager, an analyst, or the person funding the project.

A second role is that of *customer*. The customer is the person who has made the decision to fund the project or to buy the software. On a project developing software for internal use, the customer is usually a representative from another group or division. On such projects the product owner and customer roles are often combined. For a commercially distributed product, the customer will be the person who buys the software. In either case, the customer may or may not be a *user* of the software, which is, of course, another important role.

Another role worth highlighting is that of *developer*. I use developer very generally to refer to anyone developing software. That includes programmers, testers, analysts, database engineers, usability experts, technical writers, archi-

tects, designers, and so on. Using this definition, even the product owner may be thought of as a developer on many projects.

A final role is the *project manager.* As described by Highsmith (2004), the role of the project manager changes on agile projects. Agile project managers focus more on leadership than on management. On some agile projects, the person fufilling the project manager role will also act in another role, often as a developer but occasionally as a product owner.

Agile Teams Work in Short Iterations

On an agile project there is no grand delineation of phases—no upfront requirements phase followed by analysis followed by architectural design and so on. Depending upon the actual agile process you select or define, you may put a very short design, modeling, or other phase at the front end of the project. But, once the project has begun in earnest, all work (requirements, design, coding, testing, and so on) happens concurrently within each iteration.

Iterations are timeboxed, meaning they finish on time even if functionality is dropped to do so, and are often very short. Most agile teams work in iterations of from one to four weeks long but some teams maintain their agility with iterations of up to three months. Most teams settle upon a relatively consistent iteration length. Some, however, choose the appropriate length for an iteration at the start of each iteration.

Agile Teams Deliver Something Each Iteration

More crucial than the specific iteration length chosen by a team is that during the iteration they transfrom one or more imprecise requirements statements into coded, tested, and potentially shippable software. Of course many teams will not deliver the results of every iteration to their users; the goal is simply that they could. This means that teams make progress by adding one or more small features in each iteration but that each added feature is coded, tested, and of releaseable quality.

It is essential that the product be brought to this potentially shippable state by the end of each iteration. Practically, this does not mean a team must do absolutely everything necessary to release since they often won't release each iteration. For example, I work with one team that requires two months of mean time between failure (MTBF) testing before releasing their product, which includes both hardware and software. They cannot shorten those two months as it is contractually required by their client and that amount of time is often necessary to check for hardware failures. This team works in four-week iterations and apart

from running this two-month MTBF test, their product is at a truly releasable state at the end of each iteration.

Because a single iteration does not usually provide sufficient time to complete enough new functionality to satisfy user or customer desires, the broader concept of a *release* is introduced. A release comprises one or more (usually more) iterations that build upon each other to complete a set of related functionality. While iterations are most commonly one to four weeks, a release is typically two to six months. For example, in an investment management system, one release may include all of the functionality related to buying and selling mutual funds and money market funds. This may take six two-week iterations to complete (roughly three months). A second release may add stock and bond trading and take four additional two-week iterations. Releases may occur at varying intervals. A first release may take six months to be developed. It may be followed by another release three months later, and so on.

Agile Teams Focus on Business Priorities

Agile teams demonstrate a commitment to business priorities in two ways. First, they deliver features in the order specified by the product owner, who is expected to prioritize and combine features into a release that optimizes the return on the organization's investment in the project. To achieve this, a release plan is created based on the team's capabilities and a prioritized list of desired new features. In order for the product owner to have the most flexibility in prioritizing, features must be written so as to minimize the technical dependencies between them. It is difficult for a product owner to prioritize features into a release plan if the selection of one feature requires the prior development of three others. A team is unlikely to achieve a goal of absolutely no dependencies; however, keeping dependencies at a minimum is often quite feasible.

Second, agile teams focus on completing and delivering user-valued features rather than on completing isolated tasks (that eventually combine into a user-valued feature). One of the best ways to do this is to work with user stories, which are a lightweight technique for expressing software requirements. A user story is a brief description of functionality as viewed by a user or customer of the system. User stories are free-form and there is no mandatory syntax. However, it can be useful to think of a story generally fitting the form: "As a <type of user>, I want <capability> so that <business value>." With this template as an example, you may have the story "As a book-buyer, I want to search for a book by ISBN so that I can find the right book quickly."

User stories are lightweight because the work to gather and document them is not all done upfront. Rather than writing a lengthy requirements specification, agile teams have found it better to pursue to a just-in-time requirements approach. Typically this begins with a short description of a user story being

handwritten on a note card, or perhaps typed into a computer for larger or distributed teams. The story card is just the beginning, though, and each user story is accompanied by as many conversations between the developers and the product owner as needed. These conversations happen as often as needed and include whoever is necessary. Written documentation may continue to exist when using a story-based requirements approach. However, the focus is shifted dramatically from written to verbal communication.

Agile Teams Inspect and Adapt

The plan created at the start of any project is not a guarantee of what will occur. In fact, it is only a point-in-time guess. Many things will conspire to invalidate the plan—project personnel may come or go, technologies will work better or worse than expected, users will change their minds, competitors may force us to respond differently or more rapidly, and so on. Agile teams view every such change as presenting both the opportunity and need to update the plan in order to better reflect the reality of the current situation.

At the start of each new iteration, an agile team incorporates all new knowledge gained in the preceding iteration and adapts accordingly. If a team has learned something that is likely to affect the accuracy or value of the plan, they adjust the plan. The accuracy of the plan may be affected by the team discovering they have over- or underestimated their rate of progress. Or they may discover that a certain type of work is more time-consuming than previously thought.

The value of the plan may be altered by knowledge the product owner has gained about the desires of likely users. Perhaps, based on feedback from seeing the software from an earlier iteration, the product owner has learned that users would like to see more of one type of feature and that they don't value another feature as much as was previously thought. The value of the plan could be increased in this case by moving more of the desired features into the release at the expense of some of the lesser-valued features.

None of this is to say that agile teams take an ad hoc view of changing priorities. Priorities do tend to be relatively stable from one iteration to the next. However, the opportunity to alter priorities between iterations is a powerful contributor to the ability to maximize the return on the project investment.

An Agile Approach to Planning

Estimating and planning the development of a new product is a daunting task made more difficult by our misconceptions about projects. Macomber (2004) points out that we should not view a project solely as the execution of a series of steps. Instead, it is important that we view a project as rapidly and reliably gen-

erating a flow of useful new capabilities and new knowledge. The new capabilities are delivered in the product, the new knowledge is used to make the product the best that it can be.

On an agile project, we use this flow of new capabilities and knowledge to guide the ongoing work. The new knowledge generated by the project may be about the product or the project. New *product knowledge* helps us know more about what the product should be. New *project knowledge* is information about the team, the technologies in use, the risks, and so on.

We frequently fail to acknowledge and plan for this new knowledge. Failing to plan to acquire new knowledge leads to plans built on the assumption that we know everything necessary to create an accurate plan. In the world of software development that is rarely, if ever, the case. Ward Cunningham has said that "it's more planning what you want to learn, not what it [the product] will be in the end." (Van Schooenderwoert 2004).

I often equate the traditional view of a project as running a 10-kilometer race. You know exactly how far away the finish line is and your goal is to reach it as quickly as possible. On an an agile project, we don't know exactly where the finish line is but we often know we need to get to it or as close as we can by a known date. An agile project is more like a timed race than a 10-kilometer race: run as far as possible in sixty minutes. In this way, the agile project team knows when it will finish, but not what they will deliver. When we acknowledge that the end result is both somewhat unknown as well as unknowable in advance, planning becomes a process of setting and revising goals that lead to a longer term objective.

Multiple Levels Of Planning

When setting and revising goals, it is important to remember that we cannot see past the horizon and that the accuracy of a plan decreases rapidly the further we attempt to plan beyond where we can see. For example, suppose you are standing on a small boat and that your eyes are nine feet above the water. The distance to the horizon in this case is slightly over four miles.[1] If you are planning a twenty mile trip, you should plan on looking ahead at least five times, once every four miles. Because you cannot see past the horizon, you need to look up often and adjust your plan.

A project is at risk if its planning extends well beyond the planner's horizon and does not include time for the planner to raise her head, look at the new horizon, and make adjustments. A progressive elaboration of the plan is needed.

---

1. To calculate the distance to the horizon in miles, multiply the square root of the height of your eyes by 1.35.

Agile teams achieve this by planning at three distinct horizons. The three horizons are the release, the iteration, and the current day. The relationships between these (and other) planning horizons can be seen in the planning onion of Figure 3.1.
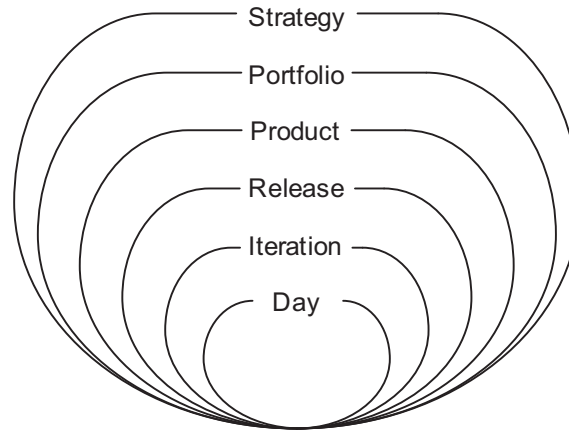

Figure 3.1   The planning onion. Agile teams plan at least at the release, iteration, and day levels.

Most agile teams are only concerned with the three innermost levels of the planning onion. Release planning considers the user stories or themes that will be developed for a new release of a product or system. The goal of release planning is to determine an appropriate answer to the questions of scope, schedule, and resources for a project. Release planning occurs at the start of a project but is not an isolated effort. A good release plan is updated throughout the project (usually at the start of each iteration) so that it always reflects the current expectations about what will be included in the release

At the next level is iteration planning, which is conducted at the start of each iteration. Based on the work accomplished in the just-finished iteration, the product owner identifies high priority work the team should address in the new iteration. Because we are looking at a closer horizon than with release planning, the components of the plan can be smaller. During iteration planning we talk about the tasks that will be needed to transform a feature request into working and tested software.

Finally, there is daily planning. Most agile teams use some form of daily standup meeting to coordinate work and synchronize daily efforts. Although it may seem excessive to consider this planning in the formal sense, teams definitely make, assess, and revise their plans during these meetings. During their daily meetings, teams constrain the planning horizon to be no further away than the next day, when they will meet again. Because of this, they focus on the plan-

ning of tasks and on coordinating the individual activities that lead up to the completion of a task.

By planning across these three time horizons—release, iteration, and day—agile teams focus on what is visible and important to the plan they are creating.

Outside the concern of most individual agile teams (and this book) are product, portfolio, and strategic planning. Product planning involves a product owner looking further ahead than the immediate release and planning for the evolution of the released product or system. Portfolio planning involves the selection of the products that will best implement a vision established through an organization's strategic planning.

Conditions of Satisfaction

Every project is initiated with a set of objectives. Your current project may be to create the world's best word processor. Creating the world's best word processor, however, will typically be only one objective for this project. There will almost certainly be additional objectives regarding schedule, budget, and quality. These objectives can be thought of as the the customer or product owner's *conditions of satisfaction*; that is, the criteria that will be used to gauge the success of the project.

Way back when I was a high school and assigned to write a paper about a book such as *Moby Dick* I would always ask the teacher how long the paper had to be. She'd respond something like "five pages" and I then knew her primary condition of satisfaction. There were, of course, a number of additional, unwritten conditions of satisfaction such as that the paper would be well-written, my own work, in English, and so on.

At the start of release planning, the team and product owner collaboratively explore the product owner's conditions of satisfaction. Common factors in her conditions of satisfaction include the usual items: scope, schedule, budget, and quality, although agile teams typically prefer to treat quality as non-negotiable. The team and product owner look for ways to meet all of the conditions of satisfaction. The product owner may, for example, be equally satisfied with a release in five months that includes one set of user stories as with a release a month later that includes additonal user stories.

Sometimes, however, all of the product owner's conditions of satisfaction cannot be met. The team can build her the world's best word processor but they cannot build it by next month. When no feasible solution can be found, the conditions of satisfaction must change. Because of this, release planning and explo-

ration of the product owner's conditions of satisfaction are highly iterative as can be seen in Figure 3.2.
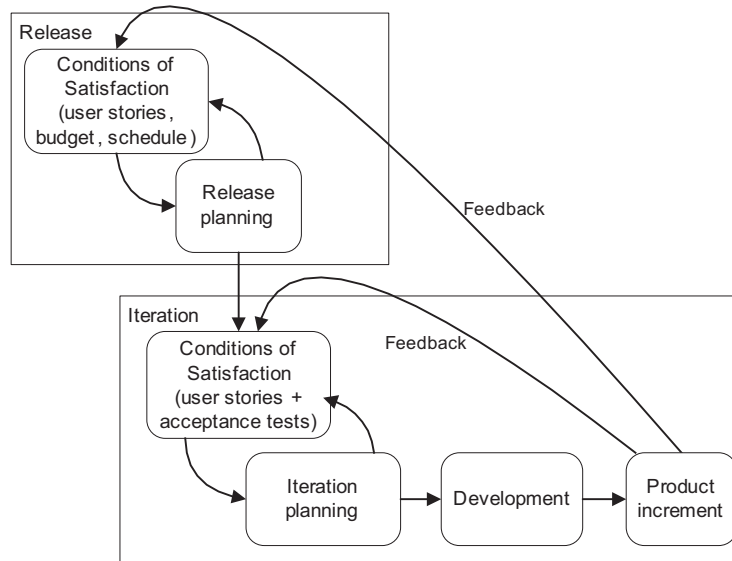

Figure 3.2   Conditions of satisfaction drive both release and iteration planning.

Once a release plan covering approximately the next three to six months is established, it is used as input into the planning of the first iteration. Just as release planning began with consideration of the product owner's conditions of satisfaction, so does iteration planning. For an iteration, the product owner's conditions of satisfaction are typically the features she'd like developed next and some high-level tests about each feature.

As an example, consider a travel reservation site that includes the user story, "As a user, I want to be able to cancel a reservation." In discussing this story with the product owner, the developers learn that her conditions of satisfaction for this story include that:

◆ A user who cancels more than 24 hours in advance gets a complete refund

◆ A user who cancels less than 24 hours in advance is refunded all but a $25 cancellation fee

◆ A cancellation code is displayed on the site and is emailed to the user

Like release planning, iteration planning is iterative. The product owner and the team discuss various ways of best meeting the conditions of satisfaction for the iteration.

Feedback loops are shown in Figure 3.2 from the resulting new product increment back into the conditions of satisfaction boxes at the start of both release and iteration planning. Based on their experience developing the product increment during the iteration the team may have gained knowledge or experience that affects planning at one or more of these levels. Similarly, showing the product increment to existing or likely users may generate new knowledge that may cause changes to the plans. An agile team will incorporate these changes into their plans to the extent they lead to a higher-value product.

Summary

Agile teams work together as a team but include roles filled by specific individuals. First is the product owner, who is responsible for the product vision and for prioritizing features the team will work on. Next is the customer, who is the person paying for the project or purchasing the software once it's available. Users, developers and managers are other roles on an agile project.

Agile teams work in short, timeboxed iterations that deliver a working product by the end of each iteration. The features developed in these iterations are selected based on the priority to the business. This ensures that the most important features are developed first. User stories are a common way for agile teams to express user needs. Agile teams understand that a plan can rapidly become out of date. Because of this they adapt their plans as appropriate.

Projects should be viewed as rapidly and reliably generating a flow of useful new capabilities and new knowledge rather than as just the execution of a series of steps. Projects generate two types of new knowledge: knowledge about the product and knowledge about the project. Each is useful in refining a product plan toward achieving the most value for the organization.

Agile teams use three levels of planning: release planning, iteration planning, and daily planning. The release plan looks ahead for the duration of the release, typically three to six months. An iteration plan looks ahead only the duration of one iteration, typically one to four weeks. A daily plan is the result of team member commitments made to each other usually during a daily standup meeting.

Understanding the product owner's conditions of satistaction is critical in both release and iteration planning. During release planning, the whole team identifies a way of meeting the conditions of satisfaction for the release, which includes scope, schedule, and resources. To achieve this, the product owner may need to relax one or more of her conditions of satisfaction. A similar process occurs during iteration planning when the conditions of satisfaction are the new features that will be implemented and the high level test cases that demonstrate the features were implemented correctly.