Chapter 1

Modernizing Project Management

In This Chapter

- Understanding why project management needs to change
- Finding out about agile project management

Agile project management is a style of project management that focuses on early delivery of business value, continuous improvement of the project's product and processes, scope flexibility, team input, and delivering well-tested products that reflect customer needs.

In this chapter, you find out why agile processes emerged as an approach to software development project management in the mid-1990s and why agile methodologies have caught the attention of project managers, customers who invest in the development of new software, and executives whose companies fund software development departments. This chapter also explains the advantages of agile methodologies over long-standing approaches to project management.

COPYRIGHTED MATERIAL
http://www.pbookshop.com

Project Management Needed Makeover

A *project* is a planned program of work that requires a definitive amount of time, effort, and planning to complete. Projects have goals and objectives and often must be completed in some fixed period of time and within a certain budget.

If you are reading this book, it's likely that you are either a project manager, or you are someone who initiates projects, works on projects, or is affected by projects in some way.

Agile approaches are a response to the need to modernize project management. To understand how agile approaches are revolutionizing projects, it helps to know a little about the history and purpose of project management and the issues that projects face today.

The origins of modern project management

Projects have been around since ancient times. From the Great Wall of China to the Mayan pyramids at Tikal, from the invention of the printing press to the invention of the Internet, people have accomplished endeavors big and small in projects.

As a formal discipline, project management as we know it has only been around since the middle of the twentieth century. Around the time of World War II, researchers around the world were making major advances in building and programming computers, mostly for the United States military. To complete those projects, they started creating formal project management processes. The first processes were based on step-by-step manufacturing models the United States military used during World War II.

People in the computing field adopted these step-based manufacturing processes because early computer-related projects relied heavily on hardware, with computers that filled up entire rooms. Software, by contrast, was a smaller part of computer projects. In the 1940s and '50s, computers might have thousands of physical vacuum tubes but fewer than 30 lines of programming code. The 1940s' manufacturing process used on these initial computers is the foundation of the project management methodology known as waterfall.

In 1970, a computer scientist named Winston Royce wrote "Managing the Development of Large Software Systems," an article for the IEEE that described the phases in the waterfall methodology. The term *waterfall* was coined later, but the phases, even if they are sometimes titled differently, are essentially the same as originally defined by Royce:

1. Requirements.
2. Design.
3. Development.
4. Integration.
5. Testing.
6. Deployment.

On waterfall projects, you move to the next phase only when the prior one is complete — hence, the name waterfall.

Pure waterfall project management — completing each step in full before moving to the next step — is actually a misinterpretation of Royce's suggestions. Royce identified that this approach was inherently risky and recommended prototyping and working within iterations to create products — suggestions that were overlooked by many organizations that adopted the waterfall methodology.

http://www.pbookshop.com

Software project success and failure

Unfortunately, the stagnation in project management approaches is catching up with the software industry. In 2009, a software statistical company called the Standish Group did a study on software project success and failure in the United States. The results of the study showed that

- *24% of projects failed outright.* This means that the projects were cancelled before they finished, and did not result in any product releases. These projects delivered no value whatsoever.
- *44% of projects were challenged.* This means that the projects finished, but they had gaps between expected and actual cost, time, quality, or a combination of these elements. The average difference between the expected and actual project results — looking at time, cost, and features not delivered — was 189%.
- *32% of projects succeeded.* This means the projects finished and delivered the expected product in the originally expected time and budget.

In 2009, companies and organizations in the U.S. spent $191.2 billion on application development. That means that more than $103 billion was wasted on failed projects.

http://www.pbookshop.com

Until improved approaches based on agile techniques surpassed it around 2008, the waterfall methodology was the most common project management approach in software development.

The problem with the status quo

Computer technology has, of course, changed a great deal since the last century. I have a computer in my pocket with more power, memory, and capabilities than the largest, most expensive machine that existed when people first started using waterfall methodologies — and my computer has a telephone attached.

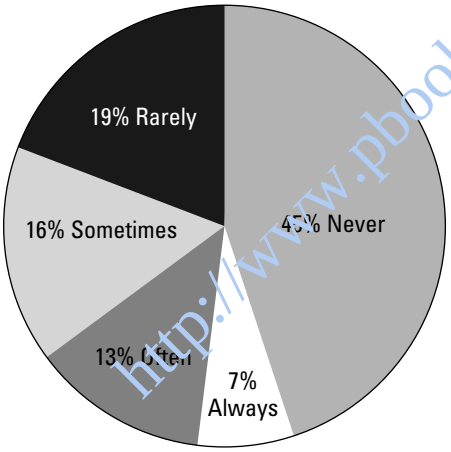
At the same time, the people using computers have changed as well. Instead of creating behemoth machines with minimal programs for a few researchers and the military, people create hardware and software for the general public. In many countries, almost everyone uses a computer, directly or indirectly, every day. Software runs our cars; it provides our daily information and daily entertainment. Even tiny children use computers — my friends' two-year-old is almost more adept with the iPhone than her parents. The demand for newer, better software products is constant.

Somehow, during all this growth of technology, the processes were left behind. Software developers are still using project management methodologies from the 1950s, and these approaches were all derived from manufacturing processes meant for the hardware-heavy computers of the mid-twentieth century.

Today traditional projects that do succeed often suffer from one problem: *scope bloat*, the introduction of unnecessary product features in a project.

Think about the software products you use every day. For example, the word-processing program I'm typing on right now has a lot of features and tools. Even though I write on this program every day, I use only some of the features all the time. There are some elements that I use less frequently. There are quite a few tools that I have never used, and come to think of it, I don't know anyone else who has used them, either. These features that few people or no one uses are the result of scope bloat.

Scope bloat appears in all kinds of software, from complex enterprise applications, to websites that everyone uses. The chart in Figure 1-1 shows data from another Standish Group study that illustrates just how common scope bloat is. In the figure, you can see the proportion of requested features that are actually used when the software goes into production. Sixty-four percent of the features are rarely or never used.



Copyright© 2011 by Standish Group.

Figure 1-1: Actual use of software features.

http://www.pbookshop.com

The numbers in Figure 1-1 illustrate an enormous waste of time and money. That waste is a direct result of traditional project management processes that are unable to accommodate change. Project managers and stakeholders know that change is not welcome mid-project, and that their best chance of getting a potentially desirable feature is at the project start, so they ask for

- Everything they need
- Everything they think they may need
- Everything they want
- Everything they think they may want

The result is the bloat in features that results in the statistics in Figure 1-1.

The problems associated with using outdated management and development approaches are not trivial. These problems waste billions of dollars a year. The $103 billion lost in project failure in 2009 (see the sidebar, "Software project success and failure") could equate to millions of jobs around the world.

Over the past two decades, people working on projects have recognized the growing problems with traditional project management and have been working to create a better model: agile project management.

Introducing Agile Project Management

The seeds for agile techniques have been around for a long time. Figure 1-2 shows a quick history of agile project management, dating back to the 1930s with Walter Sherwart's Plan-Do-Study-Act (PDSA) approach to project quality.

In 1986, Hirotaka Takeuchi and Ikujiro Nonaka published an article called "New New Product Development Game in the Harvard Business Review." Takeuchi and Nonaka's article described a rapid, flexible development strategy to meet fast-paced product demands. This article first coined the term *scrum* in conjunction with product development, comparing product development to a game of rugby. Scrum eventually became one of the most popular agile project management approaches.

In 2001, a group of software and project experts got together to talk about what their successful projects had in common. This group created the *Agile Manifesto*, a statement of values for successful software development:

> **Manifesto for Agile Software Development***
>
> We are uncovering better ways of developing
> software by doing it and helping others do it.
> Through this work we have come to 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
>
> That is, while there is value in the items on
> the right, we value the items on the left more.
>
> ** Agile Manifesto Copyright© 2001: Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas*
>
> *This declaration may be freely copied in any form, but only in its entirety through this notice.*

http://www.pbookshop.com

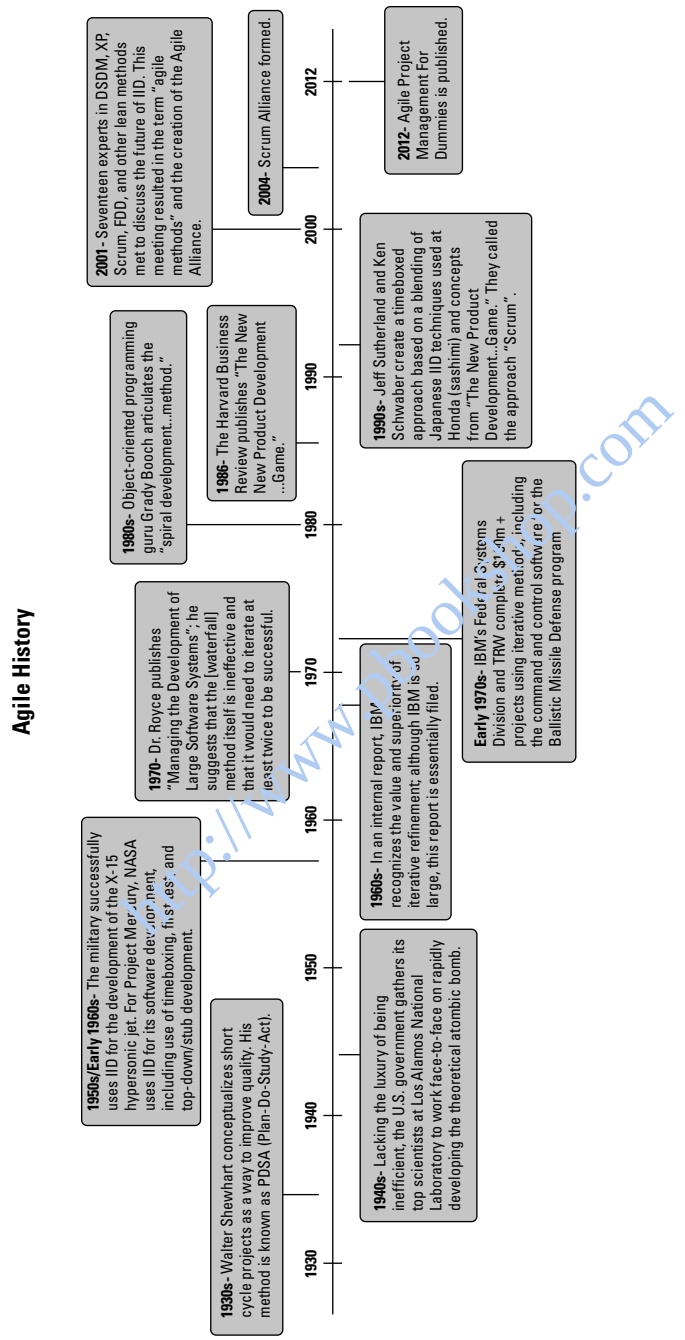


Figure 1-2: Agile project management timeline.

http://www.pbookshop.com

These experts also created the *Agile Principles*, 12 practices that help support the values in the Agile Manifesto. I list the Agile Principles and describe the Agile Manifesto in more detail in Chapter 2.

Agile, in product development terms, is a description for project management methodologies that focus on people, communications, the product, and flexibility. There are many agile methodologies, including scrum, extreme programming, and lean, but they all have one thing in common: adherence to the Agile Manifesto and the Agile Principles.

How agile projects work

Agile approaches are based on an *empirical control method* — a process of making decisions based on the realities observed in the actual project. In the context of software development methodologies, an empirical approach can be very effective in both new product development and enhancement and upgrade projects. By using frequent and firsthand inspection of the work to date, you can make immediate adjustments, if necessary. Empirical control requires

- **Transparency:** Everyone involved on an agile project knows what is going on and how the project is progressing.
- **Frequent inspection:** The people who are invested in the product and process the most should regularly evaluate the product and process.
- **Adaptation:** Make adjustments quickly to minimize problems; if inspection shows that you should change, then change immediately.

http://www.pbookshop.com

To accommodate frequent inspection and adaptation, agile projects work in *iterations* (smaller segments of the overall project). On agile projects, you still have the same type of work that is involved on a traditional waterfall project: You have to create requirements and designs, you develop your product, and if necessary, you integrate your product with other products. You test the product, fix any problems, and deploy it for use. However, instead of completing these steps for all of your product features at once, you break the project into iterations, also called *sprints*.

Figure 1-3 shows the difference between a linear waterfall project and an agile project.

Mixing traditional project management methods with agile approaches is like saying, "I have a Porsche 911 Turbo. However, I'm using a wagon wheel on the front left side and right rear side. How can I make my car as fast as the other Porsches?" The answer, of course, is you can't. If you fully commit to an agile approach, you will find you have a better chance at project success.

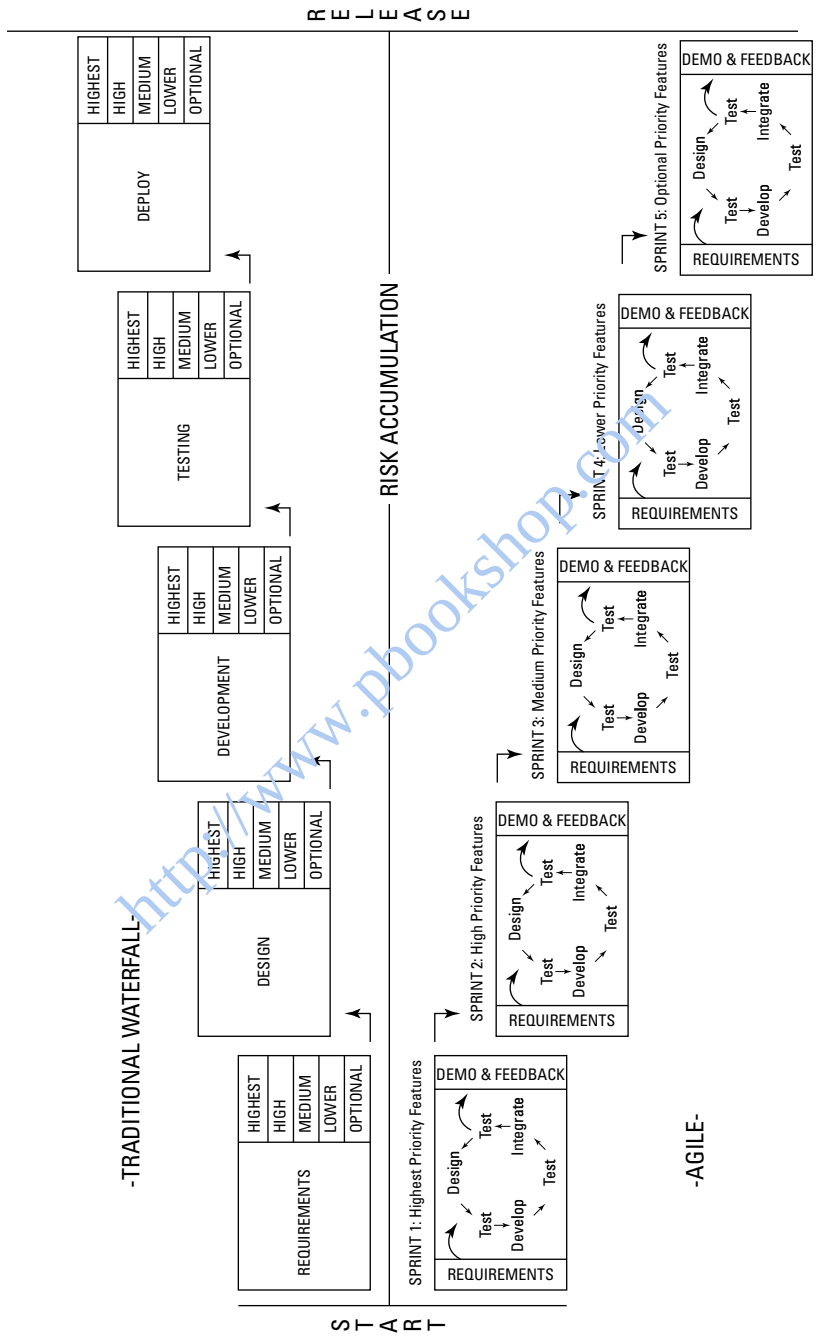


Figure 1-3: Waterfall versus agile project.

http://www.pbookshop.com

Why agile projects work better

Throughout this book, you will see how agile projects work better than traditional projects. Agile project management methodologies have been able to produce more successful projects. The Standish Group, mentioned in the sidebar "Software project success and failure," also did a study of project success rates in 2009. That year, the group found that 26 percent of projects failed outright — but in 2011, that number fell by 5 percent. The decrease in failure has, in part, been attributed to wider adoption of agile approaches.

Here are some key areas where agile approaches are superior to traditional project management methods:

- **Project success rates:** In Chapter 15, you will find out how the risk of catastrophic project failure falls to almost nothing on agile projects. Agile approaches of prioritizing by business value and risk ensures early success or failure. Agile approaches to testing throughout the project help ensure that you find problems early, instead of after spending a large amount of time and money.
- **Scope creep:** In Chapters 7, 8, and 12, you see how agile approaches accommodate changes throughout a project, minimizing the idea of scope creep. On agile projects, you can add new requirements at the beginning of each sprint without disrupting development flow. By fully developing prioritized features first, you'll prevent scope creep from threatening critical functionality.
- **Inspecting and adaptation:** In Chapters 10 and 14, you find details of how regular inspecting and adaptation work on agile projects. Agile project teams can improve their processes and their products with each sprint armed with information from complete development cycles and actual product.

http://www.pbookshop.com

Throughout many of the chapters in this book, you discover how you gain control of the outcome of agile projects. Testing early and often, adjusting priorities as needed, using better communication techniques, and regularly demonstrating and releasing product functionality allow you to fine-tune your control over a wide variety of factors on agile projects.

If you're interested in the possibility of agile, then this is the perfect book for you.

http://www.pbookshop.com