THE EXPERT'S VOICE® IN .NET

Pro Agile .NET Development with Scrum

DISCOVER HOW AGILE PRACTICES WITH SCRUM WORK IN A REAL-WORLD ASP.NET MVC PROJECT

Jerrel Blankenship, Matthew Bussa, and Scott Millett

Apress®

Pro Agile .NET Development with Scrum

Jerrel Blankenship
Matthew Bussa
Scott Millett

Apress®

Pro Agile .NET Development with Scrum

ISBN 978-1-4302-3533-0

ISBN 978-1-4302-3534-7 (eBook)

President and Publisher: Paul Manning
Lead Editor: Jonathan Hassell
Technical Reviewer: Russ Lewis and Damien Foggon
Editorial Board: Clay Andres, Steve Anglin, Mark Beckner, Ewan Buckingham, Gary Cornell, Jonathan Gennick, Jonathan Hassell, Michelle Lowman, Matthew Moodie, Duncan Parkes, Jeffrey Pepper, Frank Pohlmann, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Coordinating Editor: Jessica Belanger
Copy Editor: Kim Burton-Weisman
Compositor: Bytheway Publishing Services
Indexer: SPI Global
Artist: SPI Global
Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer Science+Business Media, LLC., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

The source code for this book is available to readers at www.apress.com. You will need to answer questions pertaining to this book in order to successfully download the code.

To my wife, Stacy, my son, Tyler, and my family
–Jerrel

To my wife, Rebakah, and my family
–Matthew

Contents at a Glance

Contents

About the Authors

Jerrel Blankenship, software craftsman, specializes in Microsoft technologies. He has been an application developer on the .NET platform for more than six years. He is a Certified ScrumMaster and a Microsoft Certified Professional.

During his career, Jerrel has developed a number of .NET-based software projects that run on desktop, web, and mobile environments. He enjoys working with impassioned developers and sharing the knowledge of agile and Scrum to teams that want to build software more effectively.

Jerrel is passionate about many things, including his family, fishing, chess, Cleveland-based sports teams, and gaming. He currently resides in Columbus, Ohio, with his wonderful wife, Stacy, and son Tyler. You can read Jerrel's ramblings at `www.jerrelblankenship.com` and he can be reached via e-mail at `jerrel@jerrelBlankenship.com`.

Matthew Bussa is software craftsman specializing in Microsoft technologies. A Certified ScrumMaster and a Microsoft Certified Technology Specialist, Matthew has developed .NET-based solutions for both the web and the desktop. He enjoys helping transform agile teams through working together more effectively—so that they can build awesome software!

Matthew currently resides in Columbus, Ohio, with his wife, Rebakah. He maintains a blog at `www.matthewbussa.com` and can be reached via e-mail at `matthew@matthewbussa.com`.

About the Technical Reviewers

■ **Russ Lewis** has been helping organizations build better software since the late 80s. An engineer by training, and entrepreneurial by background, he places the highest value in identifying and satisfying customer requirements. Agile principles and solid architecture have long driven his projects, even before the terms "Agile" and "Architecture" became fashionable.

He was one of the first to recognize the business benefits of service-orientation, developing training courses for Microsoft and Learning Tree International on Web Services and COM+.

Lately, he has been consulting with some of the world's leading organizations including Randstad, Toyota, Transport for London, and the Government of Angola, as software architect and agile lead.

■ **Damien Foggon** is a developer, writer, and technical reviewer in cutting-edge technologies and has contributed to more than 50 books on .NET, C#, Visual Basic and ASP.NET. He is the co-founder of the Newcastle based user-group NEBytes (online at `http://www.nebytes.net`), is a multiple MCPD in .NET 2.0 and .NET 3.5 and can be found online at `http://blog.fasm.co.uk`.

Acknowledgments

The thought of trying to put to paper all who have helped me get to where I am is nearly impossible. I have been fortunate in my life to have met wonderful people who have each helped me both personally and professionally. I may forget some of you and for that I am sorry. Please remember that without all of you I would not have gotten where I am.

First and foremost, I want to thank my wife, Stacy, for her love, encouragement, and understanding through everything I have been through. She is my rock, my companion, and my friend. I want to thank my son, Tyler, for always bringing a smile to my face whenever he enters a room. I want to thank my parents and family for their love, encouragement, and support.

I want to thank Tom Maier for giving me my first shot at showing the world what I can do and guiding me through the early stages of my career. I want thank Jared Conway for giving me insight and showing me what a great developer and friend can do.

I want to give a tremendous thanks to Jared Richardson, Tim Wingfield, and Michael Kramer for showing me the wonderful world that is agile. They were all mentors to me and helped shaped my career. Without these guys, my involvement with this book would have been non-existent. Thank you guys!

I thank my co-authors, Matthew Bussa and Scott Millet, who helped make this book awesome. I wish to thank everyone at Apress who gave me a way to share my voice and love of development with the world, especially Jessica Belanger, Jonathan Hassell, and Tom Welsh.

—Jerrel Blankenship

I'm not going to lie to you. The Acknowledgments section of the book is a scary proposition for me. The notion that I (or anyone) could quantify, or prioritize, the contributions (direct and indirect) by which this book was made possible is fantasy. I'll do my best, but it's entirely possible I'll forget somebody, and for that, I apologize in advance.

I foremost thank my wife, Rebakah, for her love, eternal patience, indulgence, support, and encouragingly witty comments. She is my love, my companion, and my best friend. I want to thank Michael Hoffer for his friendship and for Fridays. I thank my parents, sister, and my in-laws for their influences, grace, support, and love.

I want to thank Jared Richardson, Tim Wingfield, and Michael Kramer for first introducing me to the practices of Scrum and the Agile Manifesto, and teaching me the importance of building a great team and that software can remain soft with good engineering practices. Their mentorship made a difference and also made this book possible.

I thank my co-authors, Jerrel Blankenship and Scott Millett, who helped make this the most exciting book I've ever worked on! I also want to say thanks to the people at Apress who helped ensure the highest standards in quality every step of the way.

—Matthew Bussa

Introduction

The Agile Manifesto set forth a set of principles on how we as developers create software for our customers. Over the past 10 or so years, we have seen those ideas and principles expanded upon by developers all over the world.

Transitioning into an agile team takes hard work and may be a bit overwhelming. What we hope to show in this book is what this transition might look like for a .NET development team.

Who This Book Is For

This book is for software developers who want to learn how to work in an agile environment and develop software using a test-first/behavior-first approach. This book is for developers who want to start with the business, not a column in a table.

This book assumes that you have some familiarity with the .NET framework. When it comes to the testing and mocking frameworks, this book assumes you have little familiarity.

How This Book Is Structured

This book contains ten chapters and seven appendices.

Chapter 1: "The Art of Agile Development" gives a general overview of agile. This overview includes the difference between plan-driven and value-driven development.

Chapter 2: "Managing Agile Projects with Scrum" provides an introduction to Scrum.

Chapter 3: "eXtreme Programming" discusses the fundamentals of eXtreme Programming (XP) and its relationship with Scrum and behavior-driven development.

Starting in Chapter 4, the book provides a fictional case study about a team utilizing the concepts and ideas from the previous chapters to develop a web-based blackjack game.

Chapter 4: "Sprint 0: Generating the Product Backlog," covers establishing a baseline sprint to develop three different user stories: Initial Bet, Start Game, and Deck of Cards. We'll establish the logistical fundamentals of a sprint and set the tone for the next four chapters.

Chapter 5: "Sprint 1: Starting a Game" introduces the team experiencing their first sprint in the project. It shows how the daily stand-up, retrospective, planning, and product demo meetings work in the real world.

Chapter 6: "Sprint 2: Playing a Basic Game" shows the team dealing with their second sprint and the user stories they have completed.

Chapter 7: “Sprint 3: Changing the Game” finds the team dealing with a change in their group dynamics.

Chapter 8: “Sprint 4: The Release” presents the culmination of four sprints’ worth of work for the first release of the blackjack game to the customer.

Chapter 9: “Code Review” gives a brief overview of the behind-the-scenes framework used on the blackjack web application.

Chapter 10: “What’s Ahead for You and Scrum,” is our retrospective of the product release; it takes a look at what we've covered and gives some pointers on where to go from here.

Appendix A: “TDD Primer with NUnit” is a tutorial on installing and using NUnit to begin to build an automated test suite.

Appendix B: “BDD Primer with SpecFlow” gets you started with the basics of SpecFlow and shows how to transform specifications into workable code.

Appendix C: “Mocking with Moq” is a tutorial explaining why mocking is important and showing you how to mock using the Moq framework.

Appendix D: “Manage a Product Backlog with Pivotal Tracker” is an introduction to a free, online agile management tool to track user stories throughout their lifecycle.

Appendix E: “Web Testing with WaitiN” discusses how to use WaitiN, an automated GUI framework for the browser.

Appendix F: “Source Control with SVN” discusses how to set up and use a version control system for your source code.

Appendix G: “Continuous Integration with CruiseControl .NET” explains how to install and configure a continuous integration server using CruiseControl .NET.

Conventions

You will notice a tremendous amount of dialog among the team members in the case study chapters. These conversations are italicized.

In instances where a code line is too long to fit the page’s width, we break it with a code continuation character. Please note that when you try to type the code, you have to concatenate the line without any spaces.

Prerequisites

A knowledge of C# and ASP.NET MVC is tremendously useful. No other previous knowledge is required.

To make the most of this book, install Visual Studio 2010 Express with Service Pack 1 or higher and SQL Server 2008 Express R2 or higher. Both are available for free download from `www.microsoft.com/visualstudio/en-us/products/2010-editions/express`.

Downloading the Code

The source code for this book is available from the Apress web site (`www.apress.com`) in the Source Code / Download section.

Contacting the Authors

We always welcome your questions and feedback regarding the contents of this book. You can reach Jerrel Blankenship by e-mail at `jerrel@jerrelblankenship.com` or via his web site at `www.jerrelblankenship.com`. You can contact Matthew Bussa via e-mail at `matthew@matthewbussa.com` or through his web site at `www.matthewbussa.com`.

CHAPTER 1

The Art of Agile Development

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.

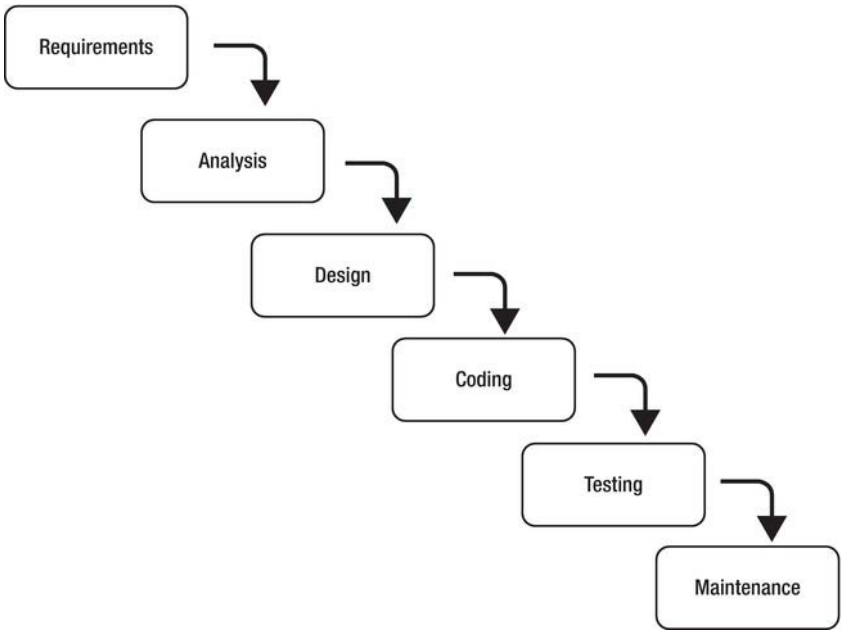


Figure 1-1. *The waterfall process*

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.

In this situation, the stakeholder and business would have to accept that they met the requirements as they were defined at the beginning of the project, even if the needs of the business and the environment that the business works in had changed since the requirements' gated stage. A plan-driven approach can only expect to deliver up to the requirements that were agreed upon at the beginning. What the business knew then has been eclipsed by what they know now, perhaps making the software redundant, or worse, obsolete.

One of the biggest issues with the plan-driven process is the lack of any real return on investment to the business until the end of project, during the deployment stage. There is no tangible benefit or value to the business during the months of design and detailed requirement documents. The business cannot just take that 400-page requirements document and use it in their day-to-day operations. It is only when the project is finally finished that the business can expect to see any inkling of business value.

The plan-driven method makes no provision for the unknown. You could say that the plan-driven method of software development's goal is to eliminate the unknown from a project precisely because it has no mechanism for dealing with it. Hence the need for gated stages: you cannot move to the next stage until all the unknowns are known. Because of this need to remove the unknown from a project, no provision is made for altering the initial design when technical issues surface that require these changes.

A by-product of this need to remove the unknown from a project is the way estimation is handled. By removing the unknown and agreeing on the time estimates of the project, all delays that occur in the project are stacked up to the end of the project. In plan-driven development, there is no correction mechanism for estimation errors and the only buffer on this is the amount of over-estimation (slack) that was originally added on to the project.

It is also true that the process does not take into consideration the technical expertise of the developers who will carry out the implementation. These developers carry the responsibility for the eventual implementation of the project. The smallest coding error can have major consequences that may go unnoticed for years, so it is appropriate to think of developers as engineers who make a myriad of decisions, implement technical designs, and solve problems many times during their working day.

The plan-driven method has some shortcomings that do not adequately support the needs of certain organizations. Experiencing projects that overrun or under-deliver also highlights the weaknesses of this method.

Plan-driven development only works in a situation where product managers and business stakeholders know exactly what they want, will not change their minds, are clear on priorities, and are sure that the business process does not change. We have not been able to find any examples of this mythical project, but if you happen to find yourself working on one, then please give us a call and we will be more than happy to join you!

Putting too much emphasis and time into upfront design and requirements gathering can be a risk to the business in terms of both opportunity and cost. The need for a more reliable and iterative approach, where risk is minimized, and that can give businesses maximum return on their investment, is where agile comes in.

Iterative Change

Software development is simply a means to an end. It enables organizations to automate, streamline, and improve their business processes to solve business problems in order to ultimately reach their goals. By adopting an agile development methodology, and its idea of value-driven development, you will be able to understand and meet the challenges of today's businesses, and in turn you will be able to offer much more value to your stakeholders.

Frequent feedback and interaction between the development team and the stakeholders, domain experts and sponsors, means that agile projects deliver value very rapidly. Task prioritization ensures urgent needs are satisfied first. Iterative development cycles minimize risk, and regular delivery of

working software leads to smooth roll-outs, user satisfaction, and reduced training and maintenance costs.

As the software development discipline has matured, agile methods were developed as an evolution from earlier methodologies.

The agile methodology is as much a philosophical shift as it is a process shift. Agile has a firm emphasis on customer satisfaction and a quick return on investment via an iterative approach to software development. Figure 1-2 shows the process of an agile workflow.

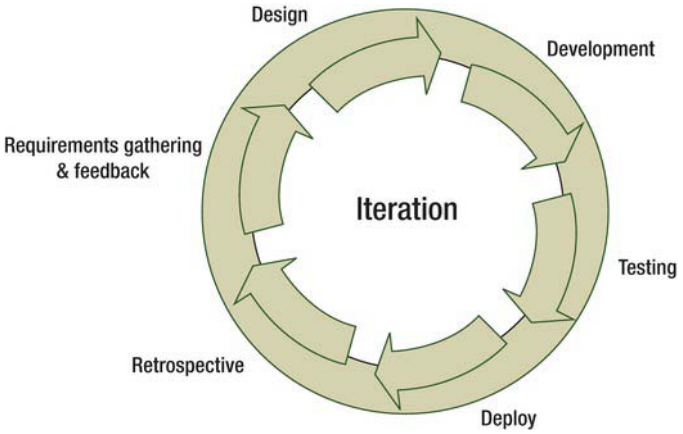


Figure 1-2. *The agile process*

Instead of upfront design and planning stages that strive to remove the unknown from a project before development, agile focuses on small, feature-driven iterations that strive to solve specific business problems. These iterations usually occupy a time box of a fixed two to four weeks' duration. These iterations include all steps of the plan-driven process and enable the business to give frequent feedback on working software in a very short time. The difference between an iteration and doing a project using the plan-driven method of software development is that each iteration is working on small chunks of the project. These chunks are what the stakeholders have designated as the highest priority requirements in the system.

The ability to give working software back to the business within a short time enables the business to start working with that software and gaining value from it—even if that value is to learn that this is not what they really wanted after all. Because agile is so closely aligned with the business, domain experts are considered first-class team members and often meet with the development teams.

Unlike the relaxed start and frantic finish of the more traditional waterfall-based approach, agile promotes a more sustainable working pace.

By breaking down the deliverables of the project into smaller pieces that can be completed in an iteration, agile is providing a mechanism for improving the accuracy of the team's estimates. This mechanism is missing in a plan-driven project. Typically, by the third or fourth iteration the team will be producing fairly accurate estimates. With more accurate estimates, the project manager or sponsor can get a good prediction of the time required to complete the whole system.

Agile is very much like a business, where it is always focusing on improvement of the process by learning and refining its processes. Constant feedback loops from business and development stakeholders help to hone these skills and processes, enabling more efficient delivery of valuable software.

In the end, by applying the agile methodology and using value-driven software development, you as a developer are delivering software that meets the needs of the business in an iterative timeframe.

Now that you have a handle on some of the problems agile has been designed to tackle, let's take a closer look at how to be an agile developer—starting by looking at the Agile Manifesto and some key features that it contains.

Defining Agile

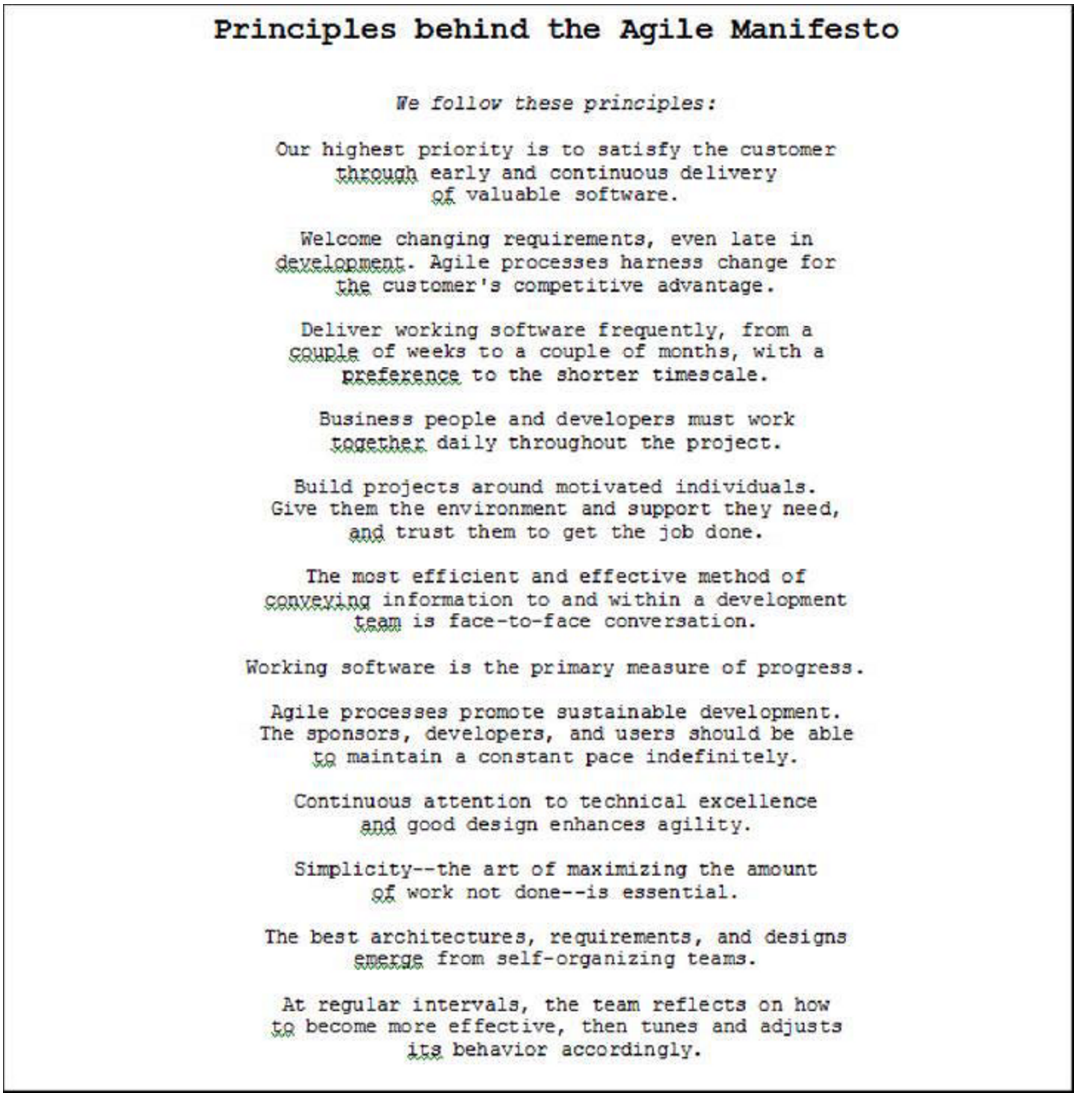
This section will provide you with a clear definition of the agile development process and some of the key features it encompasses.

The Agile Manifesto

In the 1990s there were several people in our profession who were talking about changing the way we wrote software. These discussions came to a head in 2001, when a number of software development luminaries, including the likes of Martin Fowler, Kent Beck, Bob Martin, Ken Schwaber, Jess Sutherland, and Dave Thomas met in a lodge at the Snowbird ski resort in the Wasatch Mountains of Utah. What came of this meeting became known as the Agile Manifesto (see Figure 1-3).

Figure 1-3. *The Agile Manifesto*

In addition to the manifesto, twelve principles of agile, shown in Figure 1-4, were created to expand on the manifesto's declaration.



Principles behind the Agile Manifesto

We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Figure 1-4. *The Twelve Principles behind the Agile Manifesto*

Let's expand on the manifesto and its principles to define a set of key features that an agile process should have.

Key Features of Agile

Looking through the Agile Manifesto and, in particular, the twelve principles, we can identify some key features that define the process and mindset. Let's explore these at a deeper level.

- *Embracing change by understanding the needs of the business*: Being agile is a realization that change is inevitable, nobody gets it right the first time, business priorities change, and people get things wrong. Agility comes about by embracing change, and learning from and with the business. With this in mind agile defines the ability to adapt and be flexible, to embrace change rather than resist it or sit around and moan that the goalposts have moved. Agile teams embrace change and actively identify changes in applications that will increase business value.

- *Focusing on the business value and return on investment (ROI)*: Agile development is a development mind shift and a refocusing of efforts and priorities. There are a number of techniques that you will be introduced to in this book that will help you become a more agile developer. However, becoming truly agile is so much more than the sum of its parts. The tools, project methodologies, and programming methods can certainly go some way to help one become agile, but it is the ability to apply these techniques to an ever-changing business that will truly reap the rewards. Fundamentally you must understand the business domain you are working within and align your efforts, practices, and process to realize its value.

- *Continuous delivery via incremental and iterative development*: Being agile is all about delivering working software of value as often as possible. Success of software development is not measured in the amount of design work. Businesses measure success in working software; this should be your measure of progress as well.

- *Continuous improvement by learning from and with the business*: As part of the software development team, it's our job to turn the language and processes of the business into software systems. In order to do this it is vital that we work closely with the domain experts themselves, that is, the people that will use the software. The users aren't always domain experts. They have experience using the existing process, but do not necessarily understand why it is that way. That is where the domain experts come in. The more you as a developer understand about the business you are writing software for, the better the software will be.

    - Eric Evans in his book Domain-Driven Design: Tackling Complexity in the Heart of Software (Addison-Wesley Professional, 2003) picks up on this point when he mentions the "ubiquitous language." This is a language that is shared between the developers and the business to describe the business domain being modeled.

- *Keeping the process lean by continuous reflection on process and the removal of waste*: Keeping process and practices lean is all about eliminating waste. Don't bother with lots of documentation before developing systems. Create the documentation when it is needed. You should be able to cope with a few architectural diagrams that any member of the team can reproduce on a white board. Instead of masses of requirements documentation, use story or tasks cards and write features that can act as reminders for conversations when it is time to build the feature. Lots of upfront documentation is no good to the business—there is simply no value in it. The amount of documentation that is produced in an agile project is defined as a requirement. It is not true that agile equals no documentation. Agile equals the removal of useless information. The code and the user stories with their corresponding acceptance criteria become the documentation of the project. Not a 400-page, stagnant requirements document.

    - Keeping lean is also achieved with regular retrospectives on work carried out and meetings on what's working and not working with the current processes. Continuously refining how we work and concentrating on the work at hand will contribute towards a leaner and more effective working practice.

- *Strong focus on team effort that spans more than developers in order to reduce risk and find better ways of working*: Agile is about working together with a strong focus on the team in an effort to improve your working practices and ultimately deliver more value for your business. Domain experts, product managers, business analysts, security and IT infrastructure stakeholders, and testers should be first-class citizens along with developers during the project. Including non-developers in the team helps to increase knowledge and shared ownership and decreases the "them and us" gap between developers and everyone else in more traditional methods.

    - Agile development can be the proverbial silver bullet. The problem that occurs has to do with changing the people around you. That being said, an agile project methodology can be very valuable to any organization with a need to be flexible when prioritizing application development.

The Flavors of Agile

There are various forms of agile methodologies, but they all share similar characteristics. You can think of these various methodologies as branches of the same religion. The cornerstone of each branch is the idea of customer satisfaction. They also feature many of the key ideas listed previously, as well as the practices and principles laid out in the Agile Manifesto. The key thing to remember about all the agile flavors is that every one of them is iterative.

Scrum

The Scrum methodology consists of a series of "sprints," typically lasting two to four weeks, each delivering some working, potentially shippable software. The workload of each of these sprints is driven from the "product backlog." The product backlog consists of new features, bug fixes, technical debt, and anything else that will contribute to the end deliverable. A product owner, with help from the customer, prioritizes the product backlog and works closely with the team via regular stand-up meetings and sprint retrospectives. The iterative aspect of Scrum is that this cycle is repeated over and over until the project is complete.

You will look at Scrum in more detail in Chapter 2.

eXtreme Programming (XP)

eXtreme Programming (XP) is strongly focused on customer interaction and involvement. It has the following five values:

- Simplicity
- Communication
- Feedback

- Courage
- Respect

It also follows these twelve practices:

1. Planning Game
2. Small Releases
3. Customer Acceptance Tests
4. Simple Design
5. Pair Programming
6. Test-Driven Development
7. Refactoring
8. Continuous Integration
9. Collective Code Ownership
10. Coding Standards
11. Metaphor
12. Sustainable Pace

In XP, user stories are created to capture requirements from customers. These stories are then estimated by the developers, prioritized by the customer, and then developed into working software on an iteration-by-iteration basis. Continuous planning and delivery underpin the disciplined XP process. It is also worth noting that many of the practices in XP are shared by other branches of agile, like Scrum.

You will take a closer look at XP in Chapter 3.

Crystal

The Crystal group of agile methodologies focuses more on people rather than process. It has a simple set of principles that enhances teamwork by concentrating on communication and the removal of project management noise. It also concentrates teams on the priorities and critical paths of the software development. Like Scrum and XP, it also encourages frequent delivery of working software.

Dynamic Systems Development Method (DSDM)

The Dynamic Systems Development Method (DSDM) is based on the 80/20 rule, in that 80 percent of the benefit a system will be derived from only 20 percent of the systems requirements. With this in mind, only work that is deemed critical for the system to operate is prioritized; that is, the first 20 percent of requirements. DSDM is prioritized using the so-called MoSCoW method, which is as follows:

M: Must have

S: Should have, if at all possible

C: Could have, but not critical