In this chapter you will be introduced to the principles and practices that constitute agile development. You will learn that agile development is as much a philosophical and cultural shift as it is a set of practices and processes. You will understand why the need for an agile approach to software development has developed, the issues it helps to solve, and the reasons for its rapid rise in popularity.

In this chapter you will also dive into the Agile Manifesto, the document that started the agile movement. You will then examine the key features of agile by digging deeper into the principles and values as laid out by the manifesto and understand what they mean at a more granular level.

Finally you will be introduced to a number of practices that all fall under the agile umbrella. These practices share a common goal of striving to make your development effort more flexible, adaptable, and ultimately of more value to the business.

The aim of this chapter is to provide you with the knowledge that will form the foundations on your road to becoming a master agile practitioner over the course of this book.

**Why the Need for Agile?**

So where did the need for an agile software development methodology come from and what was so bad about agile’s predecessors?

**It’s What I Asked for But Not What I Need**

Previously, when a team would develop software they would use plan-driven development. This type of development was characterized by *gated stages*, where one would gather all the requirements the customer would need on the project, and then do an analysis of the problem. Next, the whole application was designed before the first line of code was ever written.

One of the most widely adopted methodologies associated with plan-driven development was the *waterfall* approach to software development. The waterfall approach uses gated stages of requirements gathering, planning, designing, development, testing, and then, eventually, deploying, as seen in Figure 1-1.
The plan-driven method, while great for industries like construction—where requirements remain fixed throughout the project, has its drawbacks when applied to an industry where requirements can change during the lifecycle of the project, as is often the case with software development. Real-world software projects change, not every requirement can be gathered up front, things get missed, and the business is always learning and figuring out better ways to do things. We want the software to outlive the business requirements; not the business requirements outliving the software.

Plan-driven development relies on unchanging requirements. That is to say, once they have been gathered and agreed they may not be changed. If they have to be changed, it is at a great cost to the development team as well as the customer. The notion that a business would remain static for nine to thirty-six months, which is what an average project lasts, is almost absurd. Businesses and project stakeholders are constantly looking to improve process and innovate, and cannot jeopardize this evolution because they are waiting on a software tool to be completed. During the lifecycle of a plan-driven project, the business would find it difficult to give feedback on requirements and design documentation. Because requirements are a gated stage in the process, many plan-driven projects would proceed without the stakeholders really understanding what was to be delivered. Many times stakeholders are uncertain about what they want. A 400-page requirements document is not the ideal way to communicate what the new system will do. However, this was necessary to satisfy a gated stage of the plan-driven method, and development would not start until the project was through that gate.

With this gated process there is not a convenient mechanism for the development team to show their work and for the stakeholders to offer feedback on that work. Therefore, oftentimes the first opportunity that stakeholders would have to offer feedback on the project was during the QA (quality assurance) stage of the process, which would happen after the coding gate was completed. What this means is that a stakeholder would ask for a solution to a problem and would not see a response from the team for a year or more. This is a black-box type of development environment. The customer sends issues in and doesn’t see a possible resolution for a year or more.

Figure 1-1. The waterfall process