Chapter 1

Introduction to Agile Methods

Agile methods are revolutionizing the approach to software development.

Achieving Competitive Advantage in a Software Economy

The software industry has evolved to become one of the most important industries of our time. Employing millions of practitioners throughout virtually every developed country worldwide, this industry creates some of the most essential products we use to maintain and extend our lifestyles. From controlling the production of the food we eat, to providing safety and control of the vehicles we drive, to providing life-sustaining medical advances and automating the businesses that employ us, software has become the embodiment of much of the world's most valuable intellectual property.

In this intensely competitive environment, what separates the winners from the losers, the leaders from the second-place finishers? In many cases, it is their ability to more quickly create and deliver software solutions that better address their users' and customers' real needs. When it comes to enterprises whose business is the sale of software, the winners can often be characterized by the following:

- They are often first to market.
- Their solutions directly address customers' real pain points, and they have built-in mechanisms to assure that their products do so.
- The solutions they deliver have the requisite quality and functionality.
- They adapt more rapidly to business and technological change than do their competitors.

In other words, the leaders deliver better software more quickly, and they are relentless in constantly enhancing their solutions to assure an ongoing fit to their customers' needs.

This mantra seems simple enough, so why doesn't everybody follow it? The answer lies deep inside the software development process itself, and we have observed that those who master this demanding and difficult process are most likely to emerge as the winners.

Software Development Methods Advance with the Industry

We've been creating large-scale software solutions for just a few decades. While that may seem like an eternity to those of us who have been at it for much of that time, in technological terms it is a relatively short time span. During this period, we've developed and applied a number of methods to control and manage the production of software, starting with the "code it–fix it–code it some more" early practices and progressing through the more structured and formal methods such as the waterfall (sequential, stage-gated) models of development. Many of these methods have been formalized and well documented (IEEE, CMM, DoD, etc.), and yet our results when applying these methods have been spotty. While we have achieved great triumphs, we have also written millions of lines of code that were late, buggy, or that perhaps never even saw the light of actual use. Every practitioner has scars and war stories, and we constantly search for a better way to create software.

ENTER AGILE METHODS

In the last decade or so, the trend to more *agile* and even *extreme* methods has been the most significant event we have seen since the application of the waterfall model in the 1970s. While still debated in some quarters, the benefits of agility, including faster time to market, better responsiveness to changing customer requirements, and higher application quality, are undeniable to those who have mastered these practices. Ranging from Extreme Programming (XP) through the methods of Scrum, DSDM, Feature-Driven Development (FDD), and Lean Software Development to the iterative and incremental methods espoused by the Rational Unified Process and its agile variants, the basic principles of software agility have now been effectively applied in thousands of projects. In addition, a number of supportive agile practices, such as test-first development or test-driven development (TDD), developer-to-developer and developer-to-tester pairing, and shared code ownership, have been applied within these methods and as stand-alone practices as well.

Most agile methods, however, have been defined and recommended primarily to small team environments in which collocation, ready access to interactive customers, and small team size are the defining rule. Are the benefits of agility to be denied to those larger software enterprises that don't share these simple paradigms? Or can the industry learn from these practices and apply

some of the core principles to large-scale development of applications that require 100, 200, or even 1,000 distributed team members to achieve?

AGILE AT SCALE

In the last few years, a number of thought leaders and executives with the courage to try something new have experimented with applying these methods at enterprise scale [BMC and Rally 2006], and the results have been highly encouraging. As we have experienced these various methods in practice, we have also come to understand that while the methods themselves vary, the core practices have much in common. We have also seen that, once mastered, many of these core practices *scale natively* and can be applied directly to the enterprise level where larger and more distributed teams and some degree of outsourcing is the norm.

In Part II of this book, we describe seven agile team practices that natively scale to the enterprise level:

1. The define/build/test (d/b/t) component team
2. Two-level planning and tracking
3. Mastering the iteration
4. Smaller, more frequent releases
5. Concurrent testing
6. Continuous integration
7. Regular reflection and adaptation

The fact that these seven practices work efficiently in the enterprise, large or small, should provide some comfort to those CIOs, vice presidents of development, and other agents of organizational change who look to adopt these methods to improve the software productivity of their larger enterprise.

However, we have also learned that these practices alone do not fully address all the issues that must be addressed to achieve software agility at enterprise scale. To create the agile enterprise requires additional work. In Part III, we describe an additional set of seven agile enterprise practices that a company can master to achieve even more of the benefits of software agility, especially for those with large, distributed teams who are developing large, complex, and long-lived systems and applications. These practices include:

1. Intentional architecture
2. Lean requirements at scale: vision, roadmap, and just-in-time elaboration

3. Systems of systems and the agile release train
4. Managing highly distributed teams
5. Impact on customers and operations
6. Changing the organization
7. Measuring business performance

Taken together, the seven agile team practices that scale and the seven agile enterprise practices will substantially improve the performance of the software enterprise. While the undertaking is significant, the rewards are significant as well, and the benefits of higher productivity, improved time to market, higher quality, and lower support costs, coupled with unleashing the creative and productive power of empowered project teams, will launch the company toward its goal of a *creating the agile enterprise.*

A Look at the Methods

In the last 5 to 10 years, many new and different agile methodologies came into practice. They carried different names, tactics, activities, and acronyms, but they all were aimed at the same problem: creating reliable software more quickly. These methods include:

- Dynamic System Development Method (Dane Faulkner and others)
- Adaptive Software Development (Jim Highsmith)
- Crystal Clear (a family of methods, Alistair Cockburn)
- Scrum (Ken Schwaber, Jeff Sutherland, Mark Beedle)
- XP (Kent Beck, Eric Gamma, and others)
- Lean Software Development (Mary and Tom Poppendieck)
- Feature-Driven Development (Peter Coad and Jeff DeLuca)
- Agile Unified Process (Scott Ambler)

In our experience, the most widely adopted methods in the United States are Scrum and XP, and we use those methods as the cornerstone of many of the practices in this book. In addition, DSDM is a well-documented agile method whose roots lie in a European consortium of companies that came together to forge a common method to improve software outcomes. The movement to Lean Software also emphasizes numerous principles that underlie many agile methods, so we look to the lean movement for core agile philosophies that we can build on. Feature-Driven Development and agile variants of the Unified Process also contribute to our understanding, so we will look at those methods as well.

Agile Manifesto

In 2001, the founders of many of the agile software development methodologies came together with others who were also implementing various agile methods in the field and created an "Agile Manifesto" (www.agilemanifesto.org) summarizing their belief that there is a better way to produce software. The Agile Manifesto was a synthesis of common beliefs that underlie the various methods they were promoting and practicing. It substantially boosted the adoption of agile in the field because it provided common ground for all who headed down this path, and it did an excellent job of synthesizing and defining the core beliefs underlying the agile movement.

The Agile Manifesto reads, in part:

> *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.*

In addition, the participants described a number of key principles that support the philosophy espoused by the Manifesto:[1]

- *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.*

1. These principles and more, including the signers of the original Manifesto, can be found at the Agile Manifesto Web site (www.agilemanifesto.org).

- *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.*

The Agile Manifesto's focus on *continuously delivering working software* while *allowing for and supporting changing requirements* struck a chord in the industry, and this simple explanation of *what really matters most* when it comes to software development can be heard ringing broadly in the industry today. This Manifesto and its supporting principles provide the basic philosophy of agility, and to them, virtually all applied agile best practices can be directly correlated. We'll see these principles, and some of the underlying assumptions of the methods themselves, at work throughout Parts II and III of this book.

THE TREND TO AGILE ADOPTION

The first wave of agile adoption was led by independent software vendors, a few innovative IT shops, and agile consulting companies such as Thoughtworks. Many of these companies had engaged the software development thought leaders, including those listed earlier, as consultants to drive software process improvement. Since then, agile adoption has increased rapidly. According to Forrester Research [2005]:

> *Agile software development processes are in use at 14% of North American and European enterprises, and another 19% of enterprises are either interested in adopting agile or already planning to do so.*

Not surprisingly, the motivations for doing so are the same as those that have driven prior software development process innovations. In a survey of 21

companies using or considering using an agile process, Forrester [2005] notes their reasons:

- Productivity and time to market (66% of respondents)
- Reducing costs (48% of respondents)
- Improving quality (43% of respondents)

In other words, the motivations for agile are the same as always: to develop software *faster, better, and cheaper!*

BUSINESS BENEFITS OF SOFTWARE AGILITY

Even more importantly, now that we have a few years of experience in applying these methods in one form or another, hard data is starting to emerge about the benefits of agility across almost all dimensions important to the software enterprise. In the most comprehensive study to date, an Australian group, Shine Technologies [2003], surveyed 131 respondents from teams and companies that had applied agility. The results were truly eye opening:

- Ninety-three percent stated that productivity was better or significantly better.
- Forty-nine percent stated that costs were reduced or significantly reduced (46 percent stated that costs were unchanged).
- Eighty-eight percent stated that quality was better or significantly better.
- Eighty-three percent stated that business satisfaction was better or significantly better.

At a time when almost any software executive would leap at any model that could provably increase productivity or quality by even a small amount, and also lower cost, the data that emerges is increasingly compelling:

Increases in Productivity

- In the original XP project at Daimler Chrysler, Beck [2000] reports that it took 12 to 15 people 2 years to write and deploy a system that a team of 30 had failed to deliver in the prior 4 years.
- "Having a solid sense of team productivity from years with the waterfall approach, I am amazed with the productivity agile brings. Using agile across our global team of slightly more than 20 engineers, our

team delivered two major enterprise server products, three proof-of-concept systems, and two feature enhancement releases in less than twelve months."[2]

- On one very large-scale project (over 300 practitioners on one large application), BMC Software, Inc. increased individual developer and team productivity by an estimated 20 percent to 50 percent [BMC and Rally 2006].

Increases in Quality

The holy grail of software development has always been to increase both productivity and quality of the software process. This is an area where agile really shines—adopters report increases in quality commensurate with the overall gains in productivity:

> *Our implementation of agile practices . . . helps us find bugs earlier, helps us achieve higher quality, and helps us work well with SW QA.*
>
> —Jon Spence, Medtronic [2005]

> *I measure quality by the life of a defect, time measured from injection to finding and fixing. Agile gives us solid results with most defects living no longer than one to two iterations. Using this measure, I'd have to say that agile delivers higher quality than anything I've found with the waterfall model.*
>
> —Bill Wood, VP Development, Ping Identity Corporation

Increases in Team Morale and Job Satisfaction

Teams who have made the conversion to agile experience increases in morale and job satisfaction, thus providing another important, tangible benefit to the enterprises that adopt agile:

> *Development teams are more engaged, empowered and highly supportive of the new development process.*
>
> —BMC Software, Inc. [BMC and Rally 2006]

> *Our implementation of agile practices . . . (1) makes the work more enjoyable, (2) helps us work together, and (3) is empowering.*
>
> —Jon Spence, Medtronic [2005]

2. Bill Wood, Vice President of Development, Ping Identity Corporation, Denver, CO.

Faster Time to Market

Customers are receiving critical functionality sooner through more frequent releases. With Scrum/agile development, BMC's IMD can now offer customers new releases 3 to 4 times per year.

—BMC Software, Inc. [BMC and Rally 2006]

Before we adopted XP, it took us up to 1½ man years (6 QA engineers for 3 months) to bring a product release from development end to release. Our first XP release required 6 man months (4 QA engineers for 6 weeks). Now we are down to 4.5 man weeks (1½ QA engineers for 3 weeks).

—Mark Striebeck, VA Software [2005]

A Brief Look at XP, Scrum, and RUP

Extreme Programming (XP)

XP is a widely used agile software development method that is described in a number of books by Kent Beck [2000; Beck and Andres 2005] and others. Key practices of XP include the following:

- A team of five to 10 programmers work at one location with customer representation on site.
- Development occurs in frequent builds or iterations, each of which is releasable and delivers incremental functionality.
- Requirements are specified as user stories, each a chunk of new functionality the user requires.
- Programmers work in pairs, follow strict coding standards, and do their own unit testing.
- Requirements, architecture, and design emerge over the course of the project.

XP is prescriptive in scope. It is best applied to small teams of under 10 developers, and the customer should be either integral to the team or readily accessible. In addition, the *P* in XP stands for *programming*, and, as opposed to the other methods of agile development, XP describes some innovative and sometimes controversial practices for the actual writing of software.

Scrum

Scrum is an agile project management method that is enjoying increasing popularity and effective use. Jeff Sutherland and Ken Schwaber [Schwaber and

Beedle 2002] at Easel developed many of the original Scrum practices in 1993. Thereafter, Scrum was formalized and subsequently presented at OOPSLA'96. Since then, Sutherland, Schwaber, and others have extended and enhanced it at many software companies and IT organizations. Key Scrum practices include the following:

- Sprints are iterations of a fixed 30 days' duration.
- Work within a sprint is fixed. Once the scope of a sprint is committed, no additional functionality can be added except by the development team.
- All work to be done is characterized as product backlog, which includes requirements to be delivered, defect workload, as well as infrastructure and design activities.
- A Scrum Master mentors and manages the empowered, self-organizing, and self-accountable teams that are responsible for delivery of successful outcomes at each sprint.
- A daily stand-up meeting is a primary communication method.
- A heavy focus is placed on time-boxing. Sprints, stand-up meetings, release review meetings, and the like are all completed in prescribed times.
- Typical Scrum guidance calls for fixed 30-day sprints, with approximately three sprints per release, thus supporting incremental market releases on a 90-day time frame.

The Rational Unified Process

During the same period that the agile methods were being developed and advanced, the Rational Unified Process (RUP), a software development method and software engineering practice and process framework, was developed by Rational Software Corporation. RUP is a special instance of the more generic Unified Process, which has been characterized in numerous books and is a companion process to the UML. RUP is also available as a commercial product now marketed by IBM's Rational Software Division.

As a process framework, RUP, with its iterative and incremental basis, can be applied in a substantially agile fashion, but the RUP founders were not direct contributors to the agile methods or Manifesto described earlier. Many developers argue that RUP is not as agile as its counterparts. Others argue that RUP, when properly applied, can be extremely agile and is an effective framework for large-scale application of agility. Unarguably, however, RUP has seen very widespread industry adoption and has been applied with good success on tens of thousands of projects of all types, including projects at very large scale.

Also available are a number of lighter-weight and more agile instantiations of RUP, including Agile UP (Scott Ambler),[3] OpenUP[4] and *RUP for Extreme Programming XP Plug-in* (IBM and Object Mentor),[5] so it is likely that RUP and its variants will continue to evolve in substantially agile ways. Because RUP is based on an iterative and incremental foundation that is common to agile methods, it will also contribute heavily to our understanding of how to apply agile methods at scale throughout this book.

SUMMARY

The premise of this work is that *all* the methods mentioned here contribute substantially to a new understanding of a better way to build software at scale. But because every software project presents its unique challenges, it is the essence of the methods that matters most to the enterprise. In order to distill that essence, we next look in more detail at the methods themselves, and, more importantly, at the core principles that underlay them.

3. See http://www.ambysoft.com/scottAmbler.html.
4. See www.eclipse.org/epf.
5. The RUP Plug-In for XP is a joint development project by Rational Software and Object Mentor, a leading advocate of object-oriented technologies and Extreme Programming.

This page intentionally left blank