Compliments of IBM

IBM Limited Edition

Agile Product Development

FOR DUMMIES®

A Wiley Brand

Learn:

- **Why agile product development is vital for building the "Things" in the Internet of Things**
- **Key strategies for implementing agile product development**
- **Top mistakes to avoid in adopting agile product development**

Jonathon Chard
Bruce Powel Douglass

Agile Product Development

FOR DUMMIES®
A Wiley Brand

IBM Limited Edition

By Jonathon Chard and Bruce Powel Douglass

FOR DUMMIES®
A Wiley Brand

Agile Product Development For Dummies®, IBM Limited Edition

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

ISBN: 978-1-119-17736-4 (pbk); ISBN: 978-1-119-17737-1 (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

Table of Contents

Introduction

The Internet of Things (IoT) is a phrase that is hardly out of the headlines these days. Suddenly, there's a headlong rush to make every product and system instrumented, intelligent, and interconnected to provide features and functions that were science-fiction only a few years ago. For consumers, this is great news. Not only do you get cool new products, but also the smart infrastructure that's being developed using IoT technology has the potential to improve lives. Smart transport systems, smart health systems, smart energy grids, even smart cities — the change is profound — and all around you.

But all of this change comes at a price for the companies developing the connected products on which the IoT depends. The complexity of development — and the expectations of today's consumers and markets — have never been higher. So companies have to find a way to deliver a whole new level of faster, better, cheaper — or risk getting left behind. And that's where agile product development comes in.

About This Book

Welcome to *Agile Product Development For Dummies,* IBM Limited Edition. You've probably been hearing a lot about agile software development for a good while — and now people are starting to talk about using agile beyond software for the whole product development process. You may well be wondering whether that's something you should be doing and, if so, how to get started. If so, then relax; this book is here to help.

The six chapters are organized to help you understand the problem that agile product development sets out to solve and how it solves it:

- Chapter 1 looks at the challenges facing product developers and the application of agile principles to tackling those challenges.

- Chapter 2 highlights how strategic reuse through product line engineering can work with agile processes to manage the complexity of creating and maintaining families of similar products.
- Chapter 3 discusses how systems engineering can adopt agile processes.
- Chapter 4 digs into the details of implementing agile product development processes, covering everything from requirements management, through modeling and simulation to testing and integration.
- Chapter 5 explains how to adopt and scale agile product development, including how to gain buy-in across your organization by ensuring everyone sees the benefits.
- Chapter 6 helps you to avoid adoption pitfalls by sorting some common agile product development myths from the realities.

Icons Used in This Book

As you read this book, you'll notice we've highlighted important information with some eye-catching icons.

Tips are key things that make your life easier as you adopt agile product development.

Remember icons are important ideas and approaches you want to remember after you've finished reading.

Watch out for Warning icons. These are pitfalls and things you want to avoid in your adoption of agile product development.

Chapter 1

Why Agile Product Development?

In This Chapter

- Understanding how increased product complexity and competition is changing product design
- Looking at the Internet of Things as a driver for change
- Introducing continuous engineering
- Defining agile
- Reinventing agile for product development
- Applying agile to product development
- Understanding the benefits of agile throughout the organization

In the last few years, the term *agile* has changed from a slightly geeky subject only mentioned between software developers to a topic that's discussed much more widely in businesses developing technology products. In this chapter, you discover some of the market forces driving that change and what agile can offer for product development.

A Brief History of Products

Take a look around you. Chances are you can probably see a product that has evolved dramatically from the version you knew a few years ago — if such a product even existed back then. A TV set has evolved from a device that was purely analog, to a device that contained some software to make it work better (auto-tuning, remote control, and so on) to today's smart TVs where the software is the differentiating feature.

Automobiles, washing machines, wrist watches — there's an endless list of products that have transformed from purely mechanical to the new world of smart products. And the phenomenon isn't restricted to consumer products — from smartcard-enabled public transport to smart electricity meters, smart traffic management systems to entire smart cities, everything these day seems to be made better with software.

But all this extra software and functionality means more complexity. You've probably seen statistics about how the software in a typical automobile has gone from a few thousands of lines of code to a few tens or even hundreds of millions in just a few years. The bottom line is there's just a lot more stuff to engineer — and to get right. And the difficulty doesn't increase linearly with the size of code, size of electronics, number of sensors, and so on. More complex products mean bigger development teams, perhaps outsourcing parts of development, integration with third-party technology, supply chain management, and so on. And as products do more, the potential to go wrong is greater. If those failures have significant safety or financial implications then they may, in time, lead to more regulation of the product, adding layers of compliance to the development activity.

Then there are market and customer expectations. Daily updates from the app store have everyone used to the idea that software-based products continually evolve and improve. And that sets your expectations for connected products. Because defects can often be fixed through online updates, you also have high expectations of quality — and you often turn to social media to vent your dissatisfaction if your expectations aren't met, with negative consequences for the product manufacturer's brand. All this means the pressure on product developers to keep ahead of their competition has never been greater, driving the race for ever faster design cycles.

The Rise of the Internet of Things

In looking beyond this trend of rising software complexity (intelligence) in products, you may notice two other key changes:

- **Products are becoming more instrumented.** Low-cost sensors, such as cameras, microphones, accelerometers, temperature and pressure transducers, and GPS location receivers, are allowing products to monitor and react to their environment.
- **Products are becoming more interconnected.** Through wired or wireless connections many products are no longer standalone devices, and the functionality and value they provide often depend on the wider system in which they operate.

These combined trends of intelligence, instrumentation, and interconnection now have a name: *The Internet of Things* (IoT). In some ways, the IoT isn't new; people have been connecting software-controlled products with sensors to each other and to networks for a while. But the appearance of the name "Internet of Things" signifies a general acceptance of the enormous value of this approach — and is fueling a rush to exploit its potential.

The IoT is the ability to connect all those "Things" to the cloud, where analytics can make sense of all the data they generate. This process enables a number of major benefits:

- **The data can yield new functionality and value.** Think of the real-time traffic congestion data your vehicle navigation system uses, derived from the GPS sensors in smartphones.
- **The data can provide operational performance feedback.** Think of the jet engine manufacturer that can use engine operational telemetry to inform predictive maintenance and allow the business to move from a pure product manufacturing model to a more profitable power-by-the-hour model.
- **The data can yield rapid, detailed, and complete insight into user experience and product performance.** This can inform the design of new and updated products.

In an IoT world, making the "Things" has moved from a linear specify-design-manufacture-sell cycle to a "closed loop" iterative activity in which the information driving product revisions comes not from painstaking market research and test lab results but in near real-time from the products themselves. And users increasingly expect the product they buy

to be painlessly upgraded throughout its life through new software delivered through the network. This can only intensify the increase in product complexity and the demands for faster delivery and higher quality.

Continuous Engineering for the Internet of Things

Many makers of "Things" have struggled for some time with old fashioned, siloed, waterfall ways of engineering products. Each added layer of complexity leads to more problems and unscheduled rework at the end when the components are integrated. And maintaining, let alone improving quality in the face of ever shortening development cycle times, is all but impossible.

IoT only makes the situation worse for makers and demands a fresh approach to product engineering that's designed for the closed loop of IoT product development. Continuous engineering is that fresh approach and brings with it a number of practices for more sustainable working in a closed loop development world:

- **Strategic reuse:** Intelligently reusing engineering artifacts wherever it is efficient to do so. Strategic reuse goes beyond clone-and-own approaches (with their attendant proliferation of data and maintenance issues). It encompasses the concept of product line engineering — defining product configurations and variation points — so that the common parts of a family of products can be engineered in one stream, and only variant parts are engineered separately.
- **Continuous verification:** Verifying artifacts as they're created instead of the traditional end-of-cycle approach and repeating verification at every change to ensure that quality is built-in throughout development.
- **Engineering insight:** Using integrated tooling, processes, and analytics to break down siloes of information and help engineers and project teams to make rapid, optimal design decisions that continuously drive the design in the right direction.

Continuous engineering isn't a process; it's a set of principles that can guide the application of the detailed processes for doing the design. You still need an approach for the detailed processes — and that's where agile comes in.

Defining Agile — the Agile Manifesto

The 2001 Agile Manifesto is the definition of agile for software development and proposes four basic tenets:

- Individuals and interactions are preferred over tools and processes.
- Working software is preferred over comprehensive documentation.
- Customer collaboration is preferred over contract negotiation.
- Responding to change is preferred over following a plan.

These tenets are further elaborated by 12 supporting agile principles:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity — the art of maximizing the amount of work not done — is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

The manifesto was written from the perspective of software development; however, agile is essentially an approach for maximizing success when doing complicated projects. That means agile thinking can be applied to non-software activities. In fact, software is mentioned only once in the four tenets and three times in the 12 principles. It's also important to note that neither the tenets nor the principles are a set of hard and fast rules. They are written in terms of priorities, values, and preferences. This leaves room to adapt how you implement agile for your needs while providing points of reference as to what's likely to improve the outcome. So while a working deliverable is valued more than complete documentation, if you need that documentation to comply, for example, with a mandatory standard, it's still an important part of the project. And while face-to-face communication may be highly valued, using effective collaboration tools across a globally distributed team is another way to follow the principle in a complex product development environment.

Reinventing Agile for Product Development

Software developers hit the cost-time-quality-complexity wall a few years back. Their response to this crisis, in many cases, was to move to agile software development. Certainly this happened for enterprise and IT software developers. And

more recently, as embedded software has become an ever more vital part of many products, embedded software developers have increasingly turned to agile.

The challenge for embedded software, however, is that it's part of a product, and that frequently comes with responsibilities. The environment in which it runs might be hardware-specific, have real-time operating constraints, limited processing and/or battery power, and so on. Those characteristics may change when the product hardware is changed. In short, embedded software engineers have to pay attention to other parts of product development. This can cause problems when embedded software development is agile and the rest of product development is waterfall — with its "big up-front design" and "test it at the end" philosophies at odds with agile flexibility and responsiveness.

But what if product development could become agile? With agile driving product development, you can start to deliver the development responsiveness that the IoT demands while managing complexity and improving quality and cycle times.

Applying Agile to Product Development

Software development was hitting its "cost-time-quality-complexity crunch" around the time (some would argue a good while before) the agile manifesto was written. Back then people didn't have smart products and the IoT. But now they do, and it's the turn of product development to look for new approaches.

Agile approaches have a number of general benefits that make them attractive not only to software developers but also to systems engineers and product development teams:

- By fostering ongoing collaboration with customers and openness to change, project outcomes can be more closely aligned to customer needs.
- Through incremental delivery of working parts of the system projects can achieve greater certainty of delivery.

- Through early detection of errors and defects and rapid customer feedback on delivered functionality, the late-stage risk to projects can be reduced, which in turn reduces the need to dedicate time and money to unscheduled rework.
- By eliminating unnecessary work, agile projects can achieve sustainable high productivity.

What's In It for Me?

The reason to use agile for product development is to gain some quantifiable benefits. So it's important to understand what those benefits are — and critically, which stakeholders in the business receive those benefits.

The good news is that agile product development has potential benefits for a very broad range of stakeholders:

- At a business level, agile can help meet commercial objectives by delivering higher quality products faster, at reduced cost.
- At an operational and project management level, agile can help improve project management by increasing the predictability of delivery and the responsiveness of a development organization to necessary change.
- At a practitioner level, agile can increase satisfaction by giving engineers greater autonomy and direct measures of progress in their work, and by reducing rework, which allows them to spend more time focusing on innovation.

It's important that agile delivers benefits across all levels of an organization as successfully implementing agile product development needs support from all those levels.

Chapter 2

Understanding Agile Product Line Engineering

In This Chapter

- Looking into competitive pressure
- Defining product line engineering
- Listing the core needs for product line engineering
- Understanding the techniques of variant management

Engineering knowledge is captured in intellectual property (IP) in various work products generated during product development, including requirements, architecture, trade analyses, test cases, safety analyses, security threat models, designs, implementation, and so on. That's a lot of IP, and it represents a huge investment of both money and time. As new products are built to address new markets or industry segments, all this IP should be reused efficiently and effectively. This is the subject of this chapter.

Increasing Competitive Pressure

Companies are driven by a number of forces:

- **Product complexity:** It's a well-known aphorism that products are getting more complex. In automobiles, the average lines of source code (a perhaps poor measure of complexity but something easy to measure) has gone from 2.4 million in 2005 to 10 million lines in 2010, while luxury automobiles may have as many as 100 million.

- **Need to decrease time to market:** Despite the increasing complexity and design challenge for modern products, market windows are getting shorter. This means that the design and develop cycles must also shrink.
- **Drive to reduce development costs:** As product market windows become shorter, development costs can have an increasing impact on corporate profitability. For this reason, companies are often looking to increase engineering efficiency.
- **New markets:** Internationalization has opened new markets for many companies while technology has created other opportunities, particularly as new smart technology replaces traditional infrastructure. Emphasis on being "green" for example, has created entirely new markets for power systems, and energy efficient motors.
- **Market fragmentation:** Formerly known as *niche markets,* larger markets are being fragmented by the special needs of smaller market shares and the clamor to have those needs met.
- **Increasing reliance on smart systems:** Many previously manual critical systems are being replaced with smarter systems. While this cuts costs and improves system performance, in the absence of human oversight, subtle defects in critical systems can result in catastrophic outcomes.

All of these forces mean that companies must be able to produce much more capable systems in less time. And the fact that people increasingly rely on these systems means that they need to get them right.

What is PLE?

Product Line Engineering (PLE) is a disciplined approach to constructing a family of products that have commonality in terms of their engineering data. This originated from an earlier discipline called Software Product Lines (SPL) and has been generalized to product development.

The goal of PLE is to produce product lines with a high degree of reuse of all appropriate engineering data. Many companies produce products that are highly similar, and it makes sense

to reuse the engineering data from previous systems to make new ones. The Software Engineering Institute (SEI) reports that SPL has resulted in "order of magnitude improvements in time-to-market, cost, productivity, quality, and other business drivers." (`https://www.sei.cmu.edu/productlines/`)

Although PLE can be used for opportunistic reuse, it's best applied to a planned product portfolio. Your product portfolio identifies the market segments you want to address and the products you will create to do it. PLE is then an effective way to actually implement your product lines via the strategic reuse of IP.

What isn't obvious is how to implement such a solution. Most engineering data is in stovepipe applications and not organized to facilitate reuse. It isn't clear what assumptions are made so it's hard to understand what can be reused and where. Furthermore, if you reuse engineering data, such as requirements, and a defect is identified in a product later, how do you identify which variants may be affected?

Core PLE Needs

Effective product line development requires you to manage large amounts of engineering information. To get started with any PLE implementation, you first need some underlying strategy, processes, and tools to aid in the creation and management of reusable IP assets.

Strategy

Various methods and workflows can be used to implement PLE, and they all have benefits and costs. However, because PLE is larger than a single project, it's imperative that different teams approach PLE using the same workflows and methods.

Interconnected engineering repositories

People generate a lot of engineering data of many kinds in a typical product development cycle, including

- Stakeholder and system requirements
- Software/electronic/mechanical requirements
- Systems architecture and trade studies
- Software/electronic/mechanical designs and implementations
- Safety/reliability/security analyses
- Test fixtures and test cases

The different engineering data sets are managed by specialized tools, but they must all be connected. You must be able to clearly identify a coherent set and version of the system requirements that are used to develop a specific version of the architecture. You need to be able to identify which test cases apply to which product variants. This requires integrating the repositories of engineering data to form working sets for specific products and product variants.

Traceability

A key facility to interconnect the engineering data sets is *traceability.* Traceability identifies the links between individual elements in those engineering data sets. For example, requirement R_1 might trace to architectural elements A_{15} and A_{22} and to test cases T_{155}, T_{156}, T_{158}, and T_{436}. Software design element SWD_{17} and electronics design element ED_{89} might collaborate to meet the safety analysis SA_{44}. Traceability is important in PLE because multiple variants of work products differ in subtle but critical ways. It provides the detailed relationships between data elements required to maintain multiple product variants simultaneously.

Great configuration management

World class configuration management is a core capability to support PLE. A number of different work streams (also known as *branches*) can divide or merge multiple times within a product portfolio. Management of such complexity isn't easy, but the benefits of doing it well pay off.

Techniques of Variant Management

Reuse is harder than it sounds because PLE involves reusing many different kinds of engineering artifacts. And, because you're creating variants, you're reusing some data and also creating new data. A number of techniques exist for implementing PLE with different levels of sophistication, offering different benefits.

Clone and own

In this approach, information is simply copied and modified to create the required variant. Although this is initially easy, it results in duplication of data and divergent systems. The complexity of managing changes and fixing defects across variants with isolated data sets makes this approach unsuitable for supporting an agile product development approach.

Multi-stream

In the multi-stream approach each component — both core and product variant-specific components — has a defined set of engineering data (requirements, design, and so on) and is managed in a stream to account for changes and fixes. New product variants are constructed by taking most or all of the core platform and adding in new components that define the product variant. This allows for more opportunistic reuse. Branching the streams of the components is the primary means for variant management.

Product parametrics

Parameters can be used to identify the variant to be constructed. An automotive PLE model, for example, might have parameters for the engine to be used (United States, Europe, Asia, rest of world), the drive train (economy, performance, 4x4), which trim level (basic, advanced, luxury), and the infotainment system (basic, enhanced, premium). This defines a large set of potential variants, not all of which must be produced; there might not be a need for a 4x4 car with an Asian

engine, basic infotainment, and luxury trim, for example. The parametric approach makes it easy to add new variants that fit within the pre-planned variability modes (as expressed by the parameters) but less easy to do true ad hoc reuse.

Feature management

In feature management, the reusable pieces are organized around product features. Each feature has its own set of integrated work products and engineering data and may also provide internal variants of the features based on parameters or, more likely, multiple streams.

Common to all approaches (except clone-and-own) is a high degree of modularity and the separation of the things that don't change (known as the *core platform*) from the things that change in the different variants. Each of these modular pieces is a *component* supported by a coherent set of engineering data. Figure 2-1 shows the relationship between the core platform, products, components, and their underlying data.

PLE is a key element of agile product development because the structured approach to modularity and reuse it promotes helps to support agile's dynamic refactoring, reprioritization, and responsiveness to change.

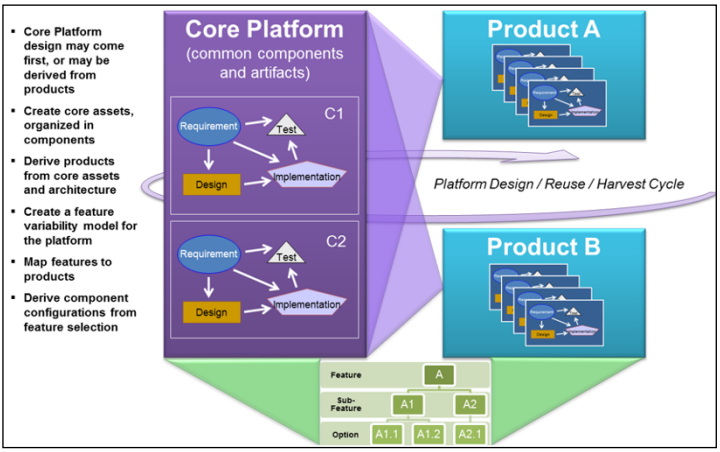

Figure 2-1: The PLE solution.

Chapter 3

Agile Systems Engineering

In This Chapter

- Looking at the challenges of systems engineering
- Understanding key agile practices for systems engineering

As products get smarter, new and more effective practices and technologies are needed to help you meet the challenges posed by their development. Systems engineering becomes even more fundamental in ensuring that your product concepts meet stakeholders' needs and product architectures can be effectively implemented. In this chapter, you see the key activities of systems engineering in product development and how agile practices can be applied to improve systems engineering.

Digging into the Challenges of Systems Engineering

Systems engineering focuses on capabilities and constraints at the systems level, above the more specific concerns of software, electronics, and mechanical implementation. The primary tasks for systems engineering include the challenges listed in this section.

System requirements specification

The System Requirements Specification (SRS) is a mostly textual document that defines the required behaviors and qualities of service of a system. Although the use of requirements management tools such as IBM Rational DOORS and RequisitePro

have improved the management of the requirements of complex systems, SRSs are often — if not usually — plagued by requirements that are incorrect, incomplete, inadequate, and inconsistent. This remains a key challenge for systems engineering that traditional approaches have improved on but failed to eradicate.

System functional analysis

System Functional Analysis (SFA) is the transformation of the system requirements specification into a coherent functional description of the system, which can form the basis of the functional and system architecture. It involves analyzing the functional needs of a system to both validate the requirements specification and to uncover requirements that are missing or problematic.

System dependability analysis

In today's smarter world, the various aspects of dependability — safety, reliability, and security — are becoming increasingly important. While these concerns fall within the umbrella of systems engineering, they're usually managed by specialists. These analyses are used to ensure that requirements adequately address the dependability needs of the system, architectural and design choices are compatible with these needs, missing requirements are identified, and the "safety case" has support when the product is submitted for approval by a regulatory agency.

Creation of a system architecture

The system architecture creation activity focuses on the identification of large-scale pieces ("subsystems") of the overall system, their responsibilities, their interfaces, and the allocation of requirements they must support. The system is usually not yet decomposed into engineering disciplines (for example, software, electronic, mechanical, pneumatic, and hydraulic).

Allocation of requirements to subsystems

In preparation for subsystem development, requirements must be allocated to subsystems prior to their being handed off to teams or subcontractors for development. These requirements must be consistent with the overall system requirements and often must result from the decomposition of system requirements into derived requirements.

Create hand-off specifications for downstream engineering

Traditionally, the output from systems engineering is a large set of textual documents that, even though manually reviewed, contain various errors that lead to significant rework late in the project. Part of the problem is that there's no way to automatically ensure that the textual statements are correct or even consistent with each other. The most common approach today is to hand off these hundreds or thousands of pages of textual specifications, errors and all, to the subsystem teams. The subsystem team must then cast this information into a format consumable by their tools and processes before proceeding. This is an open-loop approach that results in both low quality and high cost.

Defer implementation decisions as long as possible to give the maximum amount of flexibility for downstream design and implementation. With the advent of the Systems Modeling Language (SysML) and high-fidelity, model-based engineering (Hi-MBE) tools, the modern systems engineer can be more effective in accomplishing his tasks.

Agile Practices for Systems Engineering

The purpose of software engineering is very different from that of systems engineering. The former is to produce deliverable implementation while the latter is to develop specifications. The application of agile to systems engineering must take this into account.

A number of key practices can be effectively applied to systems engineering, including execution of specifications, test-driven development, model-based engineering, incremental development, and continuous integration.

Early verification of specifications

A key tenet of agile is that you ensure quality of a work product by verifying it as you create it (rather than some weeks or months after the fact). In the pursuit of quality, you need direct evidence of completeness, accuracy, and consistency of the system engineering work products. This means you need to "test" the specification. Using a more formal language, such as SysML (and appropriate tooling such as IBM Rational Rhapsody), to create executable specifications means that you can simulate the system behavior, allowing demonstration of its functionality under different circumstances. The more traditional alternative of reading hundreds of pages of text is a poor substitute.

SysML supports the precise specification of engineering data and is executable. The Rhapsody TestConductor Add On applies the methods of model-driven testing to systems engineering specifications.

Test-Driven Development

The practice of Test-Driven Development (TDD) is common in agile software development. It entails the development of test cases at, or slightly before, the development of some unit of software, and the immediate application of those tests. Unit level testing is performed incrementally, usually several times per day during 20- to 60-minute "nanocycles" to ensure that the evolving software baseline is defect-free. The same approach can be applied to the development of executable specifications. As some set of related requirements are added (resulting in the addition of a few states, transitions, and system actions), test cases can be constructed to ensure that the added capabilities are properly specified. Compared with a test-at-the-end approach, TDD significantly improves quality and reduces rework.

System specifications can be extremely complex. Building them up in tiny increments — with highly frequent testing — is a key practice to ensure that your system specifications are complete, consistent, accurate, and correct.

Model-Based Engineering

Model-Based Engineering (MBE) doesn't remove textual requirement specifications but augments it with high-fidelity models (discussed in Chapter 4) that allow you to reason about the specifications that you're creating. SysML was created for exactly this purpose. MBE allows you to explore requirements in ways that ambiguous textual specifications don't, and when you couple this with building executable specifications and TDD, you can construct superior specifications over using text alone. In addition, traceability between the modeled specifications and the textual requirements can be used to ensure coverage of all requirements and also assist in impact analysis when these requirements change.

Incremental development

Another key concept for agile systems engineering is the incremental development of the system work products, and this includes requirements, architectures, and dependability specifications. This incremental development occurs at two levels. First, you can incrementally construct your specifications by working on one coherent set of requirements at a time. The most common way to perform MBE with SysML is to group requirements into use cases (or user stories) and work out the functional (and non-functional) requirements as well as their interactions. Each use case then forms an increment of the system specification.

Second, at the smaller scale of incremental development, each use case specification may itself be complex and subtle. At this nanocycle level of systems engineering, the state-based behavior of the system use case can be developed using smaller increments. These nanocycle increments usually require 20 to 60 minutes and add some incremental piece of functionality that is then executed and tested for correctness. Many nanocycles are required to complete the specification for a use case.

Similarly, when architectural concepts are defined, the requirements can be allocated a use case at a time and the architecture validated by execution before adding the next. In this way, the subsystem interfaces (and the corresponding Interface Control Document, or ICD) can be incrementally constructed as well.

Continuous integration

If different teams are working on different use cases simultaneously, it is important that the requirements specified for one use case don't conflict with the requirements for another. This problem can be eliminated though a practice known as *continuous integration* in which the work from different engineering teams is brought together and tested to ensure consistency. This approach identifies conflicting requirements early at a much lower cost than in more traditional development life cycles.

The key to effective systems engineering is to verify the engineering data in the produced work products. This requires testable and simulatable models representing that data in a precise and verifiable way.

Chapter 4

Doing it Agile

In This Chapter

- Making effective agile plans
- Incorporating existing agile methods in your product development
- Managing continuous verification with high-fidelity models and simulation
- Connecting product development with the IoT Cloud

Agile methods have come about in the context of small teams building small non-critical software applications. So how do you actually *apply* agile approaches to the development of much larger scale systems that include electronics and mechanical aspects as well, and may be critical? This chapter addresses the "how."

Agile Planning and Management

Processes identify tasks and their properties, task execution sequences, time frames, work products to be produced, roles, and allocation of personnel to roles. Plans serve as a project roadmap so people can coordinate their activities and estimate project properties (such as completion date and development costs). Process definitions describe general flow, work products, and roles and must be instantiated for specific projects. A process definition may have a workflow for analyzing a use case, but you have to apply it to the use cases for *your* current project and all its special characteristics.

That means that process definitions include tasks that *may* be relevant in a particular case but not in your specific project. Additionally, the process definition may omit a task or work product that's relevant in your case. In any event, you must

tailor the application of a process definition to your particular project — and *that* is your project plan.

For example, your process definition may say that you need to create a software design document for a subsystem, but if that subsystem has no software in it, that task is useless. Alternatively, the process may not require you to produce a DO-178 compliant test coverage analysis, but your product requires FAA certification — so you must create one.

It is far more important to do what's relevant and important for your project than to blindly follow a generic prescription that may not completely apply in your case. You should prioritize tasks by the value they deliver and plan no task with costs greater than the value delivered.

Plans are good (just don't believe 'em!)

The reason why you plan is so you can determine when to start and stop other activities — such as manufacturing, marketing, or even other projects — and so you can estimate the cost and time associated with the product development. Plans allow you to make good business as well as technical decisions. However, product development plans are always an exercise in estimating things *you don't actually know.* It is common to spend considerable effort in upfront planning well beyond any reasonable expectation of true accuracy and then dogmatically require adherence to this plan.

Plans always have errors (some more than others). Some errors are due to estimates that prove to be inaccurate. Some errors are due to the fact that your assumptions weren't met (such as assuming a certain amount of work is done per unit time, known as *velocity*). Other errors are due to the inevitability that some of the things you know now will change in the future ("We didn't mention that it needs tail fins?"). So, never plan beyond the extent of your knowledge.

Plan to replan

If you pay attention during project execution, you can detect when project execution deviates from the plan. This might

be due to an overestimation of engineering velocity, the difficulty of getting sample parts, the unavailability of key subject matter experts . . . or anything else. However, once you detect a deviation from the plan, the plan should be updated to reflect what's actually going on. You can change the plan to account for new information, such as the measured velocity of engineering staff or a new schedule for sample parts delivery.

Plans are only useful if you actually follow them. If you're doing something other than what was planned, you have really no idea how late or early you are. Rather than merely change what you do, you should update the plan to reflect what you're doing. Plans should be updated at least several times per iteration.

To make sure you use plans effectively throughout your projects, keep a few things in mind:

- **Plans are always wrong.** No plan is completely accurate because there are things that you don't know that you are estimating and because things you do know change.
- **Instrument your projects with metrics.** Monitoring the project status and comparing it to the plan allows you to make better plans.
- **Adjust your plans frequently.** Assess the gap between the planned and measured project status and update your plans frequently to reflect "truth on the ground."
- **The point of planning is to guide and coordinate work, not to motivate.** Too many organizations use plans to instill urgency in their workers rather than to depict project progress as accurately as possible. You can't use the same plan for both purposes.
- **Plan in slack time.** If the worst thing you do in your professional career is come in under budget and ahead of plan, you'll just have to learn to live with the shame.

Adopting existing processes

There are a number of existing agile processes, including Scrum, XP, Crystal, and Harmony. All of these are generic in that they apply to different kinds of systems, and all of them are incomplete in that they don't take into account the peculiarities of your specific organization, project, or product.

For example, Scrum is currently the most widely used agile process, but the Scrum literature doesn't address the needs of safety critical systems or hardware-software integration. Harmony focuses on critical embedded systems but doesn't provide guidance on creating enterprise architectures. XP focuses on the technicalities of software development and doesn't provide guidance on project management.

Select a base process that closely aligns with your business culture, industry, and projects, and tailor it to your organization. List only the relevant aspects for your specific project.

Tool Support in Agile Product Planning

Many tools exist that can be applied to agile planning. IBM Rational Team Concert provides collaborative planning and management capabilities, including work item management and import of process tasks from the IBM process definition tool, IBM Rational Method Composer. Furthermore, Rational Team Concert integrates with other IBM and third-party tools, covering many aspects of the development life cycle, including requirements management, modeling and simulation, and quality management. This allows agile planning and workflow management to be linked to engineering processes to provide clear information for project managers and practitioners.

Agile Requirements Management and Traceability

Engineers adopting agile may be tempted to eliminate or heavily reduce requirements management; however, requirements are critical to successful agile delivery. A key function of requirements is to maintain a common understanding between the customer and the project team, which is vital for successfully planning and managing both the project and the product being developed. Agile emphasizes improving collaboration between stakeholders, which makes requirements management more rather than less important.

For all but the most trivial projects, requirements rapidly become hard to manage in text documents and spreadsheets. Consider using a requirements management tool that allows individual requirements to be managed and that can provide access to all necessary stakeholders.

Traceability is the glue that connects different layers of requirements and other engineering artifacts associated with a project into a coherent structure. It provides the mechanism to quickly and accurately assess the impact of new requirements or change requests, and ensure that the right artifacts are included when implementing a particular work item or change request. Agile emphasizes breaking large projects down into manageable pieces and also being responsive to change requests. This means that traceability is, if anything, more important for agile projects.

Traceability management is a key part of a requirements management tool. It's important, however, that creating and maintaining traceability isn't a difficult or time-consuming activity as practitioners will quickly see it as unnecessary work.

Choose a requirements management tool such as IBM Rational DOORS/DOORS Next Generation that integrates with other engineering tools and allows traceability to be built in real-time as engineering artifacts (requirements, models, tests, and so on) are created and maintained.

Agile High-Fidelity Modeling and Simulation

A key aspect of agile software development is the continuous engineering practice of continuous verification. This takes place primarily in two practices: Test Driven Development and Continuous Integration. The first of these develops and applies test cases as the software is developed to ensure that defects aren't "designed in." The second practice continuously integrates software from multiple developers to ensure that it all plays together; this avoids the typically huge integration expense and effort at the end of the traditionally run projects. In both cases, continuous verification is the key to avoiding rework, achieving timely product delivery and generating high software quality. These practices work well because software implementation is inherently executable and therefore testable.

In agile product development, you don't just produce software implementation. You produce many different kinds of engineering data, including system requirements, interface specifications, architectures, and safety, reliability, and security analyses. You must, therefore, address the question as to how to ensure the high quality of this engineering data, much of which isn't traditionally testable.

The answer is models and not simply models, but high-fidelity *testable* models that can be verified as to their correctness, completeness, consistency, and accuracy.

Modeling is the language of product architecture and design

A model is a semantic web of interconnected elements in which you omit details irrelevant to the purpose of the model and focus on details that are important to the reasoning you want to perform. You model a chair differently if you want to reason about its construction versus its different uses in various environments versus coordinated office décor. It's all the same chair, but you create models that allow you to capture and reason about details relevant to your needs.

Every (good) model has the following properties:

- **Purpose:** What reasoning is the model meant to support and what questions is it meant to answer?
- **Scope:** What are the rules for deciding what should be included in this model?
- **Precision:** What degree of detail is required of the element definitions to achieve the model's purpose?
- **Accuracy:** How accurate must the detail be to achieve the purpose of the model?
- **Views:** What views (such as diagrams) support the purpose of the model?
- **Stakeholders:** Who is to create the model, and who will view or analyze the information held within?

Models are not diagrams. Models are the data that may be represented in diagrams; diagrams are simply views of the model. High-fidelity models contain precise enough statements about the things being modeled that the information and conclusions are verifiable.

In agile product development, models are built to

- Capture and understand requirements
- Represent system architecture
- Depict interfaces between large scale pieces of the system
- Show allocations of requirements to system elements
- Depict interfaces between different engineering disciplines
- Define the interactive or singular behavior of system elements

For example, Figure 4-1 shows a state machine that represents the requirements for a use case. Each state, event, and action represents one or more requirements. This provides you with a precise language in which you can state what you want the system to do for this use case. In this usage, we've taken ambiguous textual statements and constructed a formal statement about how the system responds to user input actions for setting different parameter values that control the delivery of medical ventilation.

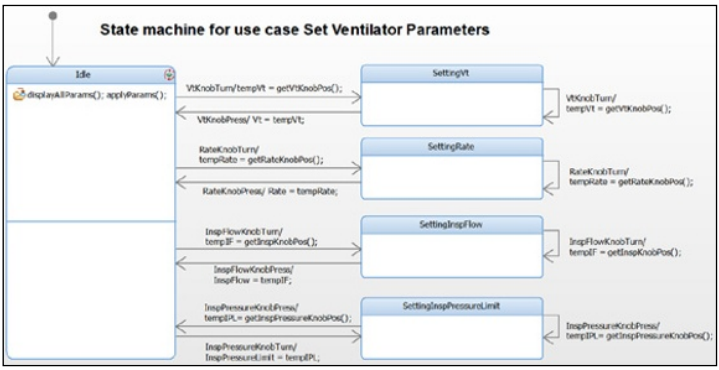

Figure 4-1: A use case requirements model.

Agile model construction

A key agile practice is to construct complex things in small steps, verifying continuously the correctness of what you're creating. Modeling isn't different. You construct potentially complex models, a little bit at a time, verifying the correctness of the model before you add more detail. In this way, you avoid defects and improve the quality of the information being modeled.

Figure 4-2 shows the workflow from the Harmony Agile Model-Based Systems Engineering process for the analysis of system requirements. In it you see that you define some portion of the functional flow (normally on SysML Activity Diagrams), then derive specific scenarios of interest (on sequence diagrams), then construct/update a simulation environment, define the state machine, and verify the model. Because this is done a bit at time — usually 2 to 5 requirements — the cycle is very rapid, and the model can be simulated quickly and easily. Once the model is correct, you can add the next couple of requirements and repeat. During these nanocycles — typically 20 to 60 minutes in duration — it is likely that missing or incomplete requirements are identified and repaired, greatly improving the quality of the requirements.

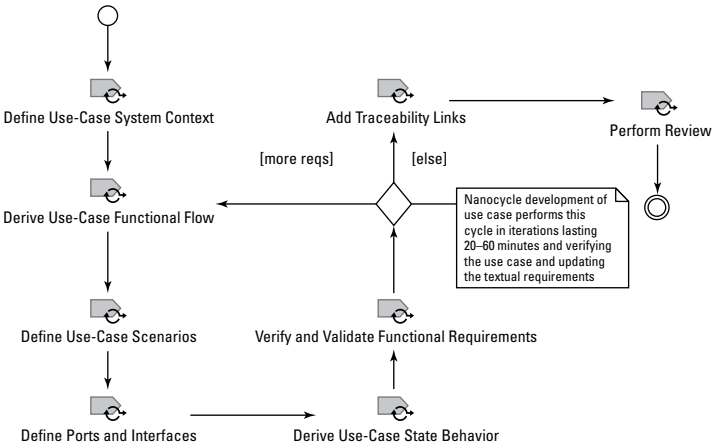

Figure 4-2: The requirements analysis workflow.

Simulation enables continuous verification

A key feature of high-fidelity models is that they can be executed (simulated). You can then test to ensure the model is properly specifying what you think it's specifying — something not possible with natural language.

A *test case* is a specification of input events with specific values for data and flows that occur with a specific sequence and timing and have a measurable defined output or outcome. A model that can be simulated can be verified by developing test cases that assert qualities of the data represented by the model.

Almost all engineering data can be specified in languages more formal than natural language, such as SysML or UML. Such specification models can then be verified, using the standard agile practices of Test-Driven Development and Continuous Integration to ensure their quality. This is crucial for effective agile product development.

The best way to know your engineering data is correct is to verify it as you construct the model of it. This requires precise executable models, such as those that can be created with SysML or UML.

Agile Quality Management and Test

Verification in product development comes in two forms. *Syntactic verification* ensures that the engineering data is well-formed; it complies with standards governing its organization and scope. *Semantic verification* is what you normally think of as verification — the data makes sense, is correct, and is sufficiently accurate and complete. The techniques for semantic verification include review, testing, and formal methods.

For engineering data beyond software implementation, review is by far the most common — and weakest — way of performing semantic verification. Testing requires executability, and this means the models must be more formal.

Continuous verification in product development

Test early — test often. One of the keys to effective agile product development is continuous verification of both the product and other engineering data. Requirements, architectures, interfaces, safety analyses — all these sets of engineering data are valuable to the extent that they're complete and correct. And that means that you — as engineers — must verify the engineering data is correct.

If you want to use testing as your primary verification approach, this means that you must produce models that are precise enough to support testing. Almost all engineering data can be modeled to support verification through simulation.

Simulation and test

The UML Testing Profile defines the means by which models can be specified by using UML (it also works with SysML) and how tests can be run. The IBM Rhapsody tool supports this standard with the Rhapsody TestConductor Add On. You can specify test cases with sequence, activity, or state diagrams to be applied to models. If you're using a tool such as Rhapsody, that means that you can construct high-fidelity models, debug them using simulation, and formally test them with TestConductor. This process assures that the data in the models is right, and subsequent engineering tasks based on that data have a high-quality foundation on which to begin.

You can test requirement sets by building executable models and then running them to evaluate what you have specified in various scenarios of use. The result is that you hand off much higher quality requirements to the designers. You can test architecture to ensure that all the requirements have been properly and consistently allocated in the system structure, and that the interfaces support all the necessary functionality. You can test designs to ensure they comply with their requirements and achieve the necessary functionality and performance.

It's even possible to perform multi-modal co-simulation, integrating a variety of different models types into a single environment for verification. This is possible with the Functional Mockup Interface (FMI) specification. Rhapsody supports FMI,

so it's possible to build a single simulation that integrates UML for the software, SysML for the systems model, Simulink for the control model, and SimulationX for the mechanical physics model.

Test management with changing requirements

Test cases ensure the system behaves appropriately in various circumstances, environments and with various inputs, timing, and sequences. In addition to test cases, supporting structures — called *test fixtures* — must be constructed to support testing. These may be simulations of external systems not yet available, platforms that assist in test execution, or systems that provide insight into system operation and outcomes during test case execution. In principle, test cases should be developed in conjunction with the requirements, and in fact, can even be thought of as an expression of the requirements they verify. As such, it's crucial that as requirements change, the test cases change along with them.

With traceability among the requirements, test cases, and implementation (software, electronics, mechanics, and integrated systems), if any of these items change, you can perform impact analysis and updates to ensure the consistency of the entire engineering data. Sometimes the implementation may change, and this can propagate change to the requirements and test cases as well.

Connecting Product Development to the IoT Cloud

In the agile product development, you focus on the "T" in the Internet of Things (IoT). Almost as important is the "I" — the interconnection of these devices. This is most commonly done with cloud services to integrate the data from many (possibly very many) devices. As devices register with your cloud, you gain unprecedented abilities to understand, analyze, control, and modify the behavior of all those devices within your purview. Consumers gain new, network-derived services that weren't previously possible, easier product

updates and more automation of product service and maintenance. Vendors gain the ability to identify detailed usage patterns and trends that can inform product updates and new versions, deliver new value-added services, and spot systemic problems and address them before they manifest. Continuous engineering aligns product development to the needs and opportunities of a connected world. It relies on agile's responsiveness to change to enable product development to make use of insights delivered from products in use.

Throughout all this, security is absolutely critical to the success of modern interconnected products. Security can be defined as the absence of intrusion, interference, or theft and is a serious cyberphysical concern. This means that not only must you pay attention to secure coding practices but also you must apply holistic security thinking to product development. This means that much like safety analysis in safety critical systems, security must be ever in your mind throughout the product development process. You must think about security requirements, security analysis, secure architectures, security design patterns, secure implementation, and security testing.

Chapter 5

Scaling Agile Product Development

In This Chapter

- Introducing the Scaled Agile Framework (SAFe)
- Recognizing the value and challenges of tooling when scaling agile
- Discovering the benefits of Open Services for Lifecycle Integration
- Aligning the enterprise-to-agile culture
- Getting buy-in across the organization
- Getting started with a pilot project
- Using an agile process to adopt agile

Historically, agile methods have been applied to small co-located teams working on small projects. Certainly, almost all the agile development literature supports that statement. Organizations building far larger systems — such as medical equipment, automobiles and even airplanes — want to reap the benefits of agile product development as well. As a result, there's a growing body of thought — along with overarching processes, guidance, and training — that deals with applied agile in the large. With one notable exception, almost all of this work is still solely focused on agile software development, including the Scaled Agile Framework (SAFe). However, recently the Scaled Agile Framework for Lean Systems Engineering (SAFe LSE) has appeared and addresses some of the issues surrounding larger scale development of products implemented with multiple engineering disciplines. In addition, IBM's continuous engineering initiative addresses the larger scope as well. These two approaches are quite compatible and integrate well with the other techniques discussed in this book.

Scaled Agile Framework (SAFe)

The scope of the vast majority of agile literature is in the daily running of software development projects. Scrum is the most common such approach, but a number of alternatives exist as well. This is just one of the parts of running the enterprise dealt with by SAFe.

SAFe goes beyond the daily running of software development projects by integrating practices together in a larger business scope — the scope of the enterprise. Beyond the straightforward application of Scrum to daily software development, SAFe adds support for agile teams, agile architecture, sprint goals, spikes, and refactoring.

SAFe also introduces the topics of project management, system teams, release management, product vision, program epics, and product roadmaps. The results are what is referred to as the *Agile Release Train* — the series of incremental product releases, the roles necessary to pull that off, and the guiding principles.

Beyond that, SAFe brings product portfolio vision to bear, at the upper levels of the enterprise. This includes applying lean methods (which have to do with minimizing waste), Kanban (a scheduling technique applied to lean systems for just-in-time production), enterprise-width strategic themes, and epics.

In short, SAFe is a framework in which compatible agile and lean methods are combined to address the issues of developing many products simultaneously within the context of an enterprise. While SAFe is fundamentally concerned with software development, the Scaled Agile Framework for Lean Systems Engineering (SAFe LSE) broadens the scope of SAFe to include systems engineering and multi-discipline product development, adding adaptive requirements and design, MBSE, and set-based design.

Continuous engineering and SAFe aren't incompatible. Continuous engineering focuses on the detailed practices of how to accomplish engineering workflows, whereas SAFe focuses on the big (enterprise) level exclusively. So you don't have to make a choice between continuous engineering and SAFe.

Tooling for the Large Scale

Any large scale adoption of agile product development practices is going to require tooling — tools that enable people in different geographical locations to work together, that allow the engineering information associated with the different development activities to be managed and maintained, and that allow for efficient reporting and insight into project status. Agile techniques emphasize collaboration between all stakeholders, including between customers and project teams, and the different engineering disciplines and practitioners responsible for the different activities of development. So it's vital to support collaboration with an effective agile tooling solution.

The need for collaboration between different geographical locations is a familiar problem and can typically be met with web-based and cloud-based tools. However, the need for collaboration across different engineering activities presents a different challenge. Here, integration between different tools is required, allowing information to be shared and linked. In fact, integration is a vital underpinning of traceability, which is key to successful agile product development (check out Chapter 4 for more info). Traditionally, integrations between tools, such as requirements management, modeling and simulation, and planning and workflow management, have been through dedicated point-to-point links. However, this kind of integration strategy is far from ideal. Typically integrations are specified for particular versions of the tools they're linking. If both tools are from the same vendor this may be okay, but integrations can often be a maintenance nightmare if any of the tools are patched or upgraded. Also point-to-point integrations link just two tools so they don't tackle the problem of providing truly cross-project access to information without adding more layers of complexity.

OSLC and Enterprise-Wide Agility

Open Services for Lifecycle Integration (OSLC) is a set of non-proprietary specifications for integrating development tools intended to make life easier for tool users. Integration

functionality is based on publicly available specifications, offering a number of benefits:

- Tools from different vendors can be integrated as long as they each support the relevant OSLC specification.
- Integrations are at much less risk of breaking with tool version upgrades or patches.
- Multi-tool integrations can yield more functionality than point-to-point integrations.

The last bullet is especially interesting from the agile product development perspective. By integrating tools across different engineering activities — such as agile work item management, requirements, modeling and simulation, quality management, and change and configuration management — you can gain new insight on engineering information.

Tools such as IBM Rational Engineering Lifecycle Manager can help engineers quickly visualize the traceability between all the engineering artifacts associated with an engineering change or backlog item, dramatically improving the responsiveness of the agile process. What's more, tools such as IBM Rational Publishing Engine can pull information from multiple tools across the life cycle, for example, to automatically generate documents to support compliance for critical standards-based projects.

Choose a vendor with strong support for OSLC across a range of tools because it's easier to extend your tool support to new functionalities from the same or other vendors as your development needs evolve.

Aligning the Enterprise to Agile Culture

Agile product development isn't just a technical process that affects a few engineers — it's a fundamental change to the way the business approaches product development that needs understanding and backing from across the business. For this reason, you need to involve stakeholders throughout the business, including engineers, operational managers, and business management.

Getting Buy-In Across the Organization

The key to getting the backing throughout your organization is to make sure the benefits are understood at all levels of the business. (Check out Chapter 1 for the benefits for the different perspectives in the organization.)

Some organizations arrive at agile product development as a last-ditch initiative when their traditional processes are overwhelmed by complexity and change. But it's better to start debating the benefits throughout the organization before that situation occurs so everyone can become a stakeholder in the success of your organization's agile product development initiative.

Starting Small

Many agile software initiatives started with skunkworks projects — projects that were run under the radar of management — that could provide confidence in broader agile adoption by providing a proof-point of success. These projects were typically small, co-located teams where the work could be completed in a single room, which is a much simpler context than today's product development. For agile product development, even at the start, you need to work across different engineering disciplines, which typically means across departments and management structures.

A pilot project is still a great idea, but make sure the objectives, deliverables — and critical benefits — are understood by all parties from the start.

Leveraging Support

You may already have agile embedded software development as part of your product development. This group can be strong advocates for agile adoption and can help with coaching and mentoring the new groups adopting agile. They benefit from the teams around them having an agile perspective

and engaging their support could have the added benefit of helping collaboration between software and other engineering disciplines.

Using an Agile Process to Adopt Agile

Making it work in a pilot is one thing; making it work across the business is another. Although adopting agile is hard, agile suggests an iterative and incremental approach for doing this:

- **Break the problem into pieces.** Identify the areas where you want to apply agile approaches to gain benefit.
- **Prioritize the pieces to identify what to do first.** You may use a number of factors including anticipated benefit, cost, and criticality of the activity.
- **Define the success metrics.** They can include time, productivity, quality — whatever is important to your organization.
- **Apply the agile process and measure results early and often.** If you hit problems you can implement changes quickly until you find what works.
- **When it's working, pick the next prioritized piece and repeat.** Remember — it's okay to reprioritize the pieces as you learn more about your agile implementation.

An advantage of this approach is that you don't move on until you have some demonstrable success. So it's less likely you will build a debt of scepticism that things are heading for disaster. And once your teams have experienced an agile environment where they can be more productive and have more control over their workflow, they're likely to become strong agile advocates for expanding the agile deployment.

Chapter 6

Ten Myths about Agile Product Development

In This Chapter

- Spotting agile product development misconceptions
- Tackling the naysayers

Agile product development offers the prize of major improvement in the way complex products are developed. But implementing agile product development is itself a challenge, and you should certainly consider the pitfalls. It's important, however, to sort the fact from the fiction that may appear to be blocking your path. In this chapter, we debunk some common myths about agile product development.

It's a Fad

Agile may be new to non-software disciplines, but it comes with a proven decade-and-a-half record in the software domain where its use is still growing. The cost-time-quality-complexity crunch that agile addresses certainly isn't going to go away any time soon. In fact, as more products get ever-smarter, agile is likely to become more necessary than ever.

It Only Works for Simple Products

While agile can certainly be used for simple products, its real benefits show when things are scaled up in complexity — with

more interdependencies, features to prioritize, and changes to manage. In these environments, agile can help ensure that important work is prioritized, there are fewer nasty surprises at the end of development, and critically, what's delivered actually meets the customers' needs.

It Won't Work for Critical Products

Some have taken agile's "eliminate unnecessary work" principle to mean "don't do documentation" — giving the impression that agile isn't suitable for serious development. In fact, for high-criticality products (which could mean safety, financially, or availability critical), agile provides a way of delivering what's needed while minimising the overheads associated with standards compliance.

It's Unproven

Agile has been in use and growing in popularity for software development for at least a decade and a half. In recent years, there has been intense interest in both agile systems engineering and agile product development, and there are plenty of companies across many industries that have successfully applied agile in these areas.

It's Just a Technical Delivery Process

The detailed work of agile product delivery processes may be enacted by engineering practitioners, but because the process requires collaboration across the organization, agile adoption must be supported by both project and senior management. Check out Chapter 1 for the benefits for all these stakeholders.

It Can't Work for Non-Software Engineering Teams

Agile was originally created for software development because it faced the cost-time-quality-complexity crunch before the rest of product development. But agile's tenets and principles aren't restricted to software development as long as the processes are applied intelligently to the needs of product development. The key step is for stakeholders to understand the benefits and to buy-in to adoption.

Quality Will Drop

Agile product development is focused on delivering working outputs. So what's delivered will be of high quality. Because agile prioritizes and delivers the maximum output within the constraints of time and resources, an agile product development approach delivers more high-quality output than a non-agile (but over-optimistic) project.

The Business Won't Know When Products Will Be Delivered

Development is inherently uncertain and many non-agile projects end up missing their delivery dates due to late-discovered problems when everything is integrated at the end. Agile is adaptive in both how much it delivers and how long it takes to deliver and constantly improve its prediction of how much can be delivered and when. Because agile is focused on optimizing productivity, the answer to "how long?" is likely to be "sooner than a non-agile project."

The Business Won't Know How Much Development Will Cost

Agile is adaptive and optimizing, so it's likely to be able to save cost over a non-agile program. It's also less likely there

will be cost-overruns at the end of the project because each incremental piece is proven to be correct instead of waiting to the end of the project to see if the final product works.

We Don't Need to Change

For many companies, the old ways of doing things have been under strain for some time. But implementing agile product development isn't trivial, so we wouldn't suggest that you consider it if it wasn't essential. Product development has now hit its cost-time-quality-complexity crunch and for many companies there's no alternative but to change. For those that don't, they risk losing the competitive edge.

Put agile product development to work to create more successful products

As the Internet of Things becomes a reality, are your product development processes struggling to keep up with demands to deliver connected products ever faster, better, and cheaper? If so, this book is here to help with clear advice on what agile product development can deliver for your engineering teams and for your business.

- ***Explore agile product line engineering — discover practical techniques for handling product families and variants***
- ***Understand agile systems engineering — use agile techniques to create and validate product and system architectures***
- ***Implement agile requirements management — improve responsiveness and customer collaboration to deliver the right products faster***
- ***Adopt agile modeling and simulation — create proven designs faster to cut rework and achieve better engineering outcomes***
- ***Scale agile product development adoption — get buy-in, support, and collaboration across your organization***

Open the book and find:

- **Key agile product development engineering practices**
- **The tools you need to support agile product development**
- **The role of the Scaled Agile Framework® (SAFe®) in agile product development**
- **Agile product development myths and common pitfalls to avoid**

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: 978-1-119-17736-4
Part #: RAM14033-USEN-00
Not for resale

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley’s ebook EULA.