Compliments of IBM

2nd IBM Limited Edition

Agile

FOR DUMMIES®

A Wiley Brand

Learn:

- How agile teams work in today's mobile, web, and hybrid cloud environments
- What strategies and tooling help you scale your agile practices
- How lean and agile principles are core to a DevOps approach
- Ten agile adoption mistakes and how to avoid them

Amy Silberbauer
Bernie Coyne

Agile

FOR DUMMIES®

A Wiley Brand

2nd IBM Limited Edition

By Amy Silberbauer and Bernie Coyne

FOR DUMMIES®

A Wiley Brand

Agile For Dummies ®, 2nd IBM Limited Edition

Published by
John Wiley & Sons, Inc.
111 River St.
Hoboken, NJ 07030-5774
www.wiley.com

ISBN: 978-1-119-14976-7 (pbk); ISBN: 978-1-119-14978-1 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Publisher's Acknowledgments

Some of the people who helped bring this book to market include the following:

Project Editor: Carrie A. Johnson
Editorial Manager: Rev Mengle
Acquisitions Editor: Steve Hayes
Business Development Representative: Sue Blessing
Production Editor: Antony Sami

Table of Contents

Introduction

Agile development principles have gone from something used only by cutting-edge teams to a mainstream approach used by teams large and small for things as varied as

- Startup software development for mobile and web apps
- Enterprise-wide development efforts
- Complex, large-scale systems engineering initiatives (such as the electronics in the cars you drive and the airplanes you fly in)
- Core (legacy) systems development (which means systems that are at the core of business, such as mainframe-based systems)
- Highly regulated environments (such as healthcare, insurance, or banking)

About This Book

Welcome to *Agile For Dummies,* 2nd IBM Limited Edition. You've probably been hearing about agile for a long time, which isn't surprising. If you're not using agile methods already though, or if you've only been exposed to agile on small projects here and there, you may wonder how to get started with it. Can agile ever work in your environment? Relax. This book is here to help.

Foolish Assumptions

Many people can benefit from this book, but we took the liberty to assume the following about you, our reader:

- You're looking to pilot a project using agile. You're a project manager, a technical lead, or an aspiring product owner who wants to adopt agile practices but aren't sure where to start.

- You may have tried some agile practices in an ad hoc manner, and you encountered some difficulties. Don't worry; many teams experience some missteps when first moving to agile.
- You've had some project success, and you're looking to grow the agile practice beyond your team. You're looking for ways to coordinate multiple teams with the same success you've experienced on your small team.
- You want to try agile, but your environment has complexities that need to be addressed. Maybe you have globally distributed teams or are subject to regulatory compliance mandates.

No matter who you are, this book helps explain the successful agile development practices available today.

Icons Used in This Book

Sometimes, information deserves special attention. The icons in this book identify such information for you. Here's a brief explanation for each icon so you recognize them when they turn up.

The Tip icon points to information that describes a special benefit of working with agile.

This icon identifies pitfalls and problems to avoid in your agile journey.

The Remember icon presents you with tidbits that you won't want to forget after you finish the book.

This icon points out content that gets a little deeper into the weeds of agile development or explains agile jargon you may encounter. The info isn't crucial to your journey, so you can skip it if you like.

Chapter 1

Getting the ABCs of Agile

In This Chapter

- Dissecting the Agile Manifesto
- Defining agile today

If you're reading this book, you've seen software being made. Regardless of your role on the project, you know it's not a perfect process. You know it's hard to do well. Software development doesn't face problems for lack of trying or for lack of brain power. People in the software business tend to be some very bright, hardworking people. They don't plan to deliver software over budget, past deadline, and with defects (or without features people need). So what's been at the root of all these issues?

Agile is an attempt to make the process of software development lean and effective, and it's seen increasing popularity and success. In this chapter, you discover how agile is an incremental, iterative approach to delivering high-quality software with frequent deliveries to ensure value throughout the process. It places a high value on individuals, collaboration, and the ability to respond to change.

Introducing the Agile Manifesto

In February of 2001, a group of developers interested in advancing lightweight development methodologies got together to talk about their views and to find common ground, and agile was born. The developers who created agile understood the importance of creating a model in which each iteration in the development cycle "learned" from the previous iteration. The result was a methodology that was more flexible, efficient, and team-oriented than any of the previous models.

All the agile methods look to the Agile Manifesto and 12 core principles for guidance. The adherence to the guidance provided by the manifesto and principles is what makes a software development team agile, not a specific process, tool, label.

The Manifesto

The *Manifesto for Agile Software Development* is a compact 68 words (now that's lightweight!) that stresses four values.

> **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.

No boxes with arrows, no swim-lane diagrams here. The Agile Manifesto is straightforward, but don't let the brevity fool you. This is powerful stuff. The following sections break down the components of the manifesto.

Individuals and interactions over processes and tools

Recognizing that software is made by people, not processes or tools, agile places a higher premium on people working together effectively. Processes and tools can aid in that but can't replace it.

Working software over comprehensive documentation

Valuing working software over comprehensive documentation stands in stark opposition to the Waterfall model. A highly

detailed, accurate, and comprehensive specification document is of no value if it doesn't result in working software that meets users' needs. Working software may involve documentation, but agile only uses it in service to creating working software, not as an end (almost) unto itself.

Customer collaboration over contract negotiation

While agile isn't ignoring the reality of contracts, it values active collaboration throughout the software development process as a better way to deliver value instead of a carefully worded contract. A contract is no proxy for actual communication when you're doing something as challenging as creating software.

Responding to change over following a plan

Except for the most incredibly simple systems, it's massively difficult to think of every feature, every piece of data, and every possible use case for software. That means, in a collaborative process with the customer, a lot is discovered during the process of developing software. Also, the world changes pretty fast: Business needs and priorities can shift in the months or even years it can take for a large system to be fully built. Agile values the ability to change in response to new discoveries and needs over sticking to a plan created before everything was known.

The 12 principles that drive the Agile Manifesto

The people who wrote the Agile Manifesto later assembled 12 principles that inform and reinforce the manifesto. These further illuminate the things agile values in software development.

The Agile Manifesto follows these principles:

1. The highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity — the art of maximizing the amount of work not done — is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective and then tunes and adjusts its behavior accordingly.

The Agile Manifesto and the principles behind it are published at `www.agilemanifesto.org`.

Redefining Today's Agile

Since the time when the Agile Manifesto was drafted, agile has grown in popularity, and its use has been extended to increasingly larger organizations and more complex projects.

Growing popularity

Agile is a widely accepted and adopted approach to software development. Recent studies have shown that 94 percent of

all organizations are now practicing agile. Hundreds of books on agile exist, covering everything from how to manage agile development to how to apply it in specific industries to how to apply it with specific programming languages. You can attend agile training courses and become an agile coach. Practitioners old and new are blogging about their challenges, discoveries, and successes.

Growing scalability

As the advantages of agile become clear and the number of success stories grows, more and more teams have been attempting to scale agile practices to ever larger and more complex software development projects. Teams are finding success with hybrid approaches that stay true to core agile principles and extend them beyond the software development stage to the entire software life cycle. Chapter 5 covers the scaling of agile practices.

DevOps and agile

Lean and agile development are the underpinnings of the DevOps approach — waste reduction for more efficient teams is one of the results. Efficiency and repetition of best practices lead to shorter development cycles, which allows teams to be more innovative and responsive, thereby increasing customer value. Scaling lean and agile principles beyond the development team to a team of teams and across the entire product and software delivery life cycle is core to the DevOps approach.

Many teams have already adopted agile and want to scale their current processes as part of their DevOps adoption. Many popular frameworks are available to help scale agile. These frameworks include the Scaled Agile Framework (SAFe) and Disciplined Agile Delivery (DAD). Some organizations have also been effective at scaling the Scrum process to very large teams. The purpose of these frameworks is to provide a methodology for adopting agile at the enterprise level. That means taking into consideration not just the development of code but also including architecture, project funding, and governance of the processes and roles required by management, applying the very same lean and agile principles that have

worked well at the team level. No matter the framework used to scale agile, you take those basic agile principles and apply best practices to leverage them to drive efficiency and effectiveness across the enterprise.

Impact of hybrid cloud

The existence of hybrid cloud introduces new challenges for agile development because it results in application delivery pipelines for systems of record and systems of engagement that may span across complex hybrid cloud and physical environments.

Traditional software applications are large systems that function as *systems of record,* which contain massive amounts of data and/or transactions and are designed to be highly reliable and stable. Because these applications don't need to change often, organizations can satisfy their customers and their own business needs by delivering only one or two large new releases a year. With the advent of mobile communications and the maturation of web applications, systems of record are being supplemented by *systems of engagement,* which customers can access directly to interact with the business. Such applications must be easy to use, high performing, and capable of rapid change to address customers' rapidly changing needs and evolving market forces.

The core requirement for adopting agile with a hybrid cloud approach is the need for application deployment across these multiple cloud and physical environments. Applications like IBM's UrbanCode Deploy with Patterns utilize application blueprints to map applications and configurations to multiple environments, physical and cloud, allowing for automated application deployment across complex, hybrid cloud environments.

Chapter 2

Understanding Agile Roles

In This Chapter

- Discovering the primary roles on agile teams
- Taking a look at additional team positions

On an agile project, any given person can act in one or more roles and can also change roles over time. Roles aren't positions, nor are they meant to be. Teams practicing agile adapt to styles that suit their needs and may vary in their exact execution. However, all tend to have the same kinds of roles and very similar processes at their core. Agile deemphasizes specialized roles and considers all team members equal — everyone works to deliver a solution regardless of their jobs. With the exception of *stakeholder,* everyone is effectively in the role of *team member.* In this chapter, you explore the roles, arming you with a basis for understanding any style you may experience.

Being a Stakeholder

A *stakeholder* is someone who's financially impacted by the outcome of the solution and is clearly more than an end-user. A stakeholder may be one of the following:

- A direct or indirect user
- A manager of users
- A senior manager
- An operations or IT staff member
- The "gold owner" who funds the project
- An auditor(s)

- Your program/portfolio manager
- A developer(s) working on other systems that integrate or interact with the one under development
- A maintenance professional(s) potentially affected by the development and/or deployment of a software project

Representing Stakeholders: The Product Owner

The *product owner* is the team member who speaks as the "one voice of the customer." This person represents the needs and desires of the stakeholder community to the agile delivery team. He clarifies any details regarding the solution and is also responsible for maintaining a prioritized list of work items that the team will implement to deliver the solution. While the product owner may not be able to answer all questions, it's his responsibility to track down the answer in a timely manner so the team can stay focused on its tasks. Each agile team has a single product owner.

The product owner has the following additional roles:

- Communicates the project status and represents the work of the agile team to key stakeholders
- Develops strategy and direction for the project and sets long- and short-term goals
- Understands and conveys the customers' and other business stakeholders' needs to the development team
- Gathers, prioritizes, and manages product requirements
- Directs the product's budget and profitability
- Chooses the release date for completed functionality
- Answers questions and makes decisions with the development team
- Accepts or rejects completed work during the sprint
- Presents the team's accomplishments at the end of each sprint
- Defines objectives for the current iteration

Being a Team Member

The role of *team member* focuses on producing the actual solution for stakeholders. Team members perform testing, analysis, architecture, design, programming, planning, estimation, and many more activities as appropriate throughout the project.

Not every team member has every single skill (at least not yet), but they have a subset of them and strive to gain more skills over time. Team members identify, estimate, sign-up for, and perform tasks and track their completion status.

Assuming the Team Lead

The *team lead* guides the team in performing management activities; agile teams don't have a *manager.* The team lead is a servant-leader to the team, upholding the conditions that enable the team's success. This role is also an agile coach who keeps the team focused on delivering work items and fulfilling its iteration goals and commitments to the product owner. The team lead facilitates communication, empowers the team to self-optimize its processes, ensures that the team has the resources it needs, and manages issue resolution in a timely manner. In Scrum, this role is the Scrum Master (see Chapter 4 for more information).

While an experienced team lead brings skills to a new team, this person can't be a true coach without mentoring. So for teams new to agile, you may have a part-time experienced coach working with the team for a few iterations.

Acting As the Architecture Owner

Architecture is a key source of project risk, and someone has to be responsible for ensuring the team mitigates this risk. The *architecture owner* is the person who owns the architecture decisions for the team and who facilitates the creation and evolution of the overall solution design.

You may not have to formally designate a team member as an architecture owner on small teams, because the person in the role of team lead often is the architecture owner as well.

Stepping Up As an Agile Mentor

A mentor is a great idea for any discipline in which you want to develop new expertise. The *agile mentor,* sometimes called an *agile coach,* implements agile projects and shares that experience with a project team. He provides valuable feedback and advice to new project teams and to project teams that want to perform at a higher level. The agile mentor is

- A coach only and isn't part of the team
- Often from outside the organization and objective in guidance without personal or political considerations
- Experienced in implementing agile techniques and running agile projects in different situations

Looking at Agile Secondary Roles

Your project may include the need to add some or all of the following secondary roles:

- **Domain expert:** Someone with deep business/domain knowledge beyond that of the product owner.
- **Specialist:** Although most agile team members are generalizing specialists, sometimes specialists such as business analysts or even project/program managers are required.
- **Technical expert:** Technical experts are brought in as needed to help the team overcome a difficult problem and to transfer their skills to one or more developers on the team.
- **Independent tester:** Some agile teams are supported by an independent test team working in parallel that validates work throughout the life cycle.
- **Integrator:** For complex environments, your team may require one or more people in the role of integrator responsible for building the entire system from its various subsystems.

Chapter 3

Getting Started with Agile

In This Chapter

- Looking at agile planning practices
- Managing and tracking your progress
- Reflecting for future improvement

In this chapter, you explore how the agile team organizes the software development process. Everything the stakeholders want in their software is broken down into small chunks, ranked, worked on in priority order over short iterations (typically one to four weeks), reviewed for approval, and delivered to production. This process repeats until the prioritized list is finished, called a *release.* An agile team expects re-prioritization and additions to and subtractions from the list throughout the process but embraces them as a means to deliver the most value and the best possible solution.

Agile Planning

Teams following agile software development methods typically divide their release schedule into a series of fixed-length development iterations of two to four weeks — shorter is generally better than longer. Planning involves scheduling the work to be done during an iteration or release and assigning individual work items to members of the team.

To be effective and to reflect the team's status and direction, plans need to be accessible to everyone on the team and able to be changed quickly and easily over the course of the iteration. Automated planning tools are available to facilitate this process, but you can do it the old-fashioned way with whiteboards, index cards, or sticky notes.

During an agile project, planning occurs at three levels:

- **Release planning:** Release plans contain a release schedule for a specific set of features. The product owner creates a release plan at the start of each release.
- **Iteration planning:** Team members gather at the beginning of the iteration (referred to as a *sprint* in the Scrum methodology) to identify the work to be done during that iteration. This is referred to as *self-organization.*
- **Daily planning:** Development teams begin each day with standup meetings to plan the day. These meetings are generally 5 to 15 minutes long.

Attending the Daily Coordination Meeting

On agile projects, you create plans throughout the entire project every day. Agile development teams start each workday with a 15-minute (or less) daily coordination meeting to note completed items, to identify impediments, or roadblocks requiring team lead involvement, and to plan their day.

In the daily coordination meeting, often called a daily standup meeting, each development team member makes the following three statements:

- Yesterday, I completed *[state items completed].*
- Today, I'm going to take on *[state task].*
- My impediments are *[state impediments, if any].*

Daily coordination meetings can actually be quite fun when using the right tools. For example, the Taskboard view in IBM Rational Team Concert (RTC) is a capability that arranges all work items as cards on a board with multiple columns. For more info on the Taskboard view, see Chapter 8.

Creating User Stories

When stakeholders realize the need for a new software system, feature set, or application, the agile process begins

with the product owner defining what the software will do and what services it provides to its users. Instead of following the more traditional process of product managers and business analysts writing lengthy requirements or specifications, agile takes a lightweight approach of writing down brief descriptions of the pieces and parts that are needed. These become work items and are captured in the form of user stories. A *user story* is a simple description of a product requirement in terms of what that requirement must accomplish for whom. Your user story needs to have, at a minimum, the following parts:

- Title: <*a name for the user story*>
- As a <*user or persona*>
- I want to <*take this action*>
- So that <*I get this benefit*>

The story should also include validation steps — steps to take to know that the working requirement for the user story is correct. That step is worded as follows:

- When I <*take this action*>, this happens <*description of action*>

User stories may also include the following:

- **An ID:** A number to differentiate this user story from other user stories.
- **The value and effort estimate:** *Value* is how beneficial a user story is to the organization creating that product. *Effort* is the ease or difficulty in creating that user story.
- **The person who created the user story:** Anyone on the project team can create a user story.

  For user stories that are too large to be completed in a single iteration or sprint, some teams use Epics. *Epics* are basically a higher-level story that's fulfilled by a group of related user stories.

Figure 3-1 shows a typical user story card, back and front. The front has the main description of the user story. The back shows how you confirm that the requirement works correctly after the development team has delivered the requirement.

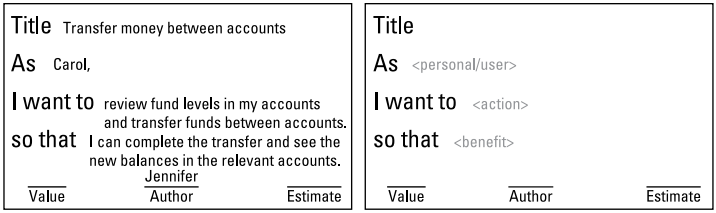


Figure 3-1: Card-based user story example.

The product owner gathers and manages the user stories. However, the development team and other stakeholders also will be involved in creating and decomposing user stories.

User stories aren't the only way to describe product requirements. You could simply make a list of requirements without any given structure. However, because user stories include a lot of useful information in a simple, compact format, they tend to be very effective at conveying exactly what a requirement needs to do.

Estimating Your Work

When the product owner sets the scope of an iteration, she needs to know that the scope is achievable — that there isn't too much work to get done in the iteration. As with any other development process, the agile process requires the estimation of work. However, unlike other processes, agile estimates work via a *relative* ranking process based on a measure of complexity, called *points*.

Points are assigned in whole numbers (1, 2, 3, and so on with no fractions or decimals) and represent relative sizes and complexity of work items. Small and simple tasks are one point tasks, slightly larger/more complex tasks are two point tasks, and so on.

Points are kind of like t-shirt sizes. There are small, medium, large, extra large, and potentially other sizes (extra small and extra extra-large). These sizes are relative — no regulation dictates how much larger medium is compared to small. Sizes vary a bit from manufacturer to manufacturer. T-shirt sizing succeeds not because of high precision — it's pretty imprecise, actually — but through its general accuracy and its

relative nature that requires little upfront investment to get the ball rolling. Relative ranking is truly lean!

In practice, one team's three-point size estimate for a work item may correlate to another team's two-point estimate for an identical work item. Teams need only agree on what size and complexity corresponds to what point count and remain internally consistent in their use.

You may be tempted to equate points to hours. Don't do it! An agile team gains consistency by using point values that don't vary based on the ability of the person doing the work. A five-point story has the same size and complexity on a given team regardless of who does the work. The team still accommodates faster and slower team members in the number of points assigned in a single iteration, but the value delivered to the product owner and stakeholders (measured by size and complexity) remains consistent. If you want to track effort and hours, keep it separate from points.

Tracking Velocity

At the end of each iteration, the agile team looks at the requirements it has finished and adds up the number of story points associated with those requirements. The total number of completed story points is the team's *velocity,* or work output, for that iteration. After the first few iterations, you'll start to see a trend and will be able to calculate the average velocity.

The average velocity is the total number of story points completed, divided by the total number of iterations completed. For example, if the development team's velocity was . . .

Iteration 1 = 15 points

Iteration 2 = 19 points

Iteration 3 = 21 points

Iteration 4 = 25 points

. . . your total number of story points completed is 80. Your average velocity is 20 to 80 story points divided by four

iterations. After you've run an iteration and know the team's velocity, you can start forecasting the remaining time on your project.

Measuring Progress with Burndown Reports

Burndown reports track the number of points completed and are used for monitoring single iterations, releases, and the entire project backlog. They get their name from the concept that the iteration or project backlog gets "burned down," completed, and cleared away. Burndown reports show progress, reflecting both the complexity delivered (in points) and the team's velocity. See Figure 3-2 for a simple project burndown report, with iterations along the X-axis and the points in the entire product backlog on the Y-axis.

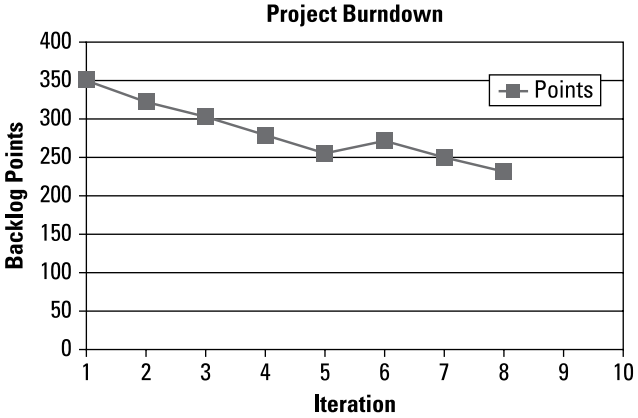


Figure 3-2: A project burndown report.

You may also find that you need to re-estimate the backlog. As the team progresses through a project, it discovers more about that project, its technology, and the business concerns the project addresses. Greater understanding leads to better estimates, so the points associated with some work items in the backlog can become more accurate over time.

Test-Driven Development

Testing occurs throughout the agile life cycle. While an independent tester may or may not be a member of the cross-functional team, developers are expected to test their code. Some of you are saying, "Hold on, developers can't test their own work. They don't want to. They're not good at it!" But trust me; it's not as bad as you think. In fact, it's not bad at all.

Agile teams use two common practices to handle their testing:

- Test-Driven Development (TDD)
- Automated unit tests

When used together, they're a powerful force.

Performing TDD means that before the developer writes a piece of code, she first writes a small test that validates the code she's about to write. She runs the test to make sure it fails and then writes the code that makes the test pass. This may seem odd, but with practice it's much more efficient than writing a lot of code, running it, and going back later to figure out everywhere it's broken (a process known as *debugging*). This process puts the developer in a testing mindset while writing code, which leads to higher-quality code.

When a developer executes a small test against code she's written, it's called a *unit test.* When these tests are run in batches all at once (automated), they become very powerful. Agile teams write a lot of unit tests, automate them, and run them frequently against the code they write as individuals and against their combined code that makes up the entire application. Running automated unit tests frequently against the code reveals problems quickly so they can be addressed quickly. This approach finds defects long before they'd ever reach a traditional test cycle, which means higher-quality applications.

Shift Left Testing

The term *Shift Left* refers to a practice in software development where teams focus on quality, work on prevention instead of detection, and begin testing earlier than ever

before. The goal is to increase quality, shorten long test cycles, and reduce the possibility of unpleasant surprises at the end of the development cycle — or still worse, in production. In many organizations, automated testing of today's composite applications is being executed via the user interface after the complete application has been developed and deployed. Waiting for all the pieces to become available before testing commences, however, often causes delays, adds risk to the project, or results in discovery of late stage defects — when they're more expensive to fix.

Shift Left practices help to avoid rework, delays, and churn that can occur when major defects are discovered late in the testing cycle — after all integrations and product components are finally brought together as a composite application and made available for the team to test. It aims to avoid these issues by performing integration tests as code is delivered and deployed in a realistic production-like test lab. If any of the dependent application components aren't available to test, virtual services can mimic the real components' behavior until they're ready.

IBM Rational Test Workbench is a leading test automation solution that helps teams address their test automation needs, which include integration, functional, regression, and performance testing. The IBM Rational Test Virtualization Server solution enables organizations to deploy virtual services or "stubs" to simulate dependent software and services across the enterprise where they can be shared by the entire delivery team.

Continuous Integration and Deployment

Continuous integration (CI) is the practice of regularly integrating and testing your solution to incorporate changes made to its definition. Changes include updating the source code, changing a database schema, or updating a configuration file. Ideally, when one or more changes are checked into your configuration management system, the solution should be rebuilt (recompiled), retested, and any code or schema analysis performed on it. Failing that, you should strive to do so at least once if not several times a day.

Continuous deployment (CD) enhances CI by automatically deploying successful builds. For example, when the build is successful on a developer's workstation, she may automatically deploy her changes to the project integration environment, which would invoke the CI system there. A successful integration in that environment could trigger an automatic deployment into another environment and so on. CI ensures high-quality working software at all times, and CD ensures that the software is running in the right place.

IBM UrbanCode Deploy is an example of an automation tool for continuous deployment across your environments. It is designed to facilitate rapid feedback and continuous deployment for agile development while providing the audit trails, versioning, and approvals needed in production.

Presenting Results at the Iteration Review

The *iteration review,* or *sprint review* in Scrum, is a meeting to review and demonstrate the user stories that the development team completed during the iteration. The iteration review is open to anyone interested in reviewing the iteration's accomplishments. This means that all stakeholders get a chance to see progress on the product and provide feedback.

Preparation for the iteration review meeting should not take more than a few minutes. Even though the iteration review might sound formal, the essence of showcasing in agile is informality. The meeting needs to be prepared and organized but doesn't require a lot of flashy materials. Instead, the iteration review focuses on demonstrating what the development team has done.

Collecting Feedback in the Iteration Review Meeting

Gather iteration review feedback informally. The product owner or team lead can take notes on behalf of the

development team, as team members often are engaged in the presentation and resulting conversation. New user stories may come out of the iteration review. The new user stories can be new features altogether or changes to the existing code.

In the first couple of iteration reviews, the team lead may need to remind stakeholders about agile practices. Some people hear the word *demonstration* and immediately expect fancy slides and printouts. The team lead has a responsibility to manage these expectations and uphold agile values and practices.

The product owner needs to add any new user stories to the product backlog and rank those stories by priority. The product owner also adds stories that were scheduled for the current iteration, but not completed, back into the product backlog, and ranks those stories again based on the most recent priorities. The product owner needs to complete updates to the product backlog in time for the next iteration planning meeting.

Learning and Improving at the Iteration Retrospective

After the iteration review is over (see the preceding section), the iteration retrospective begins. The *iteration retrospective* is a meeting where the team lead, the product owner, and the development team discuss how the iteration went and what they can do to improve the next iteration. If the team agrees, other stakeholders can attend as well. If the team regularly interacts with outside stakeholders, and it should, then those stakeholders' insights can be valuable.

You may want to take a break between the iteration review and the iteration retrospective. The team needs to come into the retrospective ready to inspect its processes and present ideas for adaptation.

Agile approaches quickly reveal problems within projects. Data from the iteration backlog shows exactly where the development team has been slowed down, so the product owner should revisit the backlog to prepare for the next iteration.

Chapter 4

Choosing an Agile Approach

In This Chapter

- Understanding various agile approaches
- Looking at the strengths and weakness of each approach

You can use several agile methods as a base to tailor a strategy to meet the unique needs of your situation. This chapter discusses these varied approaches.

Scrum: Successful Teaming

Scrum is the most popular approach to agile software development. Scrum provides a simple framework for teams to work together collaboratively developing small increments of the application or product in short sprints (Scrum calls iterations *sprints*). Each successive sprint builds upon the previous one. This approach encourages a focus on only what is needed and enables teams to demonstrate success continuously and obtain feedback early. Scrum teams are composed of the following primary roles:

- A Product Owner who identifies what needs to be built for each sprint
- The Development Team who build and demonstrate the working software
- A Scrum Master who coaches the team in understanding and executing the Scrum process

Five practices, covered in detail in Chapter 3, are key to Scrum. They are release planning, sprint planning, the daily scrum meeting, the sprint review meeting, and the sprint retrospective.

Scrum of Scrums is a technique for scaling Scrum to a teams-of-teams level. Each Scrum team proceeds normally, but one person from each team attends a Scrum of Scrums meeting to coordinate the work between teams. For more information on Scrum, visit `scrum.org`.

XP: Putting the Customer First

A popular approach to product development, specific to software, is *Extreme Programming* (XP). The focus of XP is customer satisfaction. XP teams achieve high customer satisfaction by developing features when the customer needs them. New requests are part of the development team's daily routine, and the team must deal with requests whenever they crop up. The team organizes itself around any problem that arises and solves it as efficiently as possible. The following are XP practices:

- **Coding standard:** Team members should follow established coding guidelines and standards.
- **Collective ownership:** Team members may view and edit other team members' code or any other project artifact. Collective ownership encourages transparency and accountability for work quality.
- **Continuous integration:** Team members should check in changes to their code frequently, integrating the system to ensure that their changes work, so the rest of the team is always working with the latest version of the system.
- **Test-Driven Development (TDD):** In TDD the first step is to quickly code a new test — basically just enough code for the test to fail. This test could either be high-level acceptance or a more detailed developer test. You then update your functional code to make it pass the new test, get your software running, and then iterate.
- **Customer tests:** Detailed requirements are captured just-in-time (JIT) in the form of acceptance tests (also called *story tests*).

- **Refactoring:** Refactoring is a small change to something, such as source code, your database schema, or user interface, to improve its design and make it easier to understand and modify. The act of refactoring enables you to evolve your work slowly over time.
- **Pair programming:** In this practice, two programmers work together on the same artifact at the same time. One programmer types the code while the other programmer looks at the bigger picture and provides real-time code review.
- **Planning game:** The purpose of the planning game is to guide the product into successful delivery. This includes high-level release planning to think through and monitor the big issues throughout the project as well as detailed JIT iteration/sprint planning.
- **Simple design:** Programmers should seek the simplest way to write their code while still implementing the appropriate functionality.
- **Small releases:** Frequent deployment of valuable, working software into production is encouraged. Frequent deployments build confidence in the team and trust from the customer.
- **Sustainable pace:** The team should be able to sustain an energized approach to work at a constant and gradually improving velocity.
- **Whole team:** Team members should collectively have all the skills required to deliver the solution. Stakeholders or their representatives should be available to answer questions and make decisions in a timely manner.

Lean Programming: Producing JIT

Lean has its origins in manufacturing. In the 1940s in Japan, a small company called Toyota wanted to produce cars for the Japanese market but couldn't afford the huge investment that mass production requires. The company studied supermarkets, noting how consumers buy just what they need, because they know there will always be a supply, and how the stores

restock shelves only as they empty. From this observation, Toyota created a JIT process that it could translate to the factory floor.

The result was a significant reduction in inventory of parts and finished goods and a lower investment in the machines, people, and space. The JIT process gives workers the ability to make decisions about what is most important to do next. The workers take responsibility for the results. Toyota's success with JIT processes has helped change mass manufacturing approaches globally.

The seven principles of lean manufacturing can be applied to optimize the whole IT value stream. The lean software development principles are eliminate waste, build in quality, create knowledge, defer commitment, deliver quickly, respect people, and optimize the whole.

Kanban: Improving on Existing Systems

The Kanban method is a lean process that describes techniques for improving your approach to software development. Two Kanban principles critical to success are

- **Visualizing workflow:** Teams use a Kanban board (often a whiteboard, corkboard, or electronic board) that displays *kanbans* (indications of where in the process a piece of work is). The board is organized into columns, each one representing a stage in the process, a work buffer, or queue; and optional rows, indicating the allocation of capacity to classes of service. The board is updated by team members as work proceeds, and blocking issues are identified during daily meetings.
- **Limit work in progress (WIP):** Limiting WIP reduces average lead time, improving the quality of the work produced and increasing overall productivity of your team. Reducing lead time also increases your ability to deliver frequent functionality, which helps build trust with your stakeholders. To limit WIP, understand where your blocking issues are, address them quickly, and reduce queue and buffer sizes wherever you can.

Agile Modeling

Agile Modeling (AM) is a collection of values, principles, and practices for modeling software that can be applied on a software development project in an effective and lightweight manner. AM was purposely designed to be a source of strategies that can be tailored into other base processes.

With an Agile Model Driven Development (AMDD) approach, you typically do just enough high-level modeling at the beginning of a project to understand the scope and potential architecture of the system. During construction iterations you do modeling as part of your iteration planning activities and then take a JIT model storming approach where you model for several minutes as a precursor to several hours of coding. AMDD recommends that practitioners take a test-driven approach to development although doesn't insist on it.

The Agile Modeling practices include the following:

- **Active stakeholder participation:** Stakeholders (or their representatives) provide information, make decisions, and are actively involved in the development process.
- **Architecture envisioning:** This practice involves high-level architectural modeling to identify a viable technical strategy for your solution.
- **Document continuously:** Write documentation for your deliverables throughout the life cycle in parallel to the creation of the rest of the solution. Some teams choose to write the documentation one iteration behind to focus on capturing stable information.
- **Document late:** Write deliverable documentation as late as possible to avoid speculative ideas likely to change in favor of stable information.
- **Executable specifications:** Specify detailed requirements in the form of executable customer tests and your detailed design as executable developer tests.
- **Iteration modeling:** Iteration modeling helps identify what needs to be built and how.
- **Just barely good enough artifacts:** A model needs to be sufficient for the situation at hand, and no more.

- **Look-ahead modeling:** Invest time modeling requirements you intend to implement in upcoming iterations. Requirements near the top of your work item list are fairly complex so explore them before they're popped off the top to reduce overall project risk.
- **Model storming:** Do JIT modeling to explore the details behind a requirement or to think through a design issue.
- **Multiple models:** An effective developer has a range of models in his toolkit, enabling him to apply the right model for the situation at hand.
- **Prioritized requirements:** Implement requirements in priority order, as defined by your stakeholders.
- **Requirements envisioning:** Invest your time at the start of an agile project to identify the scope of the project and create the initial prioritized stack of requirements.
- **Single-source information:** Capture info in one place only.
- **TDD:** Quickly code a new test and update your functional code to make it pass the new test.

Disciplined Agile Delivery (DAD)

Disciplined Agile Delivery (DAD) is a process framework that encompasses the entire solution life cycle, acting like a hybrid of the best practices from many agile approaches. DAD sees the solution from initiation of the project through construction to the point of releasing the solution into production.

The project is carved into phases with lightweight milestones to ensure that the project is focused on the right things at the right time, such as initial visioning, architectural modeling, risk management, and deployment planning.

This differs from methods such as Scrum and XP, which focus on the construction aspects of the life cycle while details about how to perform initiation and release activities, or even how they fit into the overall life cycle, are typically missing.

For more information on this topic, see *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise,* by Scott W. Ambler and Mark Lines (IBM Press, 2012).

Scaled Agile Framework (SAFe)

The *Scaled Agile Framework* (SAFe) is rooted in the principles of other agile approaches described previously: lean thinking, iterative and incremental development, and agile development. Based on the acknowledgment that agile in isolation doesn't imply organizational success, SAFe prescribes a set of best practices and guidance encapsulated in a framework to expand these very same principles across the enterprise.

SAFe describes an organizational structure that includes business and engineering roles collaborating to deliver more value to customers as well as to the business. The framework considers not only the development of code but also architecture, project funding, and governance, and it describes the roles and processes applicable at three organizational levels: Team, Program, and Portfolio.

SAFe-based projects enable Teams and Programs to focus on improving efficiency and effectiveness by synchronizing the alignment, collaboration, and delivery of value to the business, as defined by the Program vision and roadmap. The Program is essentially an uber Scrum team that focuses on delivery of demonstrable value in shorter increments with clearly defined objectives and success criteria, integrating the technology delivered by agile Teams that participate in that Program.

For more information on SAFe, check out Chapter 7.

Unified Process (UP)

The *Unified Process* (UP) uses iterative and incremental approaches within a set life cycle. UP focuses on the collaborative nature of software development and works with tools in a low-ceremony way. It can be extended to address a broad variety of project types, including OpenUP, Agile Unified Process (AUP), and Rational Unified Process (RUP).

UP divides the project into iterations focused on delivering incremental value to stakeholders in a predictable manner. The iteration plan defines what should be delivered within the iteration, and the result is ready for iteration review or shipping. UP teams like to self-organize around how to accomplish iteration objectives and commit to delivering the results. They do that by defining and "pulling" fine-grained tasks from a work items list. UP applies an iteration life cycle that structures how micro-increments are applied to deliver stable, cohesive builds of the system that incrementally progress toward the iteration objectives.

UP structures the project life cycle into four phases: Inception, Elaboration, Construction, and Transition. The project life cycle provides stakeholders and team members with visibility and decision points throughout the project. This enables effective oversight and allows you to make "go or no-go" decisions at appropriate times. A project plan defines the life cycle, and the end result is a released application.

Chapter 5

Scaling Agile Practices

In This Chapter

- Understanding lean and agile for the enterprise
- Seeing how agile can scale
- Incorporating additional roles on large teams

Agile approaches have been adapted to work in a wide range of environments, not just those involving small, co-located teams that dominate the early agile literature. Agile strategies are being applied throughout the entire software delivery life cycle as well, not just in the construction phases, and often in very complex environments that require far more than a small, co-located team.

Every project team finds itself in a unique situation, with its own goals, abilities, and challenges to overcome. What they have in common is the need to adopt, and then tailor, agile methods, practices, and tools to address those unique organizational demands.

Why Lean and Agile for the Enterprise?

Technology and market trends driven by mobile, cloud, and big data are requiring a different approach to software delivery. This approach demands focus on providing more value to customers quickly with higher quality and at lower cost. The adoption of agile practices has proven successful in helping teams address these imperatives in their move to a continuous delivery model, but agile in isolation doesn't imply organizational success.

Critical factors for success in continuous delivery are well aligned with many of the lean and agile principles. The factors include the following:

- Consider carefully and implement rapidly.
- Continuously improve.
- Identify and eliminate waste.
- Favor the creation of working software over documentation.
- Build integrity.
- See the whole.

Consider a team-based example: Lean and agile thinking guides teams to deliver in smaller increments and get early feedback. As a result, teams reduce cycle time by focusing only on those activities that maximize value based on feedback. Wasted effort is identified and eliminated, enabling teams to spend time on value-add activities, such as innovation and quality improvements.

Now consider scaling that process across the enterprise. By applying lean and agile principles across the teams, the organization can align as a whole to focus on what really matters — getting ideas into production quickly so customers can use them and provide feedback.

Understanding What It Means to Scale

In the early days of agile, projects managed via agile development techniques were small in scope and relatively straightforward. The small, co-located team strategies of mainstream agile processes still get the job done in these situations. Today, the picture has changed significantly and organizations want to apply agile development to a broader set of projects.

Organizations often deal with problems that require large teams; they want to leverage a distributed workforce; they want to partner with other organizations; they need to comply

with regulations and industry standards; they have significant technical or cultural issues to overcome; and they want to go beyond the single-system mindset and truly consider cross-system enterprise issues effectively.

Not every project team faces all these scaling factors or each scaling factor to the same extent, but all these issues add complexity to your situation, and you must find strategies to overcome these challenges.

To deal with the many business, organizational, and technical complexities your development organization faces, your agile delivery process needs to adapt, or scale. The following sections describe the factors most organizations must consider when scaling their agile projects.

Large teams

Mainstream agile processes work very well for smaller teams of 10 to 15 people, but what if the team is much larger? What if it's 50 people? 100 people? 1,000 people? As your team-size grows, the communication risks increase and coordination becomes more difficult. As a result paper-based, face-to-face strategies start to fall apart.

Distributed teams

What happens when the team is distributed — perhaps on floors within the same building, different locations within the same city, or even in different countries? What happens if you allow some of your engineers to work from home? What happens when you have team members in different time zones? Suddenly, effective collaboration becomes more challenging and miscommunication is more likely to occur.

Compliance

What if regulatory issues — such as Sarbanes Oxley, ISO 9000, or FDA CFR 21 — are applicable? This may mean that the team has to increase the formality of the work that it does and the artifacts that it creates. It also means that traceability is a requirement, not just a nice-to-have capability.

Domain complexity

Some project teams find themselves addressing a very straightforward problem, such as developing a data entry application or an informational website. Sometimes the problem domain is more intricate, such as the need to monitor a bio-chemical process or air traffic control. Or perhaps the situation is changing quickly, such as financial derivatives trading or electronic security assurance. More complex domains require greater emphasis on exploration and experimentation, including — but not limited to — prototyping, modeling, and simulation.

Organization distribution

Sometimes a project team includes members from different divisions, different partner companies, or from external services firms. The more organizationally distributed teams are, the more likely the relationship will be contractual in nature instead of collaborative.

A lack of organizational cohesion can greatly increase risk to your project due to lack of trust, which may reduce willingness to collaborate and may even increase the risks associated with ownership of intellectual property (IP).

Technical complexity

Some applications are more complex than others. Sometimes you're working with existing legacy systems and legacy data sources that are less than perfect or that require care and precision when applying changes. Other times, you're building a system running on several platforms or implementing in different technologies. Sometimes, the nature of the problem your team is trying to solve is just very complex in its own right, requiring a carefully considered, intricate solution.

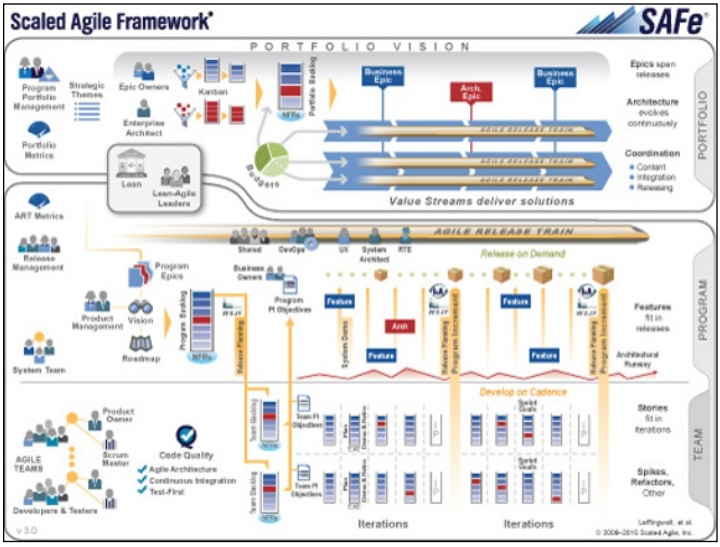
Organizational complexity

Your existing organizational structure and culture may reflect waterfall values, increasing the complexity of adopting and scaling agile strategies within your organization. Or some groups within your organization may wish to follow strategies that aren't perfectly compatible with the way yours wants to work.

Organizing Large Teams

When a team consists of 30 or more team members, it's considered large. To achieve success, large teams must be divided into subteams that are organized collectively around a common vision, deliver solutions on a unified cadence, and drive team-level work from a single organization-wide backlog of features prioritized based on business strategy and objectives. Large teams can still be agile — and should be — but effectiveness is dictated by how well the business and IT roles communicate and collaborate constantly to drive the delivery of value to the customer and to the business.

Large teams must be careful to clearly define activities and ownership of the process and deliverables. Roles, by specific names, aren't particularly important. Figure 5-1 is one example of how a large team may be effectively structured, based on the Scaled Agile Framework (SAFe). For more information on SAFe, see Chapter 7.



Reproduced with permission from © 2011-2015 Scaled Agile, Inc. All rights reserved.

Figure 5-1: Looking at the structure of a large team with SAFe.

In this image, large teams are organized across business and IT roles that collaborate together to plan the development and delivery of value to the business. Collectively, the following types of activities are addressed:

- **Business strategy and investment:** The business owners collectively decide how budgets are allocated based on potential return on investment, or value/cost, estimates. Here, long term business initiatives are captured and end-user capabilities along with architectural strategy are allocated a percentage of the budget. Prioritization at this level drives the proposed delivery of features at the next level.
- **Solution delivery coordination:** Roles involved in the coordination of solution delivery typically involve business and IT leadership working together to drive IT development from business objectives. These roles are responsible for the content that's delivered, the proposed roll out of that delivery, and the overall roadmap and vision of each project. They're also responsible for ensuring the business achieves return on investment. This team, often called the *Program Management Team,* coordinates the delivery of functionality across multiple agile teams and includes product managers, release managers, business owners, and senior IT leadership.
- **Technology delivery coordination:** This is typically owned by small agile teams. In true agile fashion, the teams are self-governing and have technological decision-making authority as well as content authority over the function that's delivered. They're guided by the prioritization of features at the solution level and coordinate their own team's delivery with those of other teams that are part of solution delivery. Roles involved are typically IT skills and include product owners, developers, and testers with the right skills for the deliverables of that team.

In large teams, there's a greater need for shared models, documentation, and plans, particularly if the team is geographically dispersed. This is where use of tooling automation that enables cross-team collaboration, tracking and planning, reporting, and artifact change is critical.

Chapter 6

Considering Enterprise DevOps

In This Chapter

- Knowing why you need DevOps
- Understanding the capabilities of DevOps

DevOps (short for development and operations), like most new approaches, is little more than a buzzword to many people. Everyone talks about it, but not everyone knows what it is. In broad terms, DevOps is an approach based on lean and agile principles in which business owners and the development, operations, and quality assurance departments collaborate to deliver software in a continuous manner that enables the business to more quickly seize market opportunities and reduce the time to incorporate customer feedback. Much of the DevOps approach is based on the agile development principles and practices founded more than a decade ago. DevOps is taking those same principles and broadening them from the development team to encompass the whole enterprise. Indeed, enterprise applications are so diverse and composed of multiple technologies, platforms, end-user devices, and so on that only a DevOps approach will be successful when dealing with these complexities.

Understanding the Business Need for DevOps

Organizations want to create innovative applications or services to solve business problems. They may want to

address internal business problems (such as creating a better customer relationship management system) or help their customers or end-users (such as by providing a new mobile app). While many developers and teams have adopted agile techniques and have seen good results, most organizations still struggle to deliver software projects, and their failures are often related to challenges in the broader view of the development life cycle.

Although most enterprises feel that software development and delivery are critical, a recent IBM survey of the industry found that only 25 percent believe that their teams are effective. This execution gap leads to missed business opportunities.

This problem is further amplified by a major shift in the types of applications that businesses are required to deliver across systems of record and systems of engagement. Because systems of engagement are used directly by customers, they require intense focus on user experience, speed of delivery, and agility — in other words, a DevOps approach that has its roots in lean and agile principles. Flip back to Chapter 1 for a primer on systems of record and systems of engagement.

Looking at DevOps Capabilities

The capabilities that make up DevOps are a broad set that span the software delivery life cycle. Where an organization starts with DevOps depends on its business objectives and goals — what challenges it's trying to address and what gaps in its software delivery capabilities need to be filled.

A *reference architecture* provides a template of a proven solution by using a set of preferred methods and capabilities. The DevOps reference architecture helps practitioners access and use the guidelines, directives, and other material that they need to architect or design a DevOps platform that accommodates people, processes, and technology.

A reference architecture provides capabilities through its various components. These capabilities in turn may be provided by a single component or a group of components that work together. Therefore, you can view the DevOps reference architecture, shown in Figure 6-1, from the perspective of the core capabilities that it's intended to provide. As the abstract

architecture evolves to concrete form, these capabilities are provided by a set of effectively enabled people, defined practices, and automation tools.

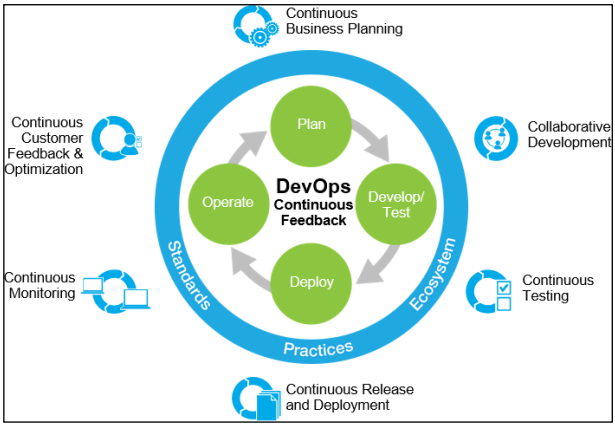


Figure 6-1: The DevOps reference architecture.

Adopting DevOps

Adopting any new capability typically requires a plan that spans people, process, and technology. You can't succeed in adopting new capabilities, especially in an enterprise that has multiple, potentially distributed stakeholders, without taking into consideration all three aspects of the capabilities being adopted.

At its root, DevOps is a cultural movement; it's all about people. An organization may adopt the most efficient processes or automated tools possible, but they're useless without the people who eventually must execute those processes and use those tools. Building a collaborative culture in conjunction with the right business process is imperative to the success of DevOps adoption.

DevOps as a capability affects a whole business. It enables the business to be more agile and improves its ability to deliver value to customers. You can extend DevOps further by looking at it as a business process, which is a collection of activities or tasks that produces a specific result (service or product) for customers. In the reference architecture, shown

in Figure 6-1 in the preceding section, the DevOps business process involves taking capabilities from the idea (typically identified with business owners) through development and testing to production and customers.

Technology enables people to focus on high-value creative work while delegating routine tasks to automation. Technology also allows teams of practitioners to leverage and scale their time and abilities. If an organization is building or maintaining multiple applications, everything it does has to be repeatable, in a reliable manner, to ensure quality across all applications. It can't start from scratch with each new release or bug fix for every application. The organization has to reuse assets, code, and practices to be cost-effective and efficient.

Standardizing automation also makes people more effective. Organizations may experience turnover in employees, contractors, or resource providers; people may move from project to project. But a common set of tools allows practitioners to work anywhere, and new team members need to learn only one set of tools — a process that's efficient, cost-effective, repeatable, and scalable.

For more information on this topic, read *DevOps For Dummies,* 2nd IBM Limited Edition, `ibm.biz/DevOpsForDummies` or visit `jazz.net/products/devops` and `ibm.com/devops`.

Chapter 7

Using the Scaled Agile Framework (SAFe)

In This Chapter

- Understanding SAFe concepts
- Adopting SAFe

The Scaled Agile Framework (SAFe) is a process framework that encompasses the entire solution life cycle. From business investment and budgeting that informs portfolio plans over time, to analysis of value and cost that determines expected return on investment for business initiatives, to the incremental development and delivery of work prioritized in alignment with business objectives and strategy, SAFe embraces lean and agile principles and provides guidance to help organizations scale those principles at an enterprise level.

SAFe is a framework, not a solution or a tool. In fact, use of tooling that supports SAFe is just one factor in achieving success. You also must consider the people and processes in place to help you address the business imperative of delivering value more quickly, with higher quality and lower cost.

DevOps and SAFe

DevOps supports the continuous delivery model through an approach that embraces lean thinking, to help drive organizational efficiency and effectiveness. The Scaled Agile Framework (SAFe) is the market-leading process framework for scaling lean and agile across the enterprise, providing guidance and best practices to help organizations realize

success. A DevOps solution grounded in SAFe methodology provides a comprehensive process and tooling framework that enables an optimized end-to-end life cycle to synchronize teams in the planning and execution of software delivery.

SAFe with IBM's DevOps solution, for example, is a combination of capabilities that enables team-based development across the enterprise, which typically involves multiple people, processes and technologies.

Three principles

SAFe prescribes that all roles in the organization collaborate in the planning and execution of software development and delivery, incorporating the very same principles that have enabled teams to successfully move to a continuous delivery model.

Respond quickly to feedback and change

Teams successfully adopting agile and lean principles are making great strides in their ability to reduce time to feedback, aided in large part by tooling that facilitates this process. With the very same tooling and SAFe best practices built in, this success is more readily applied across the organization. A DevOps solution provides the ability to monitor feedback across the delivery pipeline, including feedback from build, testing, and deployment phases as well as from business stakeholders and customers. It also allows teams to easily collaborate; SAFe provides the guidance and best practices to apply lean and agile principles in making decisions about how to respond to change.

Focus on quality

The meaning of quality is transforming. While high quality at the team level still does equate to a low number of defects, that's not the end of the story. Lean prescribes that you see the whole, which means that delivery of value to customers (and therefore the business) is the embodiment of feature/function delivery across multiple teams. In the planning and execution of high-quality software, the focus must shift from individual product features to an overall view of what customers want — or helping them understand what they want and

delivering capabilities they will enjoy using. The ability to capture the meaning of quality, informed by SAFe and supported by the tooling, and then being able to track the status and health of quality delivery, is critical.

Avoid unintended consequences

Without a broad understanding of value delivery across the organization, quick decisions to adjust plans in response to feedback can lead to unintended consequences — shortcuts in quality, missed deadlines, and more. Teams in isolation may make decisions that optimize their own ability to deliver, but without the vital insight provided by tooling that manages changes across teams, those decisions could limit the delivery ability of the organization as a whole.

Why SAFe?

The question really is "why not SAFe?" There are many frameworks out there supporting agile, scaled agile, and combinations of these and other approaches promising *process nirvana.* The Scaled Agile Framework (SAFe) provides a well-established, industry-standard way to apply lean and agile principles across all organizational levels in the enterprise, uniting business and engineering roles to deliver more value to customers and to the business. The framework considers not just the development of code, but also architecture, project funding, and governance, and describes the roles and processes applicable at three levels: team, program, and portfolio.

Extensive case studies provided by Scaled Agile, Inc. point to the effectiveness of applying lean and agile principles to the enterprise to deliver business results:

- 20-50 percent increase in productivity
- 30-75 percent faster time to market
- 50+ percent defect reduction
- Increase in employee engagement

For more details, visit `www.scaledagileframework.com/case-studies`.

Adopting SAFe

Enterprise-wide SAFe adoption can be challenging, so incremental adoption is definitely the way to go. Enterprise organizations typically have at least some teams successfully applying agile methodologies, so the SAFe Program is a logical starting point for most organizations. As skills in SAFe adoption grow and best practices are developed, additional SAFe Programs can be established. Finally, a SAFe Portfolio can be established to drive software development and delivery across these multiple SAFe Programs.

This incremental approach addresses the needs of small and mid-range IT shops as well as large, regulated IT organizations based on common patterns of required support in the environment to

- Balance simplicity with scalability
- Provide flexibility for complex, heterogeneous organizations
- Include robust requirements and quality management as well as portfolio planning

By starting with planning, tracking, and reporting capabilities focused on a SAFe Program to develop and deliver a single project or application, and then expanding capabilities and scaling to more programs within a portfolio over time, organizations have the opportunity to "learn by doing" and develop their own best practices and repeatable processes.

SAFe Support: An IBM Example

IBM provides support for SAFe through tooling that encapsulates the artifacts, activities, roles, and events prescribed by SAFe along with mentoring to help drive process adoption in an evolutionary way.

The SAFe Program

The SAFe Program is supported by the SAFe 3.0 Process (Program) process template offered in IBM Rational Team

Concert. The solution provides agile planning and reporting capabilities consistent with SAFe as well as links in context to SAFe best practices and guidance to help organizations adopt the SAFe methodology. The process template is a quick and easy way to establish a SAFe Program project area complete with roles, an Agile Release Train timeline, work item types and attributes, work flows, plan views, and dashboards prescribed by the SAFe framework. Reports built on SAFe Team and Program-level metrics are provided along with capabilities to create additional reports through a simple report builder interface.

The SAFe Portfolio

Once organizations begin to realize success in scaling lean and agile practices across teams, IBM's solution can be expanded to support the SAFe Portfolio. Today, the SAFe Portfolio can be supported through the out-of-the-box configuration capabilities of IBM Rational DOORS Next Generation and IBM Rational Team Concert to define the business-level artifacts such as Strategic Theme, Lightweight Business Case, Value Stream and Portfolio Epic. IBM Rational DOORS Next Generation provides robust requirements articulation and management capabilities and is well-suited to manage the end-user requirements being driven by the portfolio. By combining this capability with work flow supported in IBM Rational Team Concert change management, the SAFe Kanban system prescribed for Portfolio and Program Epics offers complete support for SAFe across the portfolio.

Considerations for regulated IT

Regulated IT organizations have additional requirements to ensure a level of compliance above and beyond that of smaller or less complex organizations, including the need to ensure that a clear and precise audit trail exists as part of enacting change. Today, IBM supports this kind of enterprise by ensuring complete top-down and bottom-up traceability of artifacts dictating change across the portfolio.

By providing SAFe support in IBM's DevOps solution, IBM is uniquely positioned to support scaling lean and agile principles across organizations while meeting the requirements for regulatory compliance.

What's next?

As you consider SAFe adoption and then perhaps commit to leading a SAFe transformation of your own, you can benefit from collaboration with others who are marching down that path or have already achieved some level of success. To get started, consider joining the industry-wide SAFe communities, such as the Scaled Agile Framework, Inc. Check out the website at `http://scaledagileframework.com` or visit IBM's SAFe site: `bit.ly/ibmsafesupport`.

Chapter 8

Evaluating Agile Tools

In This Chapter

- Knowing the key criteria for selecting agile tools
- Using Software as a Service (SaaS) for rapid agile development
- Managing your enterprise agile process with IBM Rational Team Concert

To support and automate the agile process in your organization, you can consider incorporating tools that facilitate a streamlined process for your agile teams. This chapter helps you make smart decisions when considering software purchases that serve your agile development needs.

Considering Key Criteria for Selecting Agile Tools

Each team approaches agile in a different way. Some teams start small, adopting Scrum and simple approaches using open source tools, while others require more extensive agile life cycle management solutions. Regardless of where they attain them, most agile teams make use of specific agile tools that help them get the most from their agile process.

Before you decide to use a new tool, determine whether it's the best tool for your needs. The easiest way to do this is to keep these seven key criteria in mind as you make your selection:

- **Core agile capabilities:** First and foremost, the agile tool must support people over process by facilitating collaboration, planning, and productivity among the agile team, product owners, and other stakeholders.

- **Integrated and open agile delivery:** The agile tool must allow you to protect investments in existing tooling and minimize tool maintenance with simplified integrations.
- **Team collaboration in context:** The ability for team members, customers, managers and stakeholders to work together in context of the task at hand allows the team to focus on delivering working software.
- **Life cycle traceability:** The ability to understand how your actions affect others, to find gaps in test coverage, and be aware of defects that are blocking progress help your team identify and mitigate potential risks and reduce friction across the agile delivery life cycle.
- **Agile development analytics:** Arming managers and agile team members with real-time metrics to make informed decisions helps reduce friction and accelerate the velocity of agile teams.
- **Adaptability and flexibility:** Your tool should support your team regardless of its process or size. Look for adaptable process support that evolves as your needs change.
- **Agility at scale:** As your organization grows, it most likely needs a well-defined agile scaling process, team structure, and tooling to address real-world complexities, such as those faced by distributed teams or teams addressing compliance requirements.
- **Hybrid cloud and mobile:** In an enterprise, mobile apps are typically not stand-alone apps. They have very little business logic on the mobile device itself and serve more as front-ends to multiple enterprise applications already in use by the enterprise. Look for agile tooling that can support this type of hybrid development.

The right tools can help you succeed, regardless of your entry point. Your challenge is to identify your greatest need today, the improvements you need to make to current practices, and whether new tools are really the solution. For more information on evaluating agile tools, visit `ibm.com/software/rational/agile`.

Using the Best Tool for the Job

The ideal agile tool addresses project planning, work item tracking, source code management, continuous builds, and adaptive process support, enabling agile project teams and stakeholders to work together effectively.

IBM Rational Team Concert (RTC) provides all of these features in one integrated tool to help the project team collaboratively plan, execute, and deliver working applications. The following sections describe the key capabilities RTC offers to support agile teams. For more information on RTC features, visit `jazz.net/projects/rational-team-concert/features`.

RTC provides core capabilities, including agile tracking and planning, source code management optimized for distributed teams; continuous integration supporting personal, team, and integration builds; and customizable work item tracking to support agile, traditional, or hybrid teams.

Process awareness and customizability

Regardless of the size or maturity of your agile team, your agile tooling should give you the ability to deploy, customize, enact, and improve agile processes. RTC can improve the productivity of your teams and the quality of the work they produce by allowing each team to teach the tool its best practices. RTC uses this knowledge to automate team processes, allowing team members to focus on building great software. RTC comes with the Scrum and SAFe process templates built-in.

Team awareness

Your agile tooling should make collaboration easier across all stages of the life cycle. All team members, including stakeholders, should have access to real-time, role-relevant information with full traceability to related tasks, as well as the ability to easily make contributions to projects.

RTC knows your project teams, their internal organization, and the artifacts they are working on. It greatly simplifies the access to team-related information or performing team-related operations. In addition, RTC integrates with IBM Sametime, IBM Connections, GoogleTalk, and Skype, which help enhance collaboration when teams are geographically distributed.

Planning

Agile planning is all about keeping everyone on the same page and marching to the same beat. RTC provides capabilities to create product, release, and iteration plans for teams; to create individual plans for developers; to track the progress during an iteration; and to balance the workload of developers. In addition, RTC provides planning views to support different agile approaches including Taskboards for daily standups and Kanban to manage flow (see Figure 8-1). RTC also provides cross-project plans that allow tracking of work items that have dependencies on other work items in other projects.

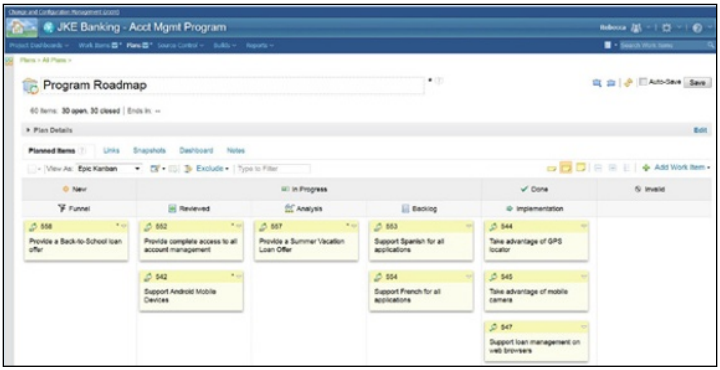


Figure 8-1: The Kanban board view.

Regardless of the process used, plans are accessible to everyone on the team via the web, Eclipse, or Visual Studio. All plans change dynamically over the course of the project to reflect the team's position and direction.

Figure 8-2 is an example of Quick Planner — a lightweight tracking and planning capability included with IBM Rational Team Concert.

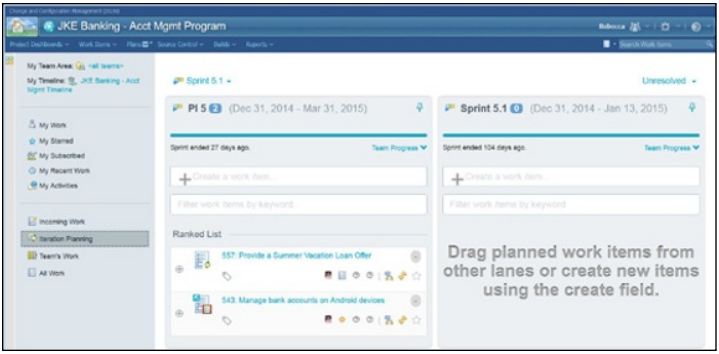


Figure 8-2: The Quick Planner.

Transparency/project health

Transparency and the ability to monitor status in real-time is vital to a successful agile project. Team members should have visibility into potential issues that could impede their progress, and managers need to have real-time information to proactively manage risks.

The RTC dashboards and reports help all team members keep tabs on the health of their projects. The Web Dashboard, as seen in Figure 8-3, provides a personalized view of plan status, work items, event feeds, reports, and other items that are critical to understanding your progress. Reports provide both real-time views and historical trends of velocity, builds, streams, work items, and other artifacts that your team works with. In addition, RTC supports the use of OpenSocial Gadgets and IBM iWidgets to extend visibility to other commercial and open source life cycle tools.

Broad platform support

The trend toward more interconnected mobile and cloud systems is leading to an increasing number of cross technology development projects that are deployed on multiple target platforms. RTC provides the ability to centrally manage, develop, and track multi-platform projects, providing visibility to all team members and stakeholders. RTC supports popular development platforms, including Windows, Linux, IBM System z, IBM Power, and Solaris.

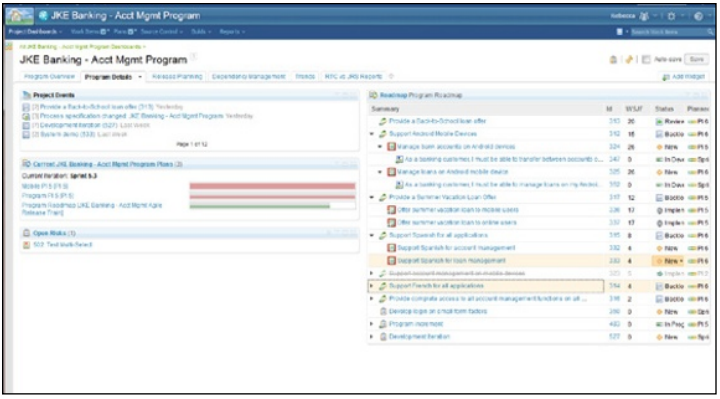


Figure 8-3: The Web Dashboard.

RTC not only provides cross platform support but also integrated development environments (IDE) for Eclipse and Microsoft Visual Studio that allow developers to collaborate across teams, track projects, manage source code, resolve defects, and execute builds.

Extending tooling beyond core agile development

While RTC serves as the core agile development tool, IBM also provides depth and breadth of tools that allow organizations to adapt and automate beyond development across the entire life cycle. Depending on which scaling factors apply (see Chapter 5), organizations can extend RTC with a vast portfolio of capabilities including more vigorous requirements for envisioning and management, test management, deployment automation, architecture and asset management, and business collaboration. In addition, RTC can be integrated with popular open source tools, including Subversion, Git, CVS, Maven, Hudson, and Jenkins. For a complete list of integrations visit `jazz.net/projects/rational-team-concert/integrations`.

Choosing a Delivery Platform

Today, you can choose from several platforms for developing applications. You used to be able to use only on-premise

tooling for enterprise applications, where you installed, configured, and maintained the entire hardware, software, and development tool stack yourself. Now, you also have cloud choices with both SaaS and hosted options that make for more flexible and cost-effective development solutions. No one solution is best for all development so typically companies are using a mix of all three delivery platforms today.

On-premise

On-premise is still where the majority of enterprise development occurs. Sometimes there isn't a business justification to shift all development to the cloud — especially for systems of record applications. Often, security and privacy concerns also cause enterprises to want to keep the development and runtime all in-house.

Cloud managed

One step toward the cloud is simply moving your development infrastructure and tooling to a managed cloud service. This type of service typically hosts the development tools you're already using but transparently provides access to your agile tooling just as if it was on-premise. This causes minimal disruption to your current processes and allows you and your team to focus on development activities by freeing up development resources from maintaining the development infrastructure. This means your team doesn't have to worry about hardware, operating systems, and development tool upgrades, patches, and availability because the cloud managed service provider looks after all that. Cloud managed solutions also allow you to scale up and down and pay for the computing resources only as you need them.

An example of this capability is the IBM Cloud Managed DevOps service that includes a cloud hosted IBM Rational Team Concert solution. For more information visit `ibm.com/ibm/devops/us/en/cloud`.

Software as a Service

Software as a Service (SaaS) is the new way of rapidly building, deploying, and managing composable business

applications for web and mobile, while tapping into an ecosystem of available services and runtime frameworks. A variety of development tools are available on a pay-as-you-go basis so you can easily add or remove them as your needs demand. An organization may choose to use SaaS as a development and runtime platform for its web and mobile applications while continuing to use on-premise solutions for development of the back-end office applications.

A popular example of SaaS for developers is IBM Bluemix DevOps Services. For more information, visit `ibm.biz/DevOpsServices`.

Chapter 9

Making the Move to Agile: IBM's Journey

In This Chapter

- Empowering development teams
- Creating processes to work with distributed teams
- Tooling up for success
- Understanding the software reuse initiative
- Seeing the benefits of using agile

In 2006, IBM faced a unique challenge. In a period of about five years, the company made 70 software acquisitions that resulted in a flood of new software and resources across 90 major labs and involved a number of people working remotely. This transition led to approximately 27,000 people working over five continents. IBM struggled in the software development area due to this changing dynamic that led to an organization comprised of multi-cultural, geographically dispersed and distributed development teams.

IBM needed to be more agile. The success of IBM's agile adoption depended on significant changes in three key areas: people, process, and tools.

People

Agile requires empowering people to make significant changes in how software is developed and delivered and also how it's planned across the organization — yes, *planned.* Although pure agile thrives on a team's ability to respond to feedback

and make adjustments quickly without the burden of having to document and change plans, an enterprise can't operate that way.

Training

IBM helped the software teams change by setting up a center of excellence, holding a two-day disciplined agile workshop that trained approximately 9,000 people over a period of two years, and running additional focused workshops for in-depth training on practices that included a collaborative leadership workshop. IBM identified key resource dependencies (either people or technology) and made them available to the teams when they needed help adopting agile.

This center of competency continues to operate today as more and more teams adopt agile. Furthermore, IBM has initiated a company-wide center of competency to help teams and organizations adopt agile DevOps capabilities consistently.

Culture

A change in development culture can be a challenge for some people. In a number of cases, particularly at the beginning of the process, some teams struggle to accept or adopt agile partly because they're so dispersed, and their components are developed in different geographic locations.

At IBM, the cultural challenges were addressed in part by sending an agile coach to each development lab for three to six months. Having an agile practitioner in these locations was worth the cost, because the rate of failure dropped. IBM also had to change the views of development from the top-down, not just at the team level. The executives were used to having an initial state and plan and relating them to waterfall measurements. They worked to accept that scope changes over time and that this was a good thing — as long as IBM kept release rates and quality levels high.

Organizational structure

Because IBM's complex products vary from testing tools to compilers to database management systems to application

servers and more, it needed to have more than one type of team. So, IBM created teams better aligned with the things they deliver:

- **Feature teams:** Responsible for a complete customer feature (products and components). The goal was to optimize customer value. Feature teams minimize dependencies, use iterative development, and track dependencies between adoptions with an adoption tool.
- **Component teams:** Responsible for only part of a customer feature. The development is more sequential and dependencies between teams lead to additional planning. They track execution in plan items — either in work areas or work items.
- **Internal open source:** When a component needs to evolve quickly yet is needed by multiple development teams, IBM uses open source strategies as an effective development method. IBM has deep experience in open source development, so this was an easy team structure to adopt.

Process

After agile was implemented in a number of development teams, some teams were successful, but many more teams had trouble. The teams with the most success were housed in a single location and highly collaborative. The teams that struggled were globally distributed with hundreds of people. IBM realized that it had to change some processes to make agile work better in these distributed environments.

So IBM approached scope differently, while still locking in the other pieces of the project, by dividing it into two types:

- **Release-defining scope:** Usually consumes about 70 percent of the schedule, which is all the things that will cause the project to slip if problems arise.
- **Extended content:** Consists of things that won't cause problems if they aren't shipped.

IBM is now better at delivering and IBM teams have a clearer picture of what's important and what's optionally important.

Other changes IBM made to its processes included

- Ensuring collaboration with stakeholders every iteration for timely and effective feedback
- Organizing its agile methodology into smaller, consumable practices that can be adopted easily
- Making all parts of the production process agile, not just development

  The goal was to make it difficult to go back to old waterfall ways, and the process has been successful.
- Adopting a robust planning strategy

  Each activity, whether it is going on vacation or writing code or testing, is captured in a work item that is estimated and prioritized.

Tools

The final step in IBM's transformation to adopt agile was to take advantage of tooling that allowed teams and teams of teams to do real work without letting the cultural challenges and process changes get in the way. The goal was to deliver the right things right with high quality.

Collaboration

Studies show that face-to-face collaboration is best for agile development, but this isn't always possible in a typical enterprise organization. IBM is no different. To address the need for robust and consistent collaboration, regardless of whether teams are distributed or co-located, IBM adopted widespread use of collaborative development tooling anchored on IBM Rational Team Concert to help developers, testers, teams, and teams of teams stay informed about project changes as they happen (see Chapter 8 for more info).

IBM Rational Team Concert not only helps agile teams manage change by enabling collaboration within the team, but it helps teams and the organization itself continuously plan and adjust in an agile way by providing visibility into decision-making at all levels. If a project lead and management have a conversation about a change, everyone on the team is notified

about it, the reason for it, who's affected, and how they're affected. These tools also have repositories so plans and work items can be stored and used in the future by others.

Visibility and traceability

Because IBM is a large organization, it has to be intentional about the way it communicates and collaborates, but it has to do that in a way that doesn't hamper team agility and progress. The key to successful collaboration is to provide teams with a means of exchanging information as part of their everyday work. Posting a bunch of information to a wiki won't get people to pay attention. Exchanging information has to feel natural and fit into the context of the teams' work.

The collaboration capabilities of IBM Rational Team Concert are extremely valuable. When that collaboration is combined with the ability to capture status and health of execution and also provide visibility into that information to all stakeholders involved in development and delivery, the tooling is priceless. With the help of tooling, IBM has been able to automate in a way that supports agile adoption seamlessly, easily, and naturally.

For example, traditional project managers used to canvas individuals about progress, but by the time they could enter the notes into a spreadsheet, the team had moved to the next activity. Now, tools track progress and agile team leads focus on how people work with each other. At the same time, the team leads refocused on leadership activities over technical management activities, thereby adding greater value to their teams and to IBM as a whole.

Transparency and traceability provided by the tooling are also critical. When you have a transparent view of the work items that are complete, the items that still need to be done, and team visibility into how certain tasks are taking more effort than originally scoped, the information helps the team achieve success in adopting agile methodology and drives conversations about whether the teams need to realign resources to stay on track. In addition, the tooling goes beyond development of source code to involve teams producing documentation, providing training and translation as well as release and deployment. Everyone can see the status on complementary work activities required to deliver software.

The Software Reuse Initiative

Component reuse is one the holy grails of software development. If you reuse code, you typically get better quality and productivity, reduced, and more consistency in the end-user experience. You can try to *mandate* reuse, but that really doesn't work well. So instead, IBM created a site that emulated an open source environment and called it *community sourcing*. With this site, developers can create their own projects, share those projects with others, gain access to the code, and decide whether the projects meet their needs.

IBM Rational Team Concert is used by the community as an environment for collaboration, and the community has recently started using IBM Bluemix DevOps Services for new projects. IBM now has over 30,000 developers using the community source site and 4,800 uses of components in its products now. For example, a component that helps IBM's installation has been reused by 152 products, and one component for security has been used by 175 projects.

Reaping the Benefits of Agile

IBM's agile transformation took some adjustments. The company made some mistakes along the way, of course, but it learned from them and has now experienced extensive improvements in quality, time-to-market, and customer satisfaction. IBM is proof that the rewards of agile adoption far outweigh the challenges of changing the culture, adopting new processes, and adapting to new tooling.

IBM's results from agile adoption include the following:

- A 31/69 percent ratio of maintenance to innovation (start was 42/58 and goal was 50/50)
- Customer touches increased from 10 to 400
- Customer calls decreased to 19 percent
- Customer defect arrivals decreased from 6,900 to 2,200
- Defect backlog reduced to 3 months from 9 months or more
- Reduced cost of poor quality cut almost in half

Chapter 10

Ten Common Agile Adoption Pitfalls

In This Chapter

- Knowing what mistakes to avoid
- Creating a better agile adoption experience

The ability to avoid common agile adoption pitfalls may seem daunting, but there's a light at the end of the tunnel. With over 10 years of experience helping customers manage and execute their agile transformations, we present you with this chapter to help briefly explore some common pitfalls organizations encounter when adopting agile strategies. Hopefully, with this advice, you can avoid making the same mistakes.

Focusing Only on Construction

You can realize the spirit of the Agile Manifesto through many approaches. Ironically, most of these approaches focus on one phase or discipline within the delivery life cycle — which goes against the spirit of lean that advises you to consider the whole. Most approaches focus on the construction phase.

Construction is typically a straightforward area to focus on when embarking on an agile transformation, but if organizations only change the way they construct software, they can't necessarily call themselves agile. The development teams could be humming along, delivering new working software every two weeks, but if the processes in Operations only allow for deployment every six months or if the Help Desk is unable to handle the churn or if customer stakeholders aren't prepared to meet regularly, the organization isn't realizing all the benefits agile can provide.

Becoming Agile Zombies

Organizations fall into the trap that if they attend a class and mandate a certain out-of-the-box (OOTB) process, they're automatically agile. They train their teams to blindly follow and enforce the anointed process without considering which practices may need to change to meet their organizations' unique needs.

Agile isn't just prescribed process or a set of practices; it's a philosophy that can be supported by a practice and no two agile approaches are the same. A single methodology that fulfills all needs doesn't exist.

Improper Planning

That old adage, "If you fail to plan, plan to fail," is really true. Planning is core to the success of any agile adoption. Organizations should answer these questions:

- Why do we want to be agile, and what benefits will agile provide?
- How will we achieve and measure agility?
- What cultural, technological or governance barriers exist, and how do we overcome them?

Without a plan that clearly shapes the initiative and includes addressing and resolving constraints to agility (for example, removing waterfall process checkpoints or getting support from other required entities), it is more difficult to shape the initiative, staff it, fund it, manage blockers, and maintain continued executive sponsorship.

Excluding the Entire Organization

You can quickly short circuit an agile adoption by working in the vacuum of a single software or system delivery team. A single team can gain *some* benefit from agile, but to be truly

successful, you need to look at the whole process around solution delivery. And many people are involved in that process.

Agile should be a change in culture for the entire organization. Find champions in Operations, lines of business, product management, Marketing, and other functional areas to increase your success.

Lack of Executive Support

An effective agile adoption requires executive sponsorship at the highest level. This involvement means more than showing up at a kickoff meeting to say a few words. Without executive sponsorship supporting the overall initiative, the agile adoption is often doomed because agile initiatives require an upfront investment of resources and funding — two areas that executives typically control.

Going Too Fast

Moving to agile is very exciting, and it can be tempting to jump right in, pick a process, get some tools, and hit the ground running. Unfortunately, if a proper roadmap for coaching, process, and tooling isn't outlined early in the adoption you can run into issues like the following:

- No defined processes for dealing with multiple dependent or distributed teams
- Scalability issues with the core agile tools
- Extending the tool to support deployment, testing, or business collaboration

Insufficient Coaching

Because an agile adoption isn't just a matter of a new delivery process, but is also major cultural shift, coaching is imperative. Developers don't like change and many people like working in their own world. As a result, the concept of not only changing the way they develop, but adding the concept that now they have to work closely with five, six, or ten other people all the time can be downright horrifying.

A coach can work with these team members and help them through the early phases of agile adoption. Have you known of a teacher or coach that possessed a unique ability to inspire students to stretch their skills and perform at higher levels? Good agile coaching can have the same affect and make the difference between the success and failure of an agile adoption.

Retaining Traditional Governance

When an organization plans its agile adoption, it needs to evaluate all current processes and procedures and whether they inhibit or enhance agility. Existing traditional governance processes can be very difficult to change due to internal politics, company history, or fear that compliance mandates may be negatively impacted. Some common governance areas that are overlooked but can have dramatic impacts to agility are project funding, change control, and phase gates.

Skimping on Training

Organizations often see agile practice training, like coaching, as an area where they can save money, sending only a few key leads to learn the new process in hopes that they can train the rest of the organization while trying to implement the new approach. Agile involves a change in behavior and process. It is critical to send all team members to the appropriate training and provide them with ongoing training to reinforce agile values and update team members on processes that may have changed.

Skimping on Tooling

Agile tooling should support and automate an organization's process. Ensuring all team members consistently use tools impacts the success of the project. If tools are used inconsistently, metrics may not correctly reflect the correct status, builds could be run incorrectly, and overall flow and quality issues result. See Chapter 8 for more information on agile tools.

Chapter 11

Ten Myths about Agile

In This Chapter

- Sorting agile facts from the fiction
- Understanding how agile can work for you

Although agile is an established development approach, it represents a new way of life for the organizations that adopt it. The prospect of working in iterations instead of with a linear approach is unsettling to managers and developers who are deciding whether to make the leap to agile. They fear that focused efforts will be compromised and that control over projects and development teams will be sacrificed. Nothing is further from the truth. This chapter debunks these myths and others to show that agile is most organizations' best bet for success.

Agile and Traditional Don't Mix

Even teams that use traditional or formal processes can take advantage of agile methodology and realize benefits from adopting agile and lean principles. Organizations can also manage the development and delivery across teams that are using a mix of agile and traditional processes while also incorporating agile methodology.

The trick for traditional development teams is to apply the principles in a pragmatic way that enables shorter delivery life cycles, response to feedback early and often, and a willingness to tolerate changing requirements as a result of feedback.

Agile Means "We Don't Plan"

With agile's reliance on collaboration instead of big documents, it may seem like no planning occurs. But in reality, the planning is incremental and evolutionary, which has been proven successful (instead of planning all at once early in a project).

The key in agile development is that change is assumed, *not* that planning doesn't occur. In this sense, a committed plan exists, but it's always subject to adjustments based on feedback, which ensures a focus on value delivery.

Agile Means "No Documentation"

Agile teams keep documentation as lightweight as possible, but they do document their solutions as they go. They follow strategies, such as documenting continuously and writing executable specifications.

Agile Is Only Effective for Co-located Teams

Sure, *ideally* agile teams are located within proximity of one another, but in this day and age, most development teams are distributed. Just remember, to succeed, you need to adopt practices and tooling that build team cohesion. If you use the proper tools, your team doesn't have to be collocated to work effectively together.

Agile Doesn't Scale

Agile definitely scales. Large teams must be organized differently, and this is where the Scaled Agile Framework (SAFe) can provide guidance (check out Chapter 7 for more info on SAFe). They also need more than index cards and whiteboard sketches. Large agile teams succeed by using automation products like IBM Rational DOORS Next Generation for requirements modeling and end-user feedback, IBM UrbanCode Deploy for automating the deployment of complex

applications and associated middleware and database changes, and IBM Quality Manager to validate the end-user capabilities are delivered.

Agile Is Unsuitable for Regulated Environments

Regulated environments are those that are subject to some regulatory mandates, such as medical device companies, financial services businesses, governmental departments and offices, the healthcare field, and more. These organizations are audited from time to time to ensure they comply with regulations. To be agile doesn't mean that organizations are not tracking change, and, in fact, with the right tooling in place to automate traceability, teams are able to be agile without thinking about auditability. It's built in.

Agile Means We Don't Know What Will Be Delivered

Because agile is an iterative process, it provides the opportunity not just for greater control but *better* control over building the right things in the life cycle than one would have with the more traditional approaches. At the end of each iteration, the development team presents working software to the product owner for feedback. Furthermore, teams adopting the Scaled Agile Framework (SAFe) explicitly articulate end-user capabilities that deliver value to customers at the beginning of the project so the organization knows what will be delivered by the teams and, more importantly, what customers can do with it! See Chapter 7 for more details.

Agile Won't Work at My Company

For many companies, the biggest challenge they face when considering agile is the cultural change that must occur to make it successful. Agile explicitly dictates constant

monitoring of feedback loops and adjustment based on that feedback. Some developers and managers may feel more exposed to scrutiny as a result. In addition, executives and business owners may sense a loss of control over what the development teams are actually delivering because they no longer base their decisions on technology, but rather value delivery. None of these challenges mean that agile won't work at your company.

For an organization to successfully adopt agile, executive support is also critical. Agile is unlikely to succeed without it.

It's Enough for My Development Team to Be Agile

For agile to work properly, *all* teams have to buy in. So if your development team is gung ho, but your testing team is blasé, you won't get your best results. Your agile delivery process is only going to be as effective as your slowest group. To make agile succeed at its greatest potential, make each piece of the chain as efficient as possible.

Furthermore, the organization guiding agile teams must itself be agile, from the executives and business owners that make investment decisions and fund projects, to the program and product managers that drive IT delivery to ensure return on business investments, to the teams that do the work, everyone must be on the same page.

Agile Is a Silver Bullet

Agile isn't needed for every team in every situation. It isn't a cure-all. Agile is a superb solution for projects that are in development or undergoing radical changes. For other projects, such as those that are in maintenance mode, agile isn't as good a fit. If your project has a stable customer base and isn't undergoing a lot of change in the code, you may not need to use agile for that particular project. But for projects that are under new product or rapid development, agile really is the best way to go.

Agile software delivery teams enjoy greater levels of success

Confused by all the agile advice? Relax! With this friendly reference by your side, you'll learn the fundamentals of agile and how to increase the productivity of your software teams while enabling them to produce higher-quality solutions that better fulfill stakeholder needs more rapidly.

- ***Understand agile basics* — *discover how agile teams are organized and what makes agile different than traditional approaches***
- ***Explore your process options* — *learn about the different agile methods, including Scrum, XP, Lean, Kanban, Agile Modeling, Disciplined Agile Delivery, and the Scaled Agile Framework™ (SAFe)***
- ***Scale your agile practice* — *discover which scaling factors apply to you and learn to choose tooling and practices to make the most of your agile adoption***
- ***Learn about the Scaled Agile Framework™ (SAFe)* — *a process framework for scaling lean and agile across the enterprise***

Open the book and find:

- **The fundamentals of agile development**
- **Cloud choices for agile delivery**
- **An introduction to the Scaled Agile Framework™ (SAFe)**
- **The right tools to support your agile practices**
- **The role of DevOps extending agile across the software delivery life cycle**
- **IBM's agile transformation journey**
- **Ten myths about agile that might surprise you**

Go to Dummies.com®
for videos, step-by-step examples, how-to articles, or to shop!

FOR DUMMIES®
A Wiley Brand

Also available as an e-book

ISBN: 978-1-119-14976-7
Part #: RAM14031-USEN-00
Not for resale