A Guide to Creating Winning Products with Agile Development Teams

AGILE EXCELLENCE

FOR PRODUCT MANAGERS

280 GROUP PRESS

Greg Cohen

Foreword by Brian Lawley

“Agile Excellence™ for Product Managers” Book Excerpt

A Guide to Creating Winning Products with Agile Development Teams

By Greg Cohen

superstar press

E-mail: info@superstarpress.com
20660 Stevens Creek Blvd., Suite 210
Cupertino, CA 95014

BOOK EXCERPT Table of Contents

Contents

NOTE: This is the Table of Contents (TOC) from the book for your reference. The eBook TOC (below) differs in page count from the tradebook TOC.

Preface

This book is written for product managers who are making the transition to working with Agile development teams, as well as for Product Owners and project managers looking for better ways to organize and lead in their companies. I have tried to describe Agile through the lens of product management, including how Agile helps deliver winning products to market. Although what follows does not require an advanced understanding of product management, it is assumed the reader has basic product management skills. Also, because this book is intended for product managers, I have kept it as brief—and as practical—as possible. I know you are busy, and stretched by many competing objectives. My goal is that you can read this book in a single sitting on a three-hour flight to visit a customer.

When I was first introduced to Agile in 2001, it was a liberating experience. It freed me to be a better product manager. I was able to manage risk and adapt to emerging requirements at an entirely new level of effectiveness and do so in a way that was not disruptive or demoralizing to the development team. It was one the happiest moments in my career. Although at that stage I did not understand the theories behind it, I realized that Agile was a better way to develop software for the customer, the company, and the employees. Later, as I learned more about the theoretical underpinnings, it became increasingly clear why traditional, serial, software development just did not work that well.

In the pages that follow, I hope to impart some of the initial excitement I felt when I first started using Agile, and share with you its fundamentals from a product management perspective so that you can support your team, be in control of your product, and realize immediate benefits. I also hope that, by knowing what to expect from Agile, you will be able to identify if your team is underperforming and facilitate change to unblock them.

This book takes the reader on a journey that starts 10,000 feet in the air, with an overview of how an Agile development team works, and then makes a circling descent to reveal increasing detail of how product managers work within an Agile development environment. We finally land in Chapter 6, with a step-by-step guide to getting started and succeeding with Agile. Important concepts are touched upon two to three times from different angles to allow the reader to acquire a deeper appreciation of how they fit in with the whole.

The book is divided into nine chapters. In Chapter 1, we will define Agile development, understand its principles, look at why it works, and, most importantly, explain why it is good for you as a product manager. In Chapter 2, we will look at Scrum, a popular Agile method, in some depth. In Chapters 3, 4, and 5 we will look at the mechanics of Agile from the product manager's perspective and use Scrum as an example. We will cover how you manage a release, how you plan for a release, and how the documentation that you provide to your team changes. In Chapter 6, we will go through a step-by-step view of everything you need to put in place to get started and be successful with Agile, including how you interface with marketing and sales.

Chapter 7 covers how teams generally organize around Agile and identifies obstacles to team performance. Chapter 8 explores XP (Extreme Programming) and Lean Software Development methods to provide a broader understanding of Agile in practice. Regardless of the methodology your team selects, it is important to adapt it to support your products and customers while also working within your company's culture and your team's skill set. This chapter is intended to help you understand different ways to implement Agile. Chapter 9 concludes with a discussion of how Agile can assist with company-wide process improvement; the methods best suited for different types of projects; and a review of the key points of the earlier chapters.

It would be easy for a reader to think this book advocates Scrum over other Agile methods because of the time devoted to it. This is not the case, and the book tries to remain agnostic while providing a solid foundation for any Agile method with which you might work. Of the three methods described in this book (Scrum, XP, and Lean), I had to select one to carry through from start to finish. Scrum is best suited for this purpose because of its rigid definition around process, and its openness around development methodologies. Scrum also has a well-defined role for the product manager, also known as the Product Owner, which covers responsibilities and methods of interfacing with the development team. After completing this book, you should be able to pick up and work with any Agile methodology.

I hope that you will enjoy this book and come to experience the same satisfaction I have had working with Agile teams. Furthermore, I hope that through your hard work and the help of Agile, you will deliver greater value to your customers, more quickly than ever before. Lastly, I hope that you will view this book as one of the waypoints on your journey of continuous improvement. Our development teams, our companies, and we as product managers, must never give up the perfectionist pursuit of building better software to satisfy our customers.

Chapter

1 Why Agile Is Good for Product Management

Agile is often introduced to an organization through a development team seeking to deliver better software, faster. The change, however, sometimes causes anxiety to product managers. If this describes you, your anxiety is unfounded. Your development team wants to improve how they deliver software, and it would be foolish to do anything less than encourage them and jump right in to help. Agile is one of the best developments to ever happen to product management and our ability to deliver products that customers love.

Traditional Software Development

Before we look at Agile, let's initially take a look at the traditional software product development process (Figure 1.1). First, a company must consider all products in which it might invest, a process otherwise known as ideation. Next the company screens each idea for fit and potential return on investment. An idea that passes the screening is further defined, designed, coded, and tested. This phase is often known as "waterfall development" because the team moves through the four stages of define, design, code, and test in a serial fashion, completing one stage before starting the next. When that process is complete, the product is typically deployed to a limited customer set for beta testing, which confirms that the product works as designed and without unintended consequences. Usually issues are uncovered in beta testing, but only the most severe get addressed, because each change adds the risk of introducing new bugs to the system and delays the delivery of the product. Once out of beta, the product is made available for production use. At each step in this process, there are decisions or stage gates, where the company evaluates whether the product is progressing as planned and whether the investment still makes sense based on the latest information. Depending on the company, these gates can be formal steps with written approvals, or less formal check-ins.

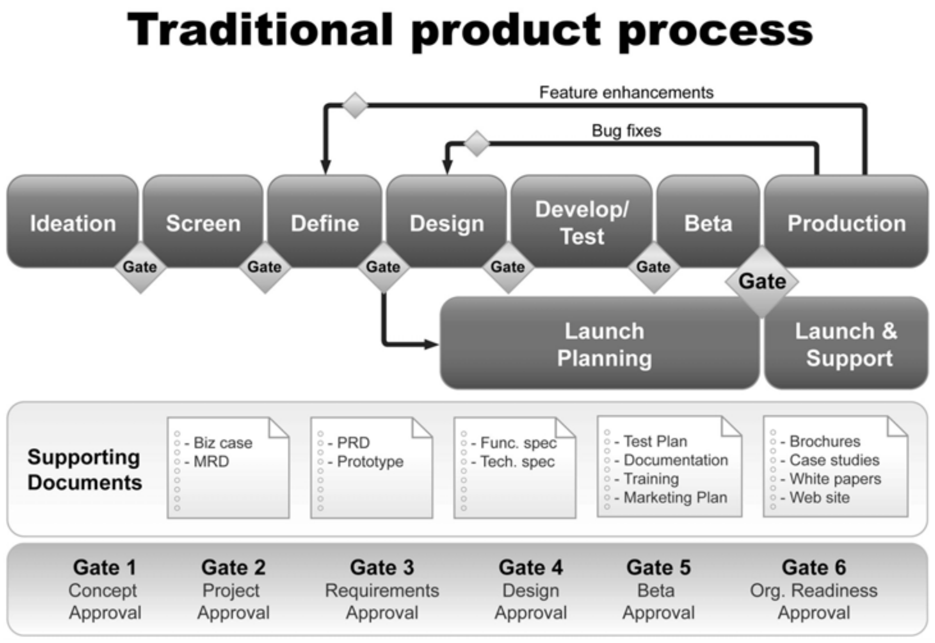

Figure 1.1: *The traditional software product development process*

Once the product clears the definition stage, it has a high probability of making it to production. Enough due diligence has occurred for the company to feel confident that it understands the market problem, market opportunity, what the product needs to do, and the investment needed to take the product to market. Thus, as the product enters the design stage, launch planning occurs in parallel to development to establish the product's positioning, messaging, and internal and external communications plans.

At the different stages, documents are created to support each process. These include a business case analysis, market requirements document, product requirements document, test plan, and collateral. The documents are used to disseminate market and product information within the team and company, capture decisions, and improve decision-making.

The product manager plays a central role in moving products through all these stages. She works closely with the different departments to make sure the product is delivered on time and to specification. She also ensures the other departments are trained and ready to support the product. Ultimately, the product manager is responsible for the commercial success of the product.

The Cost of Change

I was taught it was fairly inexpensive to make changes to a project early in the traditional development process—for example in the definition or design phases—but very expensive once development proper began, and exponentially more expensive as the product entered testing and production (Figure 1.2). This advice was reinforced by actual experience.

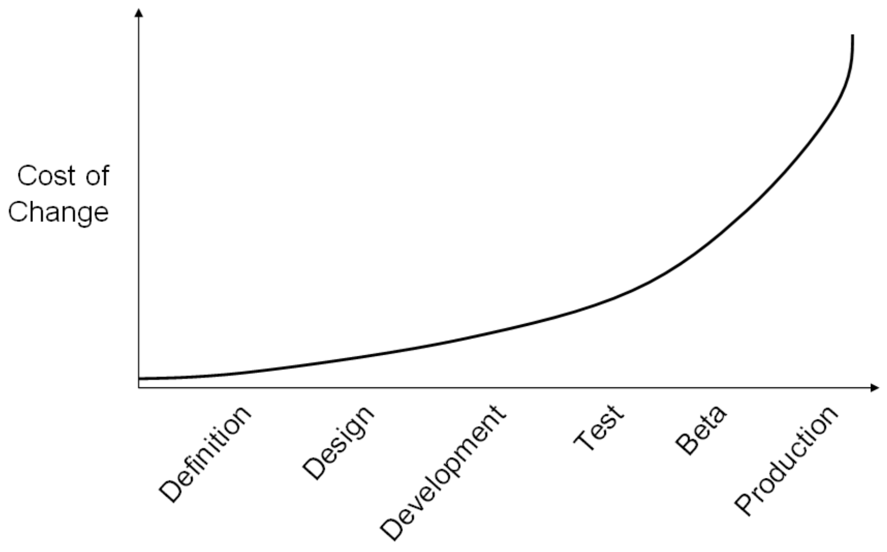

Figure 1.2: *The cost of change for a traditional software project*

When I introduced changes to a product during the development stage because of new information or a new customer requirement, a domino effect of negative consequences followed. First, the process had to go through formal change control that was intended to ensure everyone on the team, especially those handling development and QA, understood the change and that the plan of record was kept up-to-date. This meant documentation and meetings. Second, the effort had to be estimated and, often, the schedule recast.

Assessing the effort needed to develop a complex feature was hard; gauging that feature's impact on the schedule was near impossible. It also became clear that these changes caused the development team a significant amount of redesign and rework that, under the pressure of a tight schedule, was demoralizing.

Because change was so expensive in the later stages of the project, I invested a lot of time and energy in gathering requirements and getting the definition right. I also speculated about the future needs of the system. That way, even if the functionality would not be delivered in the early releases of the product, the foundation would be there to support it. Additionally, in the back of my mind, I could never let go of the idea that the second release would not be for six to nine months after the first one. If I thought a customer would need a feature in that time frame, I felt obliged to add it to the release even if there was no conclusive evidence of a need for it.

The implications of this exponential cost curve are severe. First, it is challenging to get everything right up front. Users do not always know what they want, nor is it their job to know. Even with study and prototypes, it is nearly impossible to ferret out all possible use situations until the product is deployed in a live environment and used in the context of a real customer's day. Further, as I highlighted in the previous paragraph, attempting to do so promotes bad decision-making and poor habits on the part of the product manager, including speculation (about what the product will need to do) and hedging (by adding extra, partially researched features). Lastly, even if the product manager could define the product perfectly to start with, markets are dynamic—over the six to eighteen months it takes to finish a release, the target will move.

Kent Beck, who developed Extreme Programming, asks a very thought-provoking question: if our cost curve were relatively flat (Figure 1.3) would we behave differently? Would we delay decisions until we had better information? Would we over-design in anticipation of future needs or just design and build to our current requirement? Would we be able to deliver more value to the market, and would we be able to deliver better software faster? Agile attempts to change the cost of change equation.[1]

1. Kent Beck, *Extreme Programming Explained: Embrace Change* (Indianapolis: Addison-Wesley, 2000), 21–24.

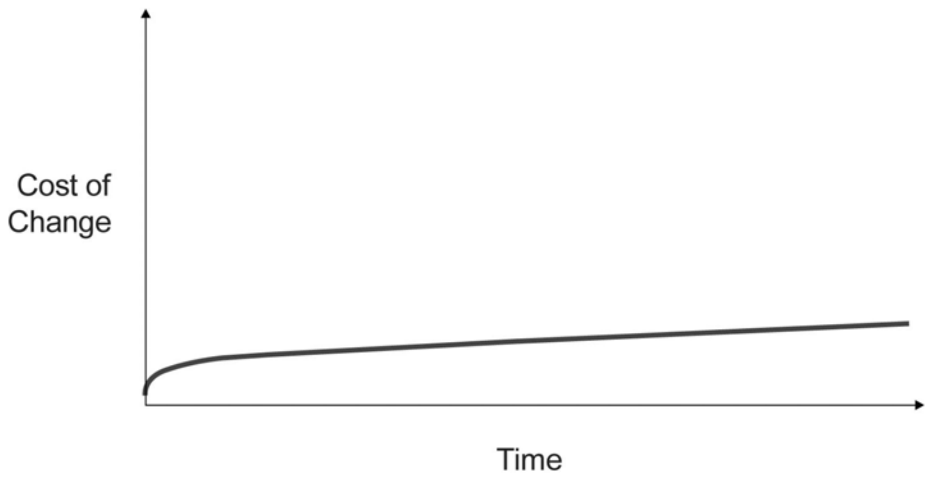

Figure 1.3: *The cost of change during a project with a flat cost curve*

Agile Software Development

Agile development is an alternate way to build products. Instead of following a linear path to define, design, develop, and test the software, Agile delivers functionality through more frequent and smaller iterations (Figure 1.4). Large requirements are developed over multiple iterations and developed in small "vertical" slices of functionality. Each slice is limited in its capabilities, but works end to end, including the graphical user interface (GUI), application logic, and the database. Each slice must deliver business value. This differs from the traditional approach, which might look to develop in "horizontal" slices by designing and developing the data, logic, and interface tiers up front and integrating them late in the project, or by developing the entire data tier first and then building the application on top of it. In the traditional approach, business value is often only realized at the end of the project when the software components are integrated.

Although Agile development changes the way a product is developed, it does not change the need for sound product management. The company still needs a good up-front process to prioritize which ideas get funded. The company also needs to position and differentiate its product in the marketplace. Thus, the fundamental role of the product manager—to be the voice of the customer in development and ensure the right product is built—remains intact. But, with Agile, it is now possible to respond more quickly to the marketplace and deliver a higher quality product, sooner.

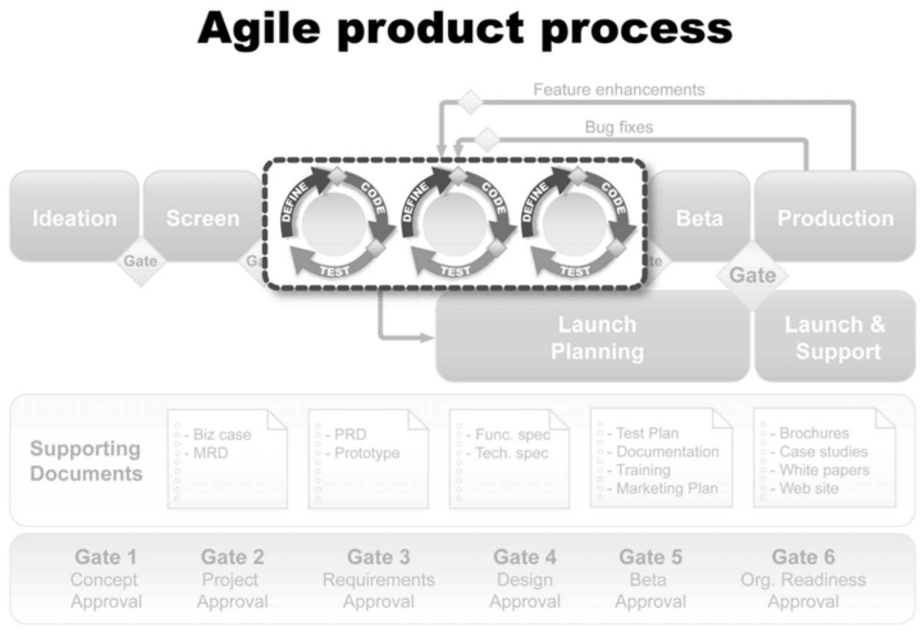

Figure 1.4: *Agile software development creates products through multiple, small iterations.*

Further, although the software is built through a series of small iterations, this does not in itself change the release cycle. If a company normally releases every six months, the company may choose to maintain that release schedule after implementing Agile. The difference is that it would develop the release in small iterations, for example, by combining six monthly iteration cycles. You should examine, however, whether your customers would prefer faster, smaller increments, or larger, less frequent releases. For example, if you currently release annually, could you instead release quarterly, accelerating value to your customers and making your product more competitive in the sales cycle?

If you use your Agility to release more often, your launch planning will need to adjust to accommodate the increased cadence. You may choose to shorten your beta programs or conduct them entirely differently. Web services companies often deploy betas directly on their live sites and expose a sub-segment of unsuspecting users to it. Others ask users to opt-in to using the new version. In addition to betas, documentation also changes, as well as how product management interfaces with development.

What's in It for You?

Why is changing to Agile development good for product managers? "What's in it for me?" you ask. There are three high-level reasons why product managers directly benefit from working with Agile teams:

- Greater **visibility** into the progress of a release, as you track your project through actual working software rather than status reports
- More **flexibility** to deal with changes during the development of a release, as priorities can be adjusted before each iteration
- Higher **quality** plus shorter QA cycles (on one project where I adopted Agile, QA shrunk from two weeks to four hours)

The increase in quality comes as a result of three measures that are ongoing throughout the development: automated unit testing, frequent integration, and acceptance testing. As a result of the automated unit tests, late changes and production-issue fixes become less risky. You can expect to experience new levels of confidence in your product and avoid the long delays associated with manual regression testing.

You will not just have visibility into the progress of a release but also greater transparency into the trade-offs regarding changes you make. When combined with the added flexibility from developing in short iterations, you will be able to more easily accommodate changing priorities and make better-informed decisions.

In the end, you will be able to build more of the right products, deliver them to the market sooner, and achieve superior results (and have more fun while you do it!).

Why Now?

Interestingly, Agile is not new. In fact it has been around since the 1990s. It is worth asking: Why now? Why has Agile become so popular?

To be a little controversial, I would say it is because of the failure of traditional serial or "waterfall" development and our attempts to control the software development process. Although I know there are many advocates of waterfall, in my experience it just does not work that well. Further, the Internet and globalization have had profound effects on the software industry, increasing the intensity of competition and the rate of change. If you are still releasing annually or over an 18-month cycle, by the time your next version is available, the market will have moved. Further, with the advent of Software-as-a-Service, hosted models, hosted download, and self-updating software, releases can occur with much greater frequency. Thus, companies have looked for a better way to develop software. Agile has met the challenge. Even so, best practices for Agile continue to evolve, including the constant challenge of working with distributed teams across multiple time zones.

The Agile Manifesto

In 2001, seventeen Agile pioneers[2] gathered at Snowbird Ski Resort in Utah to discuss common themes. The group captured its deepest beliefs in the "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.[3]

As you reflect on the meaning of the manifesto, it is worth noting that a lot of early software product development techniques were developed within IT and in the context of custom software development. So customer collaboration means collaborating with whoever represents the customer within your

2. 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, and Dave Thomas.

3. http://agilemanifesto.org/

organization. In commercial software development, the product manager (or sometimes a business analyst) is usually the proxy for the customer, representing their interests within the development process.

We in product management must also commit to the Agile Manifesto. As we look to embrace these values, we must understand how it effects our actions and how we carry out our responsibilities. Let's take a more detailed look at the four main principles of the Agile Manifesto:

Individuals and Interaction—Creating great products is purely a human endeavor. Identifying and understanding a customer problem, applying technology and design to a solution, and generating awareness of the solution is hard work. There are guidelines but no recipes for success. Further, the task is beyond the means of any single individual. Rather it requires a team of experienced professionals working in concert. Process can help ensure important steps are not missed, but it will never substitute for the judgment that each team member applies to thousands of decisions during the project. Good decisions emerge from teams whose members are aligned, honest with each other, and in which each member holds the team goal above his personal ambition. The product manager is central to maintaining a cohesive team with singular purpose. Towards this end, the product manager must foster an environment that respects the opinions, contributions, and individuality of each team member and where open and meaningful dialogue can occur.

Working Software—Our objective as product managers is to produce products that delight customers and solve a real need. Documentation can assist in achieving this goal, but documentation itself is not the goal. I know that I have not always held true to this value. I have been guilty of hiding behind documentation to deflect blame during a development project. If something were not coded as specified in the requirement, I would look incredulous and say, "I don't know how it was missed—it's right here on page 164 of the Product Requirements Document!" No doubt part of this reaction had to do with my insecurities as a young product manager, but it also had to do with the black box of development. This is not to say that the team members were hiding anything from me, or were not willing to show me what they had developed. It was just that integration rarely happened until the later stages of the project. The team did not have anything that it could show me or that I could interact with myself. I had no way of knowing if a requirement was poorly worded, misunderstood, or just missed entirely, until close to the end of the project. Further, being able to point to where the botched requirement was documented was of little consolation, because *we as a team had failed*: we missed an opportunity to improve our product in the eyes of the customer. Fortunately, Agile solves this problem with the frequent release of working software that provides a reference for meaningful conversation.

Customer Collaboration—In traditional serial software development, requirements and specifications documents are often signed off by the team and the product manager or by the company and the customer. Any scope change requires additional negotiation, trade-offs, and, possibly, increased fees. This creates a confrontational environment that prioritizes stability over building the right product. Thus, developing to the agreed plan may ensure contract compliance, but will not produce satisfied customers or users. Contract compliance is a short-term goal that requires the unrealistic effort of trying to define a product entirely up front. Acquiring satisfied customers who will give good references is the long-term goal, and it means working with the customer to understand what is needed, then working with the development team to understand what is possible.

Responding to Change—As product managers, we invest time talking to customers about their needs, observing them using our products, or solving their problems without our products; in addition, we validate concepts, prototypes, and software in pre-release form. At each stage of discovery, customers acquire new levels of understanding about our design intent, and we learn new lessons. We know changes will be necessary, and, therefore, we need to adjust our practices to accommodate change at all stages of the product development process. As in the case with contract compliance, above, following the plan is not the goal. Rather, our goal is working software that solves customers' problems, and maybe even delights them with its utility.

Post a copy of the Agile Manifesto on your wall and in your meeting rooms. If you find yourself or your team slipping back into old patterns, look at it again. The values embodied in these principles will keep your team grounded and moving in the right direction.

The Common Threads of Agile

Agile software development is not a single methodology but rather a development philosophy or approach. Many different methodologies fit under the Agile umbrella, including Scrum, Extreme Programming, Dynamic Systems Development Method, Feature Driven Development, Agile Unified Process, and many more. Common themes include:

- Development in short iterations or time boxes that typically last from one to four weeks
- Responsibility by the team during each iteration for the full software development cycle, including planning, requirements definition, analysis, design, coding, unit testing, and acceptance testing, at which point the product is demonstrated to its stakeholders

- Disciplined project management
- Frequent inspection and adjustment
- Collaboration between self-organizing, cross-functional teams
- Emphasis on customer needs

Agile techniques minimize overall project risk, and let the product adapt to change very quickly. At the end of each iteration, the product should be functional and able to be released. The product manager, however, may decide to combine multiple iterations to have enough new functionality to warrant a market release.

Why Agile Works

Agile works because it supports the process of software development in four key areas:

1. Empirical process
2. Daily visibility
3. Socialization of information
4. Rapid feedback cycles

Empirical Process

It is easier to understand an empirical process by first looking at a defined process. A defined process is one in which defined inputs generate defined outputs. This is the original manufacturing model. It is repeatable, and it is the model that traditional software development has assumed (Figure 1.5).

If I take a blueprint to any machine shop around the world, I can expect to get matching products back from all the shops. The blueprint specifies the material, the dimensions, and tolerances. The actual cutting paths may differ, but the end product will largely be the same. However, if I take a product requirements document and give that to ten different development teams, the resultant code bases may perform similar functions, but the products will probably look and behave differently and optimize along different dimensions. Further, the lines of code will be different and the programming language may be different. It would be a little like asking two journalists to cover the same story: although the facts of the resulting reports should be the same, the styles and perspectives will be different.

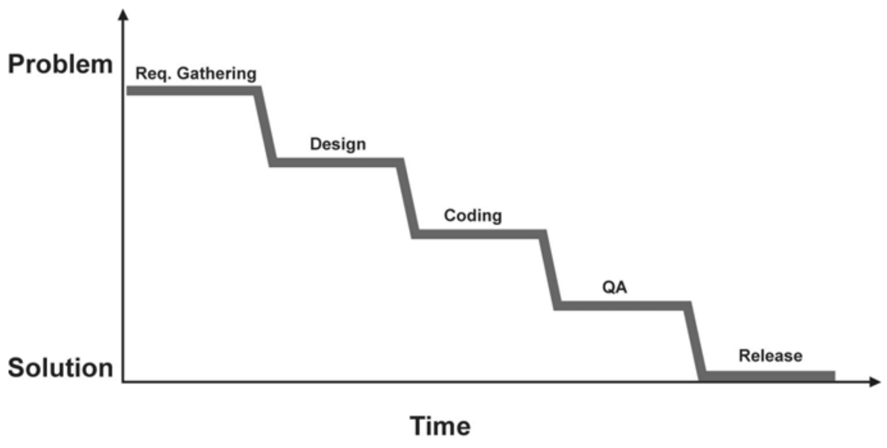

Figure 1.5: *Traditional software development assumes a defined process.*

In contrast, the empirical process is nonlinear. It requires frequent inspection and adjustment. It is well suited for new product development, which requires research and creativity—both relatively unpredictable activities. Software development, including the product management step of requirements gathering and validation, is best suited to an empirical model.[4]

Software development teams are always creating something new. They do not follow a straight path with a defined output that is the same each time they go through the process. Instead, teams cycle through knowledge creation and problem solving (Figure 1.6). As team members formulate the solution, new questions emerge and new requirements need to be gathered. Occasionally the team has to restart. An assumption turns out to be false, such as the availability of complete and accurate data to drive a critical process. Sometimes the environment changes and a new technology emerges that needs to be supported. At times the solution functionally works but the performance is slow. Further, there is no escaping the QA team, whom you can always count on to identify some good issues that need solving. The list goes on. To accommodate this, the team must frequently check its progress and adjust its plan.[5]

4. Ken Schwaber and Mike Beedle, *Agile Software Development with Scrum* (New Jersey: Prentice Hall, 2002), 24–25 and 106–108.

5. Steve Tennant, "Create Better Products with a Structured Process for Collaboration," (presented at the second annual Silicon Valley P-Camp for Product Managers, Sunnyvale, California, March 14, 2009).

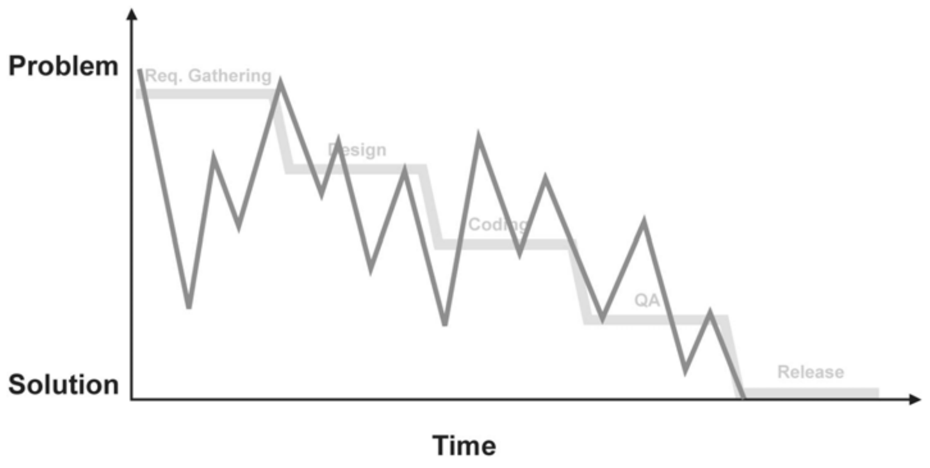

Figure 1.6: *Software development follows an empirical process that is similar to new product development.*

Daily Visibility

Agile improves visibility into the development process with a daily standup meeting. At the meeting, team members let each other know what they are working on and if they need help. Roadblocks are explicitly called out. Because this meeting is daily, issues cannot be ignored. Further, each developer commits to what she will be working on next in front of the team. Through this process, the team develops trust and the ability to be open and honest.

Socialization of Information

Agile processes facilitate the socialization of information. First, as mentioned above, the daily standup drives trust in the team. This creates an environment conducive to sharing news, even if it is discouraging. Secondly, because team members meet daily to share their accomplishments and plans, individual knowledge quickly becomes team knowledge, and the team stays aligned.[6]

Rapid Feedback Cycles

Rapid feedback cycles are embedded in the Agile process. The feedback loop accelerates the team's learning and supports the empirical process needed in software development. There are three connected cycles: the release, the

6. Schwaber and Beedle, *Agile Software Development with Scrum*, 111–113.

iteration, and daily standup. The release may be an extended cycle, but the iteration is one to four weeks, and the daily standup provides feedback every twenty-four hours. Agile feedback loops share much in common with the Plan, Do, Check, and Adjust (PDCA) process popularized by Dr. W. Edwards Deming in his quality control work: the team members formulate a course of action, execute against it, inspect their progress, and adjust their course accordingly.

Product Management Just Got Better

Returning to the main topic of this chapter, what does Agile mean for you as a product manager? It means your job isn't changing, but your ability to perform your job well has dramatically improved. Everything you should be doing as a product manager is still intact, including conducting market and customer research, creating personas and business cases, making tough trade-offs and prioritizations, planning roadmaps, developing pricing, creating sales tools, and launching the product. But with Agile, you can get feedback faster, incorporate research findings earlier, and adjust your plan sooner. You can expect to deliver better products, with richer features and fewer bugs, to your customers. Plus, you can reduce your time to market and accelerate revenue for your product.

Of course, you will need to change parts of your process to support the development team and leverage the benefits of Agile. This will be covered in the remaining chapters of this book. So get set as we first look at Scrum as an example of an Agile development methodology and then explore release management, release planning, documentation, and everything else you need to know to achieve ***Agile Excellence***.

About the Author

Greg Cohen is a 15-year Product Management veteran with extensive experience and knowledge of Agile development. He is a certified Scrum Master, member of the Agile Alliance and former President of the Silicon Valley Product Management Association. He has worked for venture start-ups and large companies alike, and has trained product managers from around the world on Agile development methods. As a practitioner and frequent commentator on product management issues, he has written and spoken on varied topics such as embracing Agile development, Lean product management, and recession proofing your career.

Greg earned an MBA with honor from Babson College and a Bachelor of Science in Mechanical Engineering with second major in Electrical Engineering from Tufts University.