HOWARD PODESWA
Foreword by ALAIN ARSENEAULT

Rough Cuts

The Agile Guide to Business Analysis and Planning

From Strategic Plan to Detailed Requirements

The Agile Guide to Business Analysis and Planning

From Strategic Plan to Detailed Requirements

Howard Podeswa

Addison-Wesley

Contents at a Glance

Contents

Chapter 1. The Art of Agile Analysis and Planning

This chapter provides a personal introduction to agile analysis and planning from my perspective as an artist and IT professional. It explains how the approach supports creativity and responsiveness. The chapter uses storytelling to introduce the main themes of the book, with two narratives that illustrate the value that the competency brings to the business.

1.1 OBJECTIVES

This chapter will help you

- Understand how agile analysis and planning supports a creative, adaptive process of product development.
- Understand through examples, how the competency benefits the business.

1.2 ON ART AND AGILE ANALYSIS

I come from a family of artists: my grandfather, uncle, and father were painters. (One of my father's works is shown in Figures 1.1) I am one, too. Throughout most of my time in software development, I've had a parallel life as a professional artist.

Figure 1.1 *Yidel Podeswa*, Apples, *2012, oil on canvas. (Photo by Toni Hafkenscheid.)*

From my art practice, I've learned a valuable lesson about the creative process. If you want to create a good painting every time reliably, you need to plan and execute each step very carefully. First, you prepare the canvas, then you make a complete drawing of the final image, and only then do you start to apply paint. I created the painting in Figure 1.2 using that process.

Figure 1.2 *Howard Podeswa*, Clementine, *2019.*

However, if you want to create something truly innovative, something original that has never existed before, you can't use that process because you don't know beforehand what the result is going to look like. It's something you figure out in the making. And so, you just start with a rough vision of what you want; then you have to experiment—iteratively refining that vision over time through a process of trial and error. Figures 1.3 through 1.5 illustrate how I used this process to create the painting series, *A Brief History*, exhibited in the Koffler Gallery (Toronto, Ontario) in 2016 and Kelowna Art Gallery (Kelowna, British Columbia) in 2017. Figure 1.3 is an installation view of the two main paintings, *Heaven* (2015) and *Hell* (2013).

Figure 1.3 *Howard Podeswa*, Heaven and Hell, *installation view, 9 ft × 15 ft, Koffler Gallery, 2016. (Photo by Howard Podeswa.)*

When I began work on the series in 2012, I had a rough vision for the exhibition and the two central paintings in it. I wanted viewers to have to make a choice—to face Heaven or Hell—and to have an immersive experience whichever choice they made. I wanted that experience to invoke a reflection on where we are headed—the future. The series and exhibition title, *A Brief History*, is an allusion to Stephen Hawking's writings on this question from the point of view of astrophysics. I had a

rough concept for the *Hell* painting: a vortex suctioning in the dark images that were beginning to fill our newsfeeds. And finally, I knew the constraints with which I had to work—such as the dimensions of doors and elevators the paintings had to pass through and the height of the gallery. I left everything else to be worked out through prototyping and trial and error. Figure 1.4 shows an early prototype for the painting.

Figure 1.4 *Howard Podeswa*, Hell, *2012 (work in progress, early version), 9 ft × 15 ft. (Photo by Howard Podeswa.)*

Figure 1.5 shows the final version of the painting.

Figure 1.5 *Howard Podeswa*, Hell, *2013 (final state), 9 ft × 15 ft. (Photo by Toni Hafkenscheid.)*

Comparing the two versions, you can see that some of the features that were present in the early prototype have been retained: the circular composition, the central perspective point inside the circle, as well as the overall tenor of the images. Nevertheless, the final result is significantly different from earlier versions. One feature —the circle motif—has become much more emphasized. All of the details have changed. This process is strikingly similar to the agile analysis and planning process you'll be reading about in this book: It begins with a broad vision of the product and how users will experience it. It continues with an analysis of constraints and context for the usage of the product. It proceeds with a series of prototypes and "sketches" to test out hypotheses about the product. And it uses an iterative process based on intervention (trying something out), measurement (gathering user feedback), and adjustment to determine the features that make their way into the product.

What is true in art is true in software development: the traditional, step-by-step approach to analysis and planning—what software developers refer to as the *waterfall* approach—works pretty well when you are working on something routine. But when you're working on something truly innovative, about which little can be known in advance with certainty, it's more effective and natural to adopt an experimental approach —the agile analysis and planning approach that is the subject of this book. As we'll see, this approach even has benefits for noninnovative development, too. For example, it shortens time to market by focusing effort on high-value features, using data to inform decisions.

1.3 I WORK FOR A MAINSTREAM COMPANY! WHAT'S THIS GOT TO DO WITH ME?

Some readers might now be thinking, "I work in a mainstream business. I'm in

insurance/finance/government (fill in your business domain), where it's all about *reliability*. What does 'agile' have to do with me? We don't *do* trial and error!"

Here's an example of why agile development is still relevant in these contexts. These days, I'm working with teams on a usage-based insurance (UBI) product. The product will base costs and benefits on customer behaviors: what times of day they drive, how they accelerate and brake, where they travel, and so on. It's a very controversial product to develop because of privacy issues. And that's the point. There's so much uncertainty about what features and inducements customers will and won't accept that if analysts specified *all* the requirements upfront, much of their work would likely end up on the cutting room floor. Instead, they are using an approach similar to the one I've described for creating novelty in art—one that begins with a broad vision and gradually brings features into focus through a process of experimentation: trying something out and making adjustments based on the response. That is the agile approach.

This kind of product development is not an anomaly today in mainstream businesses. As new data sources and Internet of Things (IoT) devices proliferate, so, too, do opportunities for innovation, even in traditional business domains. Add in the accelerated pace of change all businesses are experiencing today, and it's no wonder that the adoption of agile approaches is now widespread even in mainstream business domains. Since these traditional business domains also happen to be the natural habitat for business analysis (BA), two disciplines—BA and agile development—are now in close contact with each other in ways they rarely used to be. The result is the emergence of an agile analysis and planning competency—BA, as practiced in an agile context, in line with agile planning approaches. Why analysis *and* planning? In agile development, the two

activities are too tightly intertwined to be effectively separated: the backlog is in a continual state of flux, with changes affecting both the *analysis* of requirements items (features and stories) and the *plan* for rolling them out. Consequently, a single paradigm is needed that encompasses both: Agile *business* analysis and planning." For convenience, I'll also be referring to the competency by the slightly shorter term, "agile analysis and planning."

Let me give you a personal perspective on the evolution of the competency. I started out in IT as a developer for Atomic Energy of Canada (AECL) and eventually moved on to systems design and analysis. In the late 1990s, my clients began asking me for help developing a new competency—business analysis—one that borrowed many of the techniques I had used in the technical realm and applied them to analyze a business and its needs. Through this work, I became an early contributor to the emerging field of BA: I created Boston University Corporate Education Center's first BA certification program, acted as BA subject matter expert for the National IT Apprenticeship System (NITAS) BA program (an apprenticeship program from the US Department of Labor and CompTia) and created internal BA training programs for companies such as Bell, CGI Inc., Bank of Montreal (BMO), and Moody's. As the discipline evolved, I came to see it as being about much more than requirements. Today I would describe it informally as follows:

> Business analysis is the examination of a business (or any other organization), or an aspect of it, in order to understand what needs to change to achieve a desired outcome.

From my development experience, I knew that iterative-incremental development (a process involving short development cycles, or *iterations*) and the broader agile approach that grew from it were the way of the future. I

intuited early on, though, that the BA competency was going to have to adapt to this newer way of working. I was fortunate to have a number of clients who felt similarly and who trusted me to help guide them on that journey. Many of these clients, such as Willy Rose at the Canadian Imperial Bank of Commerce (CIBC), were true pioneers—early adopters (back in the late 1990s) of iterative-incremental practices in mainstream businesses. That work resulted in my first book, *UML for the IT Business Analyst*.[1]

[1] Howard Podeswa, *UML for the IT Business Analyst: A Practical Guide to Object-Oriented Requirements Gathering* (Boston: Thomson Course Technology, 2005).

In those early days of agile development, most of the organizations I worked with were islands of agility within their companies; the larger enterprise still operated along traditional lines. That all changed when I was called into a meeting in 2013 by a vice president at a large telecom company. Here's what he told me: "Howard, in five or six years, the whole organization will be agile. You've been working with our business analysts. What do you think's going to happen to them? I've got agile consultants running around saying you don't need analysts in agile. If that's the case, what am I supposed to do with all my analysts?"

I immediately recognized this meeting as a pivotal moment. This executive wasn't talking about an island of agility but the transformation of an entire enterprise in a traditional business sector—the natural habitat of the business analyst. Either BA was going to adapt, or it would disappear. It was not yet clear at the time whether BA *would* survive, as it wasn't just the consultants at this one company who had been dismissing analysis. I still own a copy of *Agile Software Development with Scrum*[2]—the book that introduced the agile Scrum framework to the world in 2002. About the only reference to requirements analysis I can find in

it is the advice *not* to spend too much time on it![3] Yes, the book does deal with product backlog items—work items that can represent requirements. But it is as though those items just show up magically in the product backlog. There is very little about the planning and analysis work required to get them there.

[2] Ken Schwaber and Mike Beedle, *Agile Software Development with Scrum* (Upper Saddle River, NJ: Prentice Hall, 2002), 92–93.

[3] Schwaber and Beedle, *Agile Software Development*. The reference is to "If too much time is spent thinking about requirements, competition gets to the market first" (p. 92). The authors also discuss the relationship between requirements uncertainty and "noise" (p. 93) in their argument for using empirical processes when requirements are volatile.

The same year that I met with the telecom executive, 2013, I delivered a talk on agile BA at the Norway Developers Conference (NDC). Mine was the only one of about fifty presentations to address agile BA. It simply wasn't on the radar for most agile practitioners. It was entirely reasonable to believe that BA and business analysts would not survive the transition to agile practices.

You may be wondering what happened after that conversation with the vice president. Over the next couple of years, I worked with the company's teams across the country on the analysis aspects of their agile transition. The company ended up keeping its business analysts and includes them today on most agile teams. Why? The analysts have proven their worth by filling a competency gap on agile teams—contributing soft skills, elicitation skills, domain knowledge, and valuable analysis expertise in areas such as workflow modeling, domain modeling, and business rules. Right across the agile development lifecycle, business analysts have been critical to ensuring *business value is maximized* whenever decisions are made. (We look further at the broader research on this topic and the value proposition

for agile analysis in Chapter 2, "Agile Analysis and Planning: The Value Proposition.")

In keeping with the agile preference for communicating through stories, the following are two "truthy" (with thanks to Stephen Colbert) stories about agile analysis and planning. They are based on real events, but I've changed the details for reasons of pedagogy and confidentiality. One story gets into the weeds of agile analysis techniques, and the other deals with agile culture. Together, I hope they give you a feel for the value that agile analysis brings to an agile development organization and the business as a whole.

1.4 STORY 1: IT'S NOT MY PROBLEM

This story is called "It's Not My Problem"—but, really, it's a morality tale about why you shouldn't skip BA in agile development.

I became involved in this story when one of my clients, a company in the health-care sector, asked me to advise it regarding some recurring issues the company was experiencing. During a break in one of our meetings, a high-level manager pulled me aside to tell me, "There's a problem here that everyone recognizes, but no one wants to talk about in public." That problem was that, time and again, stories that performed well on their own would fail during end-to-end user acceptance testing (UAT). Since UAT usually happened late in the release cycle, it inevitably led to last-minute panic and missed deadlines or a release with missing features.

Figure 1.6 illustrates the issue the company faced.

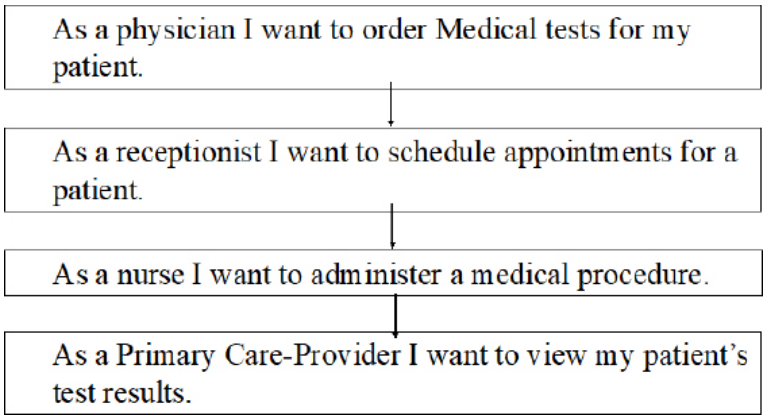

Figure 1.6 User acceptance testing for managing medical laboratory tests

Each rectangle represents a requirements item—a user story or feature. The stories are shown in the order they are usually performed by users: first, a physician orders a medical test, then a receptionist schedules an appointment for the procedure, and so on. During implementation, each team in the group would focus on a different user role (physician, receptionist, etc.). As each team finished a user story, the story was tested against its acceptance criteria. Once all the stories were completed and individually tested, QA pulled them together for UAT in the sequence shown in Figure 1.6. It's at that point that the test would fail.

I had a strong hunch about what was wrong: I guessed that the analysis of the end-to-end process for ordering and administering a medical procedure must have been skipped over during the preparations prior to development. The manager confirmed that this was, indeed, the case. The trouble was that *she couldn't convince the team to do anything about it*. She offered several reasons why. Fundamentally, many of her developers seemed to believe that you don't have to do process analysis in agile. She thought some of the outside agile consultants who had been working with the teams might have spread this view. Another issue was that the analysis of the end-to-end process didn't fall within any particular team's wheelhouse, since each team was

focused on only one user's perspective. Anyone who *might* have wondered about it simply assumed that another team was doing the analysis.

I first addressed the mistaken impression that team members had about analysis in agile development and explained that the competency is widely accepted as an essential activity in agile development today. (*The Scrum Guide*[4] refers to this work as product backlog *refinement*.) Then I offered three mechanisms for ensuring that the scenario she described wouldn't recur. The first was for her analysts to adopt the acceptance test–driven development (ATDD) practice of specifying end-to-end tests for features up front.[5] To ensure that this work occurs before feature implementation, I suggested that her teams define and use a feature definition of ready (DoR). The DoR is a set of rules—a checklist of characteristics—that an upcoming requirements item must have. To avoid the type of scenario she described, I suggested she include the following condition in the feature DoR:

[4] Ken Schwaber and Jeff Sutherland, "Sprint Planning," in *The Scrum Guide: The Definitive Guide to Scrum—The Rules of the Game*, Scrumguides.org, 2017, https://www.scrumguides.org

[5] The common way to do this is to create a feature file for the whole process or value stream in the backlog and specify end-to-end test scenarios using the Gherkin/BDD format. See Jens Engel, Benno Rice, and Richard Jones, "Feature Testing Setup," GitHub, ©2012–2019, https://behave.readthedocs.io/en/latest/gherkin.html

> The end-to-end process or value stream for the feature has been analyzed enough to determine the sequencing of steps (stories), the integration points, and determine the UAT required to test the feature.

Finally, to ensure this analysis work was taken into account during planning, I suggested her analysts capture it in a functional spike (referred to as an *enabler story* in the Scaled Agile Framework [SAFe])—a special kind of story reserved for the analysis of upcoming user stories.

In this book, you'll find guidance on each of these issues. For example, in Chapter 6, "Preparing the Process," you'll learn what to include in a feature DoR. In Chapter 13, "Story Preparation," you'll learn how to specify and use spikes. In Chapter 17, "Scaling Agility," you'll learn about ATDD and how to use the DoR during refinement events.

1.4.1 Conclusions

Here are the key points covered in this story:

- The agile analyst helps prevent last-minute integration issues through ATDD and behavior-driven development (BDD) practices.
- The agile analyst prepares features before implementation to avoid unnecessary rework.

1.5 STORY 2: THE CANTANKEROUS CUSTOMER

This story is about how you can get all the details right about agile practice but still fail if you don't pay sufficient attention to culture. The story began when I got some pushback from a director at a financial institution over some guidance that I had given her teams about estimation methods: I had advocated for the use of real-time estimates over a more abstract metric—story points. (I've since shifted my position on this issue.) As we were discussing the pros and cons of the approaches, I learned about the real problem behind this discussion: acrimonious planning meetings. I was told that during these meetings, the customer would often challenge individual developers on their estimation of effort. It was this friction that had caused her to abandon the use of real-time estimates.

When I first heard this, my initial reaction was, "There's nothing necessarily wrong with some friction. In agile development, the customer is *supposed to* negotiate and

challenge the developers to find better ways to deliver value with less effort." But I found out that's not what was happening here. The customer and the team weren't negotiating about trade-offs; business stakeholders simply didn't *believe* the estimates given by the developers and testers. Now, this is *not* the kind of challenge that is encouraged in agile practice. To create an environment for requirements to emerge organically, you need, instead, to nurture a *collaborative* culture between business stakeholders and development. Each party is expected to respect, not challenge, the other's area of expertise.

Because of this negative experience, the director had become a strong advocate of story points—a relative versus absolute metric. As she explained it, story point estimation removed one of the most acrimonious issues from discussion—how much *time* a requirement item should take. In its place, it focused discussion on a much less controversial topic—the *relative effort* to implement one requirement item versus another.

I agreed with the director that the use of story points was a useful quick fix. However, to fix the problem at its root, I suggested she look at the cultural issues at play. I advised she gain consensus on a bill of rights and responsibilities for the customer and the development team. One of those rights is the sole right of those *performing* the work to estimate the effort to do it. Balancing this right, customers have the right to decide what the requirements are and how to prioritize them in light of business value and cost. Customers also reserve the right to change their minds about that until implementation, without undue penalties. When rights and responsibilities are agreed upon up front, I argued, future confrontations can be avoided. In this book, you'll find detailed guidance on these issues. For example, in Chapter 5, "Preparing the Organization," you'll learn about customers' and developers' rights and

responsibilities. In Chapter 11, “Quarterly and Feature Planning,” and Chapter 14, “Iteration and Story Planning,” you’ll learn how to plan requirements with those rights in mind. And in Chapter 18, “Achieving Enterprise Agility,” you’ll be provided with a more expansive look at agile culture and what it means from an enterprise perspective.

1.5.1 Conclusions

Here are the key points covered in this story:

- Getting the culture right is key to success in any agile endeavor.
- Foster collaboration between the customer and developers by gaining consensus on a bill of rights and responsibilities for each side in the relationship.
- Those doing the work have the sole right to estimate the work, while the customer has the exclusive right to prioritize it.

1.6 CHAPTER SUMMARY

Here are the key points covered in this chapter:

- Agile analysis and planning is about beginning with an unformed vision and allowing detailed requirements to emerge through a process of iterative experimentation and feedback.
- Agile analysis reduces unnecessary rework by ensuring that integration issues are identified up front.
- Agile business analysts are critical to propagating cultural change.

1.7 WHAT’S NEXT?

Chapter 2 explores the fundamental concepts that underlie the agile business analysis competency and the rest of this book. The aim of that chapter is to level-set

readers. If you already have a strong background in those areas, skip ahead to Chapter 3, "Fundamentals of Agile Analysis and Planning."

Chapter 2. Agile Analysis and Planning: The Value Proposition

Chapter 1, "The Art of Agile Analysis and Planning," presented my personal perspective on agile analysis and planning. This chapter examines the origin of the competency and the research concerning its efficacy. It first presents the parallel histories of the approaches that gave birth to agile analysis and planning: agile development and business analysis. It then reviews the research on the value proposition for a company developing the competency.

2.1 OBJECTIVES

This chapter will help you

- Understand the histories of business analysis, agile development, and agile analysis and planning.
- Know the benefits of business analysis and agile development.
- Understand the value added by including an effective agile analysis competency in an agile organization.

2.2 WHAT IS AGILE ANALYSIS AND PLANNING?

The short definition of agile analysis and planning is that it is business analysis (BA) plus planning as practiced in an agile context. That definition only helps, though, if you know what the individual terms mean. Let's review, then, the more extended definition introduced earlier in the book:

> **Agile analysis and planning** is the examination of a business or any aspect of it (culture,

> organizational structure, process, products, and more) to learn *what* needs to change and *when* in order to achieve a desired outcome in a context that places a high premium on adaptability, resilience, and continuous value delivery. It includes analyzing who the product is for (the stakeholders), defining their requirements, determining when the capabilities will be delivered, and estimating costs and resources.

As explained in Chapter 1, I bundled BA together with requirements implementation planning because both are in a continual state of flux in agile development and therefore are inextricably linked. Changes to requirements and priorities affect the implementation plan, and changes to the plan affect the timing of the analysis. A more precise term for the discipline is *agile business analysis and planning*, since *analysis* without that qualifier could refer to any sort of analysis (e.g., financial, data, or systems analysis). For brevity, however, I more often use *agile analysis and planning* in this book. You'll occasionally see the even shorter term, *agile analysis*. Don't attach any special meaning to these variants; they all refer to the same competency.

As the definition points out, agile analysis and planning occurs in an environment that places a high value on adaptability and continuous delivery (CD). This context distinguishes it from traditional analysis and planning, where the business sets a higher value on *predictability* and sticking to a preset plan. This shift in context has a profound impact on the way the competency is carried out. For example, in a waterfall context, the requirements specifications and requirements implementation plan are frozen prior to development. Changes can be made but only after following a rigorous change management process. In agile development, however, we expect and even encourage changes to the

requirements and plan because they enable the business to respond quickly to learning and adapt to the market.

You might think that the kinds of businesses that would value adaptability and continuous delivery—and thus be natural candidates for agile development—would be disruptive innovators. You'd be partially right: those companies use the approach because it enables them to test hypotheses for the product and respond effectively to learning. But agile development is also widely practiced by established companies because it provides them with the strategic capability to react to changes in the market and technology without sacrificing reliability.

2.3 WHO IS A BUSINESS ANALYST?

This book is primarily concerned with agile BA as a competency and a practice and is less concerned with the title of the person doing the work. Unless otherwise specified, when I use the term *business analyst* in this book, I mean *any* person engaged in BA activities, regardless of their formal title. In practice, those who perform these activities may have the title of business analyst, IT business analyst, team analyst, business systems analyst—or even a non-BA title such as data analyst, tester, UX professional, or programmer.

This usage is in line with the International Institute for Business Analysis (IIBA), which defines a business analyst as follows: "any person who performs business analysis tasks described in the *BABOK® Guide*, no matter their job title or organizational role. Business analysts are responsible for discovering, synthesizing, and analyzing information from a variety of sources within an enterprise, including tools, processes, documentation, and stakeholders."[1]

[1] International Institute of Business Analysis (IIBA), *BABOK v3: A Guide to the Business Analysis Body of Knowledge*, 3rd ed. (Toronto, Canada: IIBA, 2015), 2–3.

In practice, organizations often use more fine-grained role names for subsets of BA responsibilities. For example, in many of the organizations with which I work, the business analyst is typically someone concerned with high-level business requirements, whereas the team analyst is concerned with detailed product requirements. As a general rule, I avoid these distinctions in this book: a business analyst, or analyst, is simply any practitioner of BA. The exception is a context where it's important to specify the role. For example, I may use the term team analyst when discussing guidelines that apply only to business analysts at the team level.

2.4 WHY AGILE ANALYSIS AND PLANNING?

In Chapter 1, you looked at the research on the benefits of agile analysis and planning. To give you a more tangible feel, let's follow the story of one specific, fictionalized company: Customer Engagement One (CEO) Inc.

CEO is developing an app, also called CEO, that enables any business to manage its engagements with customers across multiple platforms. The early version of the app was built as a simple, monolithic system using an agile methodology. The typical change request could be accommodated by a single agile team working independently. Consequently, teams got by with minimal preparation and planning.

As the product grew, however, it became more complex. A typical change now required more competencies than could fit within a single agile team. Teams now needed to collaborate more frequently, but doing so was difficult with their current minimal approach. They began to suffer from bottlenecks as they waited for others to perform work they were dependent on. Features delivered by individual teams were inconsistent, leading

to rework and delays. Stories couldn't be completed on time because there were essential requirements that were not well enough understood at the time estimates were made. The user experience of the app became fractured because teams didn't have a shared understanding of the vision for the product. The company had tried to launch a product-wide, strategic initiative to provide a more consistent experience but had given up because there never seemed to be enough teams ready at the same time to pull off a change of that size.

At this point, the development lead for CEO realized that the organization needed to spend more time on analysis and planning and develop more effective processes for carrying it out. Staff were trained in the competency and deployed to agile teams. Product owners were more productive because they could focus on market-facing responsibilities while team-level analysts took the lead on day-to-day team-facing activities.

The planning process in the organization was changed so that a portion of each team's budget was now reserved for strategic initiatives. These were planned quarterly. The rest of the budget was earmarked for customer-generated requests. These were planned using the Kanban approach to enable fast response to learning and customer feedback. Business analysts started to prepare features and user stories in advance of planning, resulting in improved estimates. This practice allowed teams to forecast completion dates better and coordinate their work in advance. As a result of these changes, the CEO was now able to take on large initiatives.

The previous story is not as fictional as I suggested. Other than the name, it's the story of an actual company. The agile analysis benefits that it describes are available to any company willing to spend the time and energy to develop the competency.

2.5 THE PARALLEL HISTORIES OF AGILE AND BUSINESS ANALYSIS

Agile analysis and planning is, as noted earlier, a hybrid of two disciplines: agile planning and development and BA. Figure 2.1 indicates how these two approaches co-evolved from the 1940s to 2017. Significant events in BA are indicated in red; events in agile development are indicated in blue; those related to agile BA are shown in purple.

Figure 2.1 Agile business analysis timeline

Following is a brief survey of that history, as summarized in the figure. We focus first on the history of BA, then on the concurrent evolution of agile development.

2.5.1 A Brief History of Business Analysis

Business analysts began to appear during the late 1990s when companies began to introduce the role into their software development organizations. The early focus of

business analysts was on requirements—specifically, the discovery, creation, communication, management, and validation of software requirements. Today the BA competency covers a much broader scope up to and including enterprise and strategy analysis. *BABOK v3: A Guide to the Business Analysis Body of Knowledge (BABOK Guide)*, third edition, defines the practice as follows: "**Business analysis** is the practice of enabling change in an enterprise by defining needs and recommending solutions that deliver value to stakeholders. . . . Initiatives may be strategic, tactical, or operational."[2]

[2] IIBA, *BABOK v3*, 2.

In 2005, the IIBA began to offer professional BA certification based on the *BABOK v3*. Other bodies soon followed suit: the International Requirements Engineering Board (IREB) introduced its Certified Professional for Requirements Engineering (CPRE) certification in 2007. The Project Management Institute (PMI) offered its PMI Professional in Business Analysis (PMI-PBA) certification in 2015.

2.5.2 A Brief History of Agile Development

The origins of agile development can be traced back to the 1940s and 1950s with the introduction of Kanban and lean thinking. Surprisingly, though these practices have become foundational to agile software development, they were developed for an entirely different business context—the manufacture and design of automobiles—and only later adapted for software.

As Figure 2.1 indicates, the spiral model and the objectory process (later known as Rational Unified Process [RUP]) were introduced in the 1980s. These processes were based on *iterative development*—the development of software in short cycles or iterations—an

approach that was to become a central practice in agile development.

Two iterative frameworks were added in the mid to late 1990s: Scrum and Extreme Programming (XP). Both used a *timeboxed approach to planning*, whereby all the work for an upcoming iteration (the timebox) is committed to upfront. Today, Scrum is one of the most widely used agile frameworks—and even those who don't adopt the entire framework use concepts and events popularized by Scrum, such as velocity and the daily standup. XP has contributed practices, such as pair programming. Its most significant impact on agile analysis and planning is the use of stories, story point estimation, and the Planning Game.

These various precursors to agile development came together in 2001 when seventeen people met in the Utah mountains and emerged with *The Agile Manifesto* and *12 Principles behind the Manifesto*. Later that year, some of its authors and others formed the Agile Alliance, a nonprofit organization, to promote agile practices. Today, the organization has more than seventy-two thousand members.[3]

[3] "About Agile Alliance," 2020, https://www.agilealliance.org/the-alliance

Since the writing of the Manifesto, many other frameworks and practices have been added to the agile family. These include test-driven development (TDD) and its extension, acceptance test–driven development (ATDD), and the Scaled Agile Framework (SAFe) for scaled agile organizations, released in 2011.

In 2009 and 2010, two essential practices were introduced that made it possible to deploy updates frequently without sacrificing reliability: DevOps and continuous delivery. These practices have enabled agile development to fully achieve its promise, as stated in its

original principles, “to satisfy the customer through early and continuous delivery of valuable software.” Using DevOps/continuous delivery practices, product improvements can be safely released multiple times per day instead of once every two to three months, as used to be the case when I was working in the 1990s when RUP was prevalent.

In practice, BA and agile development have been merging since the mid-1990s, when IT departments in large companies began to experiment with iterative development methodologies such as RUP. I worked with waterfall business analysts at that time to help them transition to an iterative-incremental approach using techniques such as use-case scenarios to plan requirements implementation. The official merger of the two streams happened in 2015 with the publication of the *Agile Extension to the BABOK Guide v1.0*, soon followed by a second version, jointly published by the IIBA and Agile Alliance in 2017. That same year, the PMI published the *Agile Practice Guide* for project leaders and project teams.

2.6 TWO DIAGNOSES FOR THE SAME PROBLEM

It’s not entirely a coincidence that agile software development and BA both began around the same time—the 1990s and early 2000s. Both were responses to the same problem vexing software engineers at the time: a disturbingly high number of software projects were deemed failures. For example, the Standish Group’s 1995 CHAOS Report found that 31.1 percent of projects were cancelled before they were completed and that only 16.2 percent of software projects were completed on time and on budget.[4] Others reported that 64 percent of features were never, or rarely ever, used.[5] Business analysis and agile development were both proposed as ways to resolve this issue. However, they proceeded from different

diagnoses of the underlying problem. Let's look at each diagnosis and the evidence regarding its success.

[4] Project Smart, *The Standish Group Report: CHAOS* (Standish Group, 1995/2014), 3.

[5] Mary Poppendieck and Tom Poppendieck, *Lean Software Development: An Agile Toolkit* (Boston: Addison-Wesley, 2003), 32. The authors quote a Standish Group study reporting 45 percent of developed features never used and 19 percent rarely used.

2.7 THE BUSINESS ANALYSIS DIAGNOSIS

Business analysis began with the hypothesis that the root cause of project failure was communication—specifically, *poor communication* between business stakeholders and solution providers. There was ample research to back up this hypothesis. For example, the Standish Group reported at the time on the top ten causes of challenged projects.[6] The top two factors pointed to poor communication problems: lack of user input and incomplete requirements and specifications.[7] So did four other factors in the list: *unrealistic expectations, unclear objectives, changing requirements*, and *unrealistic timeframes*.

[6] Project Smart, *CHAOS*, 9.

[7] Project Smart, 9.

To address this root cause, some IT organizations began to define a new role—the IT business analyst. Initially, many business analysts were former systems analysts. I was one of those who made the transition—intrigued by the new competency because it included the stakeholder interactions I had come to enjoy as a systems analyst but without the technical aspects of the role. After years working at the cutting edge of technology, I was burning out and ready to make the switch.

Many other former IT developers did the same, but just as many moved to BA from business operations. The first business analysts that I trained were social workers

who were about to be assigned to work as business analysts on a project to replace the case-management system they had been using. Today, I'd estimate that about half of today's business analysts come from the business side. That's a good thing because it means that those with in-depth knowledge of the users and business context for a software product can take a leadership role concerning its requirements.

2.8 THE BUSINESS ANALYSIS TRACK RECORD

The data indicate that the BA approach has been highly successful. Let's begin with growth metrics. In 2004, one year after IIBA was founded, it had thirty-seven members. Today it has almost thirty thousand members, more than one hundred twenty chapters in over forty countries.[8] PMI reports that "Business analysis has become a competency of critical importance to project management."[9] It's doubtful we would be seeing this kind of growth and acceptance unless the BA competency was providing significant value.

[8] IIBA, "About IIBA," 2020, https://www.iiba.org/about-iiba

[9] Project Management Institute, "PMI Professional in Business Analysis (PMI-PBA)®," 2020, https://www.pmi.org/certifications/business-analysis-pba

What do software development organizations themselves think about the importance of BA to their success? Some insight is provided by the Standish Group's *CHAOS Report 2015*, which lists the main factors cited by organizations for project success. Table 2.1 summarizes the study's results.

Table 2.1 CHAOS Factors of Success

Factors of Success	Points	Investment
Executive Sponsorship	15	15%
Emotional Maturity	15	15%
User Involvement	15	15%
Optimization	15	15%
Skilled Resources	10	10%
Standard Architecture	8	8%
Agile Process	7	7%
Modest Execution	6	6%
Project Management Expertise	5	5%
Clear Business Objectives	4	4%

Source: Adapted from Standish Group International, *CHAOS Report 2015* (Standish Group, 2015), 11.

While the study doesn't explicitly call out BA, three of the top ten factors listed are directly addressed by the competency:[10]

[10] Shane Hastie and Stéphane Wojewoda, "Standish Group 2015 CHAOS Report—Q&A with Jennifer Lynch," 2015, https://www.infoq.com/articles/standish-chaos-2015

1. *User Involvement:* Including user feedback, requirements review, basic research, prototyping, and other consensus-building tools.
2. *Clear Business Objectives:* The development of a shared understanding across all stakeholders and participants about the purpose of the project and how it aligns with the organization's goals and strategy.
3. *Optimization:* A structured process for improving business effectiveness.

You might have noticed that one of the other factors is *agile process*. We'll get back to that later in this chapter.

What does the data say about the impact of BA on cost? One of the most scientifically rigorous studies of its kind, the Business Analysis Benchmark,[11] found that organizations using poor requirements practices spent *62 percent more* on similarly sized projects than organizations using the *best* requirements practices.[12] Figure 2.2 illustrates how this added cost, referred to by

the study's authors as the *requirements premium*, decreases to zero as requirements quality increases.

[11] Keith Ellis, *Business Analysis Benchmark—The Impact of Business Requirements on the Success of Technology Projects* (Toronto, Canada: IAG, 2009), 2–3.

[12] Ellis, *Business Analysis Benchmark*. Specifically, the study found that an organization with best requirements practices spent on average $3.63 million on a project originally estimated at $3 million, while an organization with poor requirements practices paid $5.87 million for a project estimated at $3 million—representing a 62% cost increase.

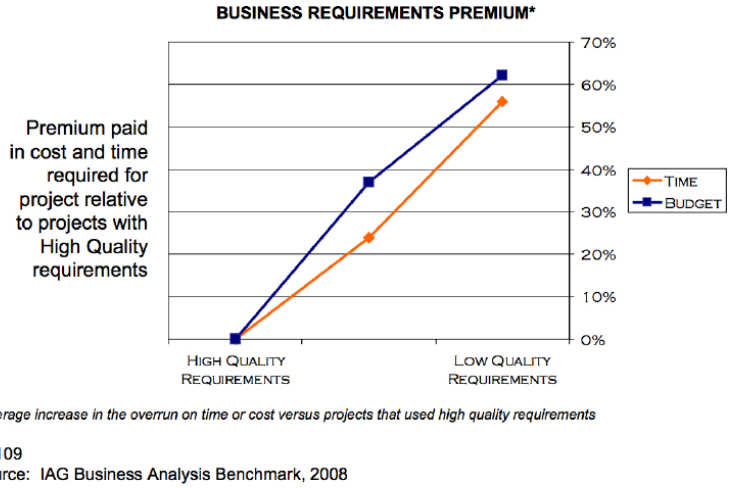

Figure 2.2 Business requirements premium: average increase in the overrun on time or cost versus projects that used high-quality requirements. *N* = 109. (Source: Keith Ellis, *Business Analysis Benchmark*—The Impact of Business Requirements on the Success of Technology Projects *[Toronto, Canada: IAG, 2009], 9.*)

The benefits don't end there. The report found that *every measure* in its study improved as requirements discovery and management maturity levels rose. These include

- Percentage of projects delivered on time
- Percentage of projects delivered on budget
- Percentage of projects delivering all required features
- Percentage of projects deemed successful

The following are some of the other conclusions of the study regarding the benefits of BA:[13]

[13] Ellis, 8.

- In companies that have only an average level of requirements competency, the lack of excellence will consume approximately 41.5 percent of the IT development budget on strategic projects.
- Companies in the upper third of requirements capability report 54 percent of projects on time, on budget and on function and pay 50 percent less for their applications. In contrast, companies with a poor requirements capability had three times as many project failures as successes.
- Companies with excellent requirements processes reported that over 70 percent of their projects were successful.

Importantly, these benefits apply across all software development approaches. So, if your team is already experiencing the benefits of agile development, it will experience even more benefits by adding an effective BA capability.

2.9 THE AGILE DIAGNOSIS

In the late 1990s and early 2000s, as BA was being established, another hypothesis for project failures was evolving. It proposed that the root cause was the waterfall development process widely in use at the time and that the solution was to replace it with an approach referred to as *agile* development. *Waterfall development* is a predictive, sequential process wherein each step is completed before the next begins: first, a comprehensive needs and requirements analysis is performed up front, then a complete design is created and is followed by implementation, testing, and finally, release into production. Plans are made up front on the basis of requirements and other factors, and success is measured

on the basis of conformance to the plan. When projects failed, the waterfall answer was to enforce the process more strictly: do *even more* comprehensive analysis up front to ensure no critical requirements were missed and impose *stricter* controls on requirements changes. The problem was that no matter how hard companies tried, this approach didn't do much to move the needle. (Recall that disappointing 1995 *CHAOS Report*.)

The early agilists began to wonder if the problem wasn't with waterfall's assumptions of stability and predictability—noting that the opposite assumption is more often true: that software development often occurs under *volatile*, *unpredictable* conditions. Requirements often change after they've been baselined because of changing customer behaviors, market competition, new technology, new learning, and other factors. Instead of spending more time on upfront analysis and planning, the agilists proposed spending *less*. Instead of relying on a preset plan that was doomed to obsolescence the moment it was signed, they offered an adaptive approach that uses feedback to revise planning decisions throughout development to optimize *agility*—"the ability to create and respond to change in order to succeed in an uncertain and turbulent environment." The adjective *agile* can be used to describe a development process or organization that is responsive to change.

The agile toolkit draws from many sources. It includes workflow-management practices pulled from Kanban, guidelines for reducing waste from lean software development; events and techniques from the Scrum and XP frameworks; and scaled agile techniques from SAFe, DevOps, and continuous integration and continuous delivery (CI/CD). Later in this chapter, we look at all these foundational approaches and their impact on the practice of agile analysis.

2.10 THE AGILE TRACK RECORD

Agile development is no longer an interesting theory; it's accepted practice. The *14th Annual State of Agile Report*[14] found that "95% of respondents report their software organizations practice Agile development methods" and that 61 percent have been doing so for three or more years. Fifty-one percent of respondents reported that more than half or all of their teams were agile. Today, it is the presumed approach in most development organizations that I encounter.

[14] Digital.ai, *14th State of Agile Report*, 2020, 8, https://explore.digital.ai/state-of-agile/14th-annual-state-of-agile-report

Agile process was also cited as one of the top ten success factors for software development initiatives in the Standish Group's *CHAOS Report 2015*, as shown in Table 2.1. Table 2.2 summarizes the report's findings concerning the success of projects using agile versus waterfall approaches.

Table 2.2 CHAOS Resolution by Agile versus Waterfall

Size	Method	Successful	Challenged	Failed
All size projects	Agile	39%	52%	9%
	Waterfall	11%	60%	29%

Source: Adapted from Standish Group International, *CHAOS Report 2015* (Standish Group, 2015), 7.

As the table indicates, the use of an agile versus a waterfall method tripled the percentage of projects considered successful and reduced the percentage of failures to one-third that of the waterfall rate.

Agile development has also been shown to have a positive impact on productivity, with a 2015 QSM Associates study, commissioned by Rally Software, finding that "Agile projects experienced a 16% increase in productivity compared to [the] industry average."[15] It's worth noting, though, that this metric doesn't quite match the more extravagant claims made by Scrum's

authors when the framework was introduced: "When we say Scrum provides higher productivity, we often mean several orders of magnitude higher, i.e., several 100 percents higher."[16]

[15] Melinda Ballou, "As Agile Goes Mainstream, It's Time for Metrics," Rally Software Development, 2008, https://www.broadcom.com/products/software/agile-development/rally-software

[16] Ken Schwaber and Mike Beedle, *Agile Software Development with Scrum*. (Upper Saddle River, NJ: Prentice Hall, 2002), viii.

One reason for the discrepancy is that if you do a feature-by-feature comparison, agile development probably costs almost as much as waterfall because savings are mostly offset by rework involved in agile's trial-and-error approach. Its benefit, however, lies elsewhere: in the effectiveness of the investment. Agile methods help the business quickly determine *which* features to develop and then get that minimum marketable product (MMP) to market quickly. In that respect, agile has been highly successful: according to a QSMA study, "Agile projects are 37% faster to market than [the] industry average."[17]

[17] QSM Associates, *The Agile Impact Report, Proven Performance Metrics from the Agile Enterprise* (Boulder, CO: Rally Software, 2015), 4.

The evidence also suggests that agile development is leading to better products. In one study,[18] a full 78 percent of participants believed that using agile had led to higher stakeholder satisfaction.

[18] A *Dr. Dobbs Journal* survey quoted by Mike Cohn reported that *78 percent of participants believed that using agile had led to higher stakeholder satisfaction*. See Mike Cohn, *Succeeding with Agile* (Boston: Addison-Wesley, 2010), 16.

2.11 WHY AGILE TEAMS SHOULD INCLUDE AN EFFECTIVE BA COMPETENCY

With such a strong case for BA and agile development individually, you'd think the case for a competency that combined the two would be a slam dunk. The research bears this out. According to the *Business Analysis Benchmark*,[19] requirements maturity is closely correlated with success outcomes across all development approaches, including agile. Figure 2.3 illustrates its findings for organizations using agile development.

[19] Ellis, *Business Analysis Benchmark*, from a file provided by the author on research for the report.

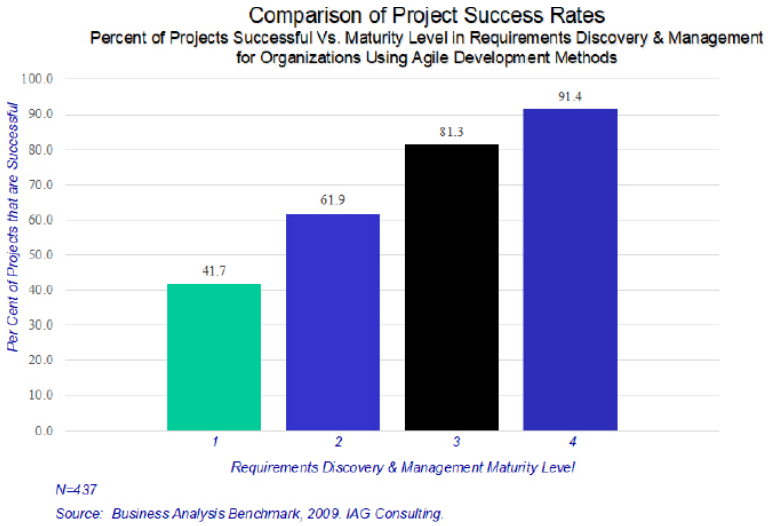

Figure 2.3 *Percentage of projects successful versus requirements maturity level for agile organizations.* N = *437. (Source: Keith Ellis,* Business Analysis Benchmark*—The Impact of Business Requirements on the Success of Technology Projects [Toronto, Canada: IAG, 2009] from a file provided by the author summarizing research for the* Business Analysis Benchmark.)

The figure clearly shows that if your agile team doesn't yet include a strong BA competency, it should introduce one. For example, if your team is currently at the *lowest* requirements maturity level, it stands to *more than double* its success rates by moving to the highest maturity level. A single-step increase from level 1 to level 2 results in a 49 percent increase in success rates.[20]

[20] Ellis, *Business Analysis Benchmark*.

Interestingly, the report also showed that the impact on success rates of requirements maturity level alone was far more significant than the effect of development methodology alone.[21] If you hold the maturity level constant, the *maximum* improvement for organizations using agile versus waterfall methods is an 11.6 percent increase in project success rates (at maturity level 2)—not nearly as high as the doubling of rates that are possible when improving requirements maturity levels alone.

[21] According to the report, the greatest improvement for organizations using agile versus waterfall methods occurs at maturity level 2, where there is an 11.6 percent increase in project success rates (from 54.77% to 61.94% successful). The benefit is down to 4.4 percent at level 3, while at the extremes (levels 1 and 4), agile success rates are actually slightly worse than those of waterfall.

2.12 CHAPTER SUMMARY

Here are the key points covered in this chapter:

- Agile development was launched with the publication of *The Agile Manifesto* in 2001.
- IIBA was founded in 2003.
- Research shows that increasing BA maturity levels results in improvements in percentage of projects delivered on time, percentage of projects on budget, and percentage of projects delivering all required features
- Research shows that an agile team that goes from a low requirements maturity level can double its success rates.

2.13 WHAT'S NEXT?

In this chapter, we looked at the business case for agile analysis and planning. In Chapter 3, "Fundamentals of Agile Analysis and Planning," we look at the fundamental concepts behind the competency.

Chapter 3. Fundamentals of Agile Analysis and Planning

This chapter presents a crash course in the fundamentals of agile analysis and planning, including the concepts, frameworks, practices, professional organizations, and terminology that underlie the competency. For readers who are new to business analysis (BA) or agile development, it provides a foundational understanding for the rest of the book, introducing the International Institute for Business Analysis (IIBA), Project Management Institute (PMI), the Agile Alliance, Scrum, lean software development, Kanban, Scaled Agile Framework (SAFe), user stories, and more.

3.1 OBJECTIVES

This chapter will help you

- Understand the concepts, frameworks, and practices that underlie agile BA and the rest of this book.
- Understand how epics, features, and stories are used to represent requirements.
- Understand the two main approaches to agile planning: flow-based planning and timeboxed planning.
- Understand the Kanban, Scrum, and Extreme Programming (XP).
- Understand the agile analysis contribution to eliminating the seven wastes identified in lean software development and to supporting lean practice.

3.2 WHAT THE AGILE MANIFESTO MEANS FOR BUSINESS ANALYSIS

In the previous two chapters, we learned that BA is practiced differently in an agile context than in a traditional waterfall process. To fully comprehend the impact of agile development on BA, you have to begin at the beginning, with the Agile Manifesto and "12 Principles behind the Agile Manifesto."

3.2.1 Agile Manifesto

The *Agile Manifesto*[1] lists the four core values of agile development:

[1] Agile Alliance, *The Agile Manifesto*, 2001, https://www.agilealliance.org/agile101/the-agile-manifesto

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.

As the last line of the Manifesto indicates, the Manifesto is a declaration of *relative* values, not *absolutes*: its aim is to minimize "items on the right," such as processes and documentation, not to *prohibit* them. It's an important distinction because if you believe there should be *no* planning in agile, then agile analysis and planning would not be a competency—it would be an oxymoron. Fortunately for its practitioners (and for you, if you are one), that's not the case.

3.2.2 The Impact of the First Value on Analysis

The first core value of the Manifesto is, “Individuals and interactions over processes and tools.” In line with this value, agile requirements communication and management is lightweight, with a preference for direct, personal interactions with stakeholders and developers over formal procedures.

3.2.3 The Impact of the Second Value on Analysis

The second line of the Manifesto values “working software over comprehensive documentation.” If you’re a waterfall business analyst transitioning to agile, it means you’ll be writing less requirements documentation than you’re used to. Because the requirements are captured and communicated just before they’re needed, you don’t have to write as much down to avoid misunderstandings.

3.2.4 The Impact of the Third Value on Analysis

The third line of the Manifesto values “customer collaboration over contract negotiation.” If you’ve formerly been a waterfall business analyst, this value represents a fundamental shift in how you interact with solution providers and business stakeholders. Instead of providing comprehensive specifications up front that can be used as the basis for contractual commitments, your focus shifts to helping business and development collaborate throughout development on requirements discovery.

3.2.5 The Impact of the Fourth Value on Analysis

The fourth line of the Manifesto values the "ability to respond easily to change over following a plan." This value has a profound impact on requirements change management and implementation planning. If you're working on a waterfall project, the requirements and the implementation plan are frozen before implementation. Any changes after that trigger a formal change process. On an agile initiative, you don't freeze the requirements, and only a lightweight process is needed to change them.

3.3 WHAT THE TWELVE PRINCIPLES MEAN FOR BUSINESS ANALYSIS

The Agile Alliance backed up the Manifesto with the following twelve principles. I've used *italics* to highlight areas of particular interest to BA.

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

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. *Simplicity—the art of maximizing the amount of work not done—is essential.*
11. The best architectures, requirements, and designs emerge from *self-organizing teams.*
12. At regular intervals, *the team reflects on how to become more effective*, then tunes and adjusts its behavior accordingly.[2]

[2] Agile Alliance, "12 Principles behind the Agile Manifesto," 2001, https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto

Principle 1 emphasizes that delivery must have value. As an agile business analyst, you help the team follow this principle by helping ensure it is always occupied with work of the highest value. For example, you negotiate with developers to explore alternatives that deliver similar value at lower cost. Agile Principles 2, 4, and 6 revisit BA issues raised in the Manifesto: changing requirements, customer collaboration, and reliance on conversation over documentation.

Principle 3 is to deliver software frequently. While this principle doesn't refer explicitly to analysis, it has a profound impact on how you practice it: because of agile's frequent delivery cycles, you have to decompose requirements into units small enough to be developed within a short timeframe. This decomposition isn't trivial. It turns out to be one of the most challenging problems business analysts face when they begin practicing in an agile development environment. We deal at length with this issue in Chapter 13, "Story Preparation."

Principle 10 is "Simplicity—the art of maximizing the amount of work not done—is essential." As an agile business analyst, you practice this principle by minimizing upfront analysis and delaying all analysis until the *last responsible moment*—the point at which the cost of a further delay outweighs the benefits. In so doing, you reduce wasted effort on requirements that will later be changed or abandoned.

Principle 11 calls for *self-organizing teams* and shared team responsibility. With self-organization, the team decides who takes on which role depending on the work being done—and this can change from day to day. What that means for you is that you or any other team member with the necessary skills might be doing analysis one day and something else, such as testing, on another day.

I should point out, though, that this principle is rarely practiced in its pure form. In the absence of a formal leader, an informal leader often emerges. Rather than full self-organization, today's agile teams are following the leader-as-coach or servant-leader model. In this model, the leader's purpose is to serve team members and support their growth as persons.[3]

[3] Robert K. Greenleaf Center for Servant Leadership, "The Servant as Leader," 2016, https://www.greenleaf.org/what-is-servant-leadership

> For more on agile leadership, see Chapter 17, section 17.5.1, and Chapter 18, "Achieving Enterprise Agility."

Principle 12 calls for self-reflection by the team. As an agile business analyst, you support this principle during team retrospectives by providing team members with an external perspective on the team's performance, as seen by business stakeholders and customers.

3.4 PRACTICES, STANDARDS, AND FRAMEWORKS

Let's now turn from the Agile Manifesto and the twelve principles to other practices, standards, and frameworks that underlie agile analysis and planning.

3.4.1 Business Analysis Standards

Agile BA is an extension of a more general competency, business analysis. As a working professional in this area, you should understand the basic concepts, standards, and professional bodies that underlie and govern the discipline. The following is an overview of those items.

3.4.1.1 *BABOK* (IIBA) and *Business Analysis Practice Guide* (PMI)

BABOK: A Guide to the Business Analysis Body of Knowledge (BABOK Guide) is IIBA's guide to the BA discipline and a basis for its certification path. Its most popular professional certification is the *Certified Business Analysis Professional (CBAP)*. As an agile business analyst, you'll also be interested in *The Agile Extension to the BABOK Guide*, copublished by IIBA and the Agile Alliance, to address BA in the context of agile development.

Following IIBA, the PMI developed its own BA certification—the *PMI Professional in Business Analysis (PMI-PBI)*—and published its BA guide, the *Business Analysis Practice Guide*. The PMI also worked with the Agile Alliance to publish *The Agile Practice Guide*, focusing on the project leader and project team perspective. The *International Requirement Engineering Board (IREB)* offers the *Certified Professional for Requirements Engineering (CPRE)*, a certificate aimed at those working in requirements engineering, BA, and testing. (Full disclosure: I have acted as a reviewer for both IIBA's *BABOK Guide* and PMI's *Business Analysis Practice Guide*.)

IIBA and PMI guides break down the BA competency into domains of expertise. As you'll see, there's a lot of similarity between the two main guides.

3.4.1.2 *BABOK* Knowledge Areas

The third edition of *BABOK Guide* divides the BA competency into six knowledge areas (KAs):

1. *Business Analysis Planning and Monitoring:* Planning and coordinating analysis activities
2. *Elicitation and Collaboration:* Eliciting requirements from stakeholders and collaborating with them throughout the lifecycle
3. *Requirements Lifecycle Management:* Managing and tracing requirements and changes across the development lifecycle
4. *Strategy Analysis:* Identifying strategic business needs and developing strategies to address them
5. *Requirements Analysis and Design Definition:* Organizing, specifying, and modeling requirements
6. *Solution Evaluation:* Evaluating solution performance

3.4.1.3 PMI Guide Domains

PMI's *Business Analysis Practice Guide* specifies five BA domains:

Domain 1—Needs Assessment: Analyzing the problem or opportunity, goals, objectives, business case, organizational changes
Domain 2—Business Analysis Planning: Planning the Business Analysis Approach; analyze stakeholders
Domain 3—Requirements Elicitation and Analysis: Eliciting, analyzing and creating high-quality requirements in an iterative-incremental manner

Domain 4—Traceability and Monitoring: Managing requirements and requirements attributes; managing changes to requirements; tracing requirements
Domain 5—Solution Evaluation: Evaluating a solution

IIBA and PMI guides define several terms that underlie BA practice—a baseline of concepts for the practicing analyst. These are summarized next.

3.4.2 Requirements-Related Terminology

You'll be encountering various BA and requirements terms throughout this book. These terms are often used in different ways by different people. To avoid confusion, let's clarify how they will be used in this book.

3.4.2.1 Product Vision Statement

The **product vision statement** is a single sentence or phrase that describes the reason for creating the product.[4]

[4] Roman Pichler, "8 Tips for Creating a Compelling Product Vision," October 8, 2014, https://www.romanpichler.com/blog/tips-for-writing-compelling-product-vision

3.4.2.2 Business Goal

A **business goal** is something the business aspires to. Goals should be measurable and timebound. For example, a goal may relate to revenue, profitability, customer service, or customer retention.

Example business goals:

- Increase sales by 10 percent by the end of the year
- Grow membership by 8 percent by date X.

- Establish an enterprise-wide assortment planning capability by the end of the year.

Goals may be expressed for various levels. A business goal refers to a desired high-level outcome at the C-level (executive) or below.

3.4.2.3 Business Objective

A **business objective** is a lower-level outcome (i.e., below the enterprise level). Objectives should also be measurable and timebound.

Example business objectives:

- Provide the ability to promote services and benefits by the third quarter.
- Merchandising will drive top-line sales by 2.5 percent due to the ability to establish localized assortments, by X.

3.4.2.4 Requirements

A **requirement** is "a usable representation of a need. . . . The nature of the representation may be a document (or set of documents), but can vary widely depending on the circumstances."[5] Requirements should be measurable, testable, and timebound. *BABOK Guide* describes four requirements types: business requirements, stakeholder requirements, solution requirements, and transition requirements. I like to add another: user requirements. These are described in the next sections.

[5] International Institute of Business Analysis (IIBA), *BABOK v3: A Guide to the Business Analysis Body of Knowledge*, 3rd ed. (Toronto, Canada: IIBA, 2015), 15.

3.4.2.5 Business Requirements

Business requirements are high-level needs, goals, and objectives of the organization. They should express

what needs to happen, not *how*. They should be measurable and timebound.

Example business requirement:

> Provide a directory of services by the end of the quarter.

3.4.2.6 Stakeholder Requirements

Stakeholder requirements describe the "needs of stakeholders that must be met in order to achieve the business requirements."[6] The analysis is typically midlevel—not detailed enough for designing a solution.

[6] IIBA, *BABOK v3*, 16.

3.4.2.7 User Requirements

User requirements describe the user's experience *using* the product. User requirements should be solution-agnostic. They act as a bridge between higher-level stakeholder requirements and more detailed solution requirements. User stories and use cases are examples of ways to represent user requirements.

User requirements per se should not include design or detailed functional specifications, but you can use them as a frame upon which to hang those specifications. For example, a user story or use case for a search function may include a link to a performance requirement.

3.4.2.8 Solution Requirements

Solution requirements describe the characteristics of a solution in enough detail to design a solution. Two subcategories of solution requirements are functional requirements and nonfunctional requirements.

3.4.2.9 Functional Requirements

Functional requirements (FRs) describe the required behaviors of the solution. These are often expressed from

the perspective of the system.

Example FRs:

- The ability to search by product code, name, or category
- Compliance with specified business rules
- Expected system behavior when a precondition or system fails
- Data validation rules

3.4.2.10 Nonfunctional requirements

Nonfunctional requirements (NFRs) are so called because they do not directly specify functionality. Rather, they describe how well the system must perform in its intended environment and how it should respond to constraints on its behavior. NFRs are also called **service-level requirements (SLRs)**, **supplementary requirements**, and **quality requirements**. There are many different kinds of NFRs. Let's look at the common ones.

> **Scalability** is the ability of the solution to grow and accommodate increased demand or scope. For example, "The solution should be built to accommodate twice the current volume while still meeting performance requirements."
> **Reliability** describes the level of resiliency of the system. For example, "Mean time between failures (MTBF) shall not exceed X."
> **Availability** is the ability of the solution to perform its agreed function when required. It includes the specification of when and how the business can access the solution. For example, "System should be available 24/7 during year X."
> **Recoverability** describes how quickly the solution should be made available following an outage. For

example, “Time to restore service following an outage shall not exceed 1 hour.”

Capacity is the amount of data or services that the solution can manage. For example, “The system will maintain 5 years of historical data” and “The system will support 1,000 simultaneous users.”

Maintainability is the ability of the system to be changed or repaired easily and its ability to react to an expanded user base or new business units. For example, “The system will be configurable to support new business units without additional coding.”

Security is the ability to ensure the confidentiality, integrity, and availability of assets, information, and services. For example, “The system will authenticate all users via XYZ single sign-on (SSO).”

Data integrity is the ability to manage data consistency across the business. For example, “All APIs will follow architecture best practice to ensure data structures are aligned across consuming systems.”

Interoperability is the ease with which the solution may be used with other software systems. For example, “The solution will be compatible with X.”

Usability is the extent to which its intended users can use the solution with effectiveness, efficiency, and satisfaction. For example, “The solution shall prompt users to confirm financial transactions after successfully entering transaction details.”

Other examples of NFRs include **performance requirements**, **compliance** with industry standards, and **redundancy requirements**. The management of NFRs within an agile requirements process is discussed in Chapter 8, “Seeding the Backlog—Discovering and Grading Features.”

3.4.2.11 Transition Requirements

The last requirements type, **transition requirements**, are requirements that describe how the solution will be deployed and released into production—such as migration and training requirements.

3.4.2.12 Why Terminology Is (and Is Not) Important

The main benefit of knowing these requirements types is to help you ensure that someone (even if it's not you) is responsible for capturing them. That said, it's *not* worth spending precious hours debating whether a requirement belongs to one category or another; the important thing is to ensure it is included somewhere!

3.4.2.13 Trace Goals to Requirements

An important aspect of your responsibilities as an analyst is to trace requirements items and other BA information to other items. An item may be traced to an item of the same type (e.g., goal to goal, requirement to requirement), to items below it (e.g., objective to user requirement), and to items above it (e.g., functional requirement to business requirement).

Figure 3.1 is an example of a goal to grow membership and its relationships with the requirements that support that goal.

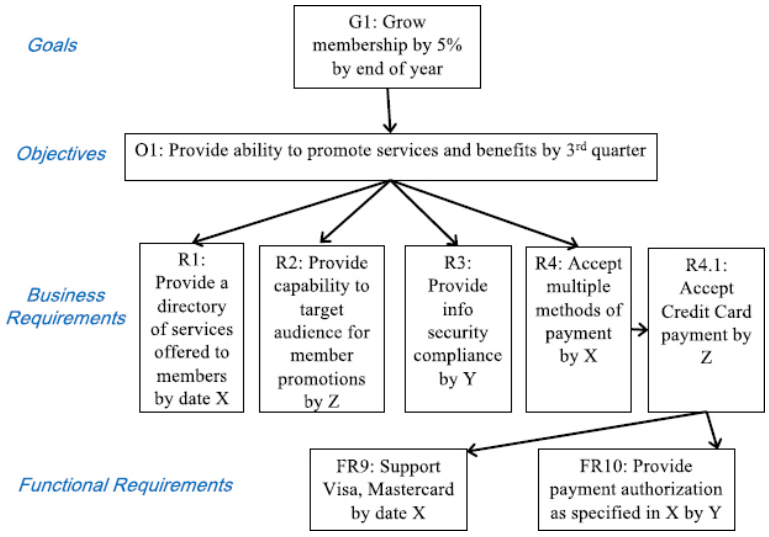

Figure 3.1 Mapping goals to requirements

3.4.2.14 Assumptions

An **assumption** is "an influencing factor that is believed to be true but has not been confirmed to be accurate, or that could be true now but may not be in the future."[7] In traditional waterfall BA, you would remove assumptions before development. In agile development, you treat assumptions that can't be verified up front as hypotheses to be tested through experimentation.

[7] IIBA, 441.

3.4.2.15 Constraint

A **constraint** is "an influencing factor that cannot be changed, and that places a limit or restriction on a possible solution or solution option."[8] Constraints may be business constraints—such as cost limits—or technical ones, such as constraints on supported platforms and programming languages.

[8] IIBA, 444.

3.4.2.16 Business Rule

A **business rule** is "a specific, practicable, testable directive that is under the control of the business and that serves as a criterion for guiding behavior, shaping judgments, or making decisions."[9]

[9] IIBA, 443.

Business rules are not "requirements" in and of themselves because they are inherent properties of the business: they would exist independently of a software development initiative. However, they become tied to the requirements when a solution is required to *comply* with them.

BABOK v3 differentiates between two kinds of business rules: behavioral and definitional. **Behavioral business**

rules are rules that the organization chooses to enforce as a matter of policy. An example is the rule that an insurance policy must be verified before an adjustment is performed on a claim. Behavioral business rules covered in this book include decision tables that proscribe the required response to various input conditions.

Definitional business rules describe business concepts and objects and their relationships with each other as well as calculations and derivation methods (e.g., how to derive a student's final grade). Definitional rules are expressed in the structural model—sometimes referred to as the *business domain model*, *class diagrams*, or *business-perspective entity-relationship diagram* (ERD).

3.4.2.17 Milestone

A **milestone** is a significant event or achievement.

Examples milestones:

- We will be using the planning tool for ski and winter sports items by the end of the first quarter.
- Use of the planning tool will spread to travel and kids' items by the end of the second quarter.

3.4.2.18 Requirements Units

In agile analysis, you decompose requirements into small items that can be developed within a short time—anywhere from a day or two to about eight days, depending on the practice. What you call these items depends on the framework you're using. Scrum refers to them as **product backlog items (PBIs)**; XP calls them **stories**; Kanban refers to them as **work items**. In *Use-Case 2.0*,[10] they're referred to as *use-case slices*. The most commonly used term is *story*.

[10] Ivar Jacobson, Ian Spence, and Kurt Bittner, *Use-Case 2.0: The Guide to Succeeding with Use Cases* (London: Ivar Jacobson International SA, 2011).

Requirements items—or work items—that are bigger than the story size limit are referred to by names that indicate their size. Although there is no single consensus on terminology, the usage that follows is common. It's the one I use in this book.

3.4.2.19 From Epics to Features to Stories

Figure 3.2 illustrates the relationship between epics, features, and stories. An epic is decomposed into features, and each feature is decomposed into stories.

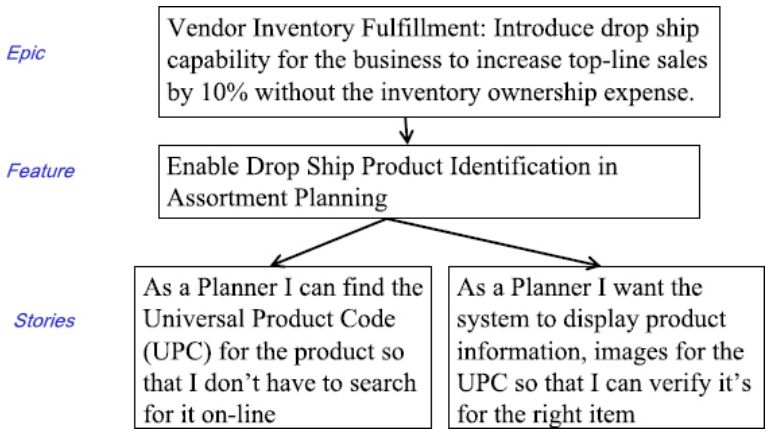

Figure 3.2 Decomposing an epic

3.4.2.20 Epics

In its original usage, *epic* simply meant a requirements item larger than a story. However, today it's common to reserve the term for the largest work items. The following definition is in line with that usage: an **epic** is a large, product-level work item that may require multiple teams over multiple quarters. The value delivered by an epic should represent a high-level capability of the product affecting a user or group of users. For example, an epic may deliver

- A top-level menu item on the user interface (e.g., customer management, suppliers, orders).
- A functional area of product.
- The expansion of an existing capability.

- A business process, or a step in a process or value stream.
- A project.
- Internal improvements.

3.4.2.21 Feature

A **feature** is a product-level work item that can be completed by one or more teams within one quarter or release cycle. A feature should represent a characteristic of a product that a user or group of users or customers care about. It is not a fine-grained requirement, but rather something significant enough to be highlighted to customers in marketing materials.

The above definition is in line with frameworks such as SAFe and *Use Case 2.0*. Note, though, that others may use the term differently. For example, in XP, a feature is a story; others use the term to refer to a set of user stories that should be released at the same time.

3.4.2.22 Story

Let's now return to the term *story* and define it more precisely. A **story** is a work item that *delivers value*, sized so that it can be implemented by a single team within a short period of time. The story size limit may vary from a day or two to about two weeks, depending on the approach and the team.

3.4.2.23 User Story

The recipient of the value delivered by a story—the actor who wants it—may be a user, a technical architect, sponsor, marketing representative, or software component. When the entity who wants the story is a user, the story is referred to as a *user story*.

A user story is "something a user wants,"[11] described from the user's perspective. Like any story, a user story must be small.

[11] Mike Cohn, "User Stories, Epics and Themes," Mountain Goat Software, October 24, 2011, https://www.mountaingoatsoftware.com/blog/stories-epics-and-themes

A User Story Is a Reminder to Have a Conversation

A User Story is not a requirement in the traditional sense. It is only a *reminder* to talk about a requirement; the details are expected to come out in the conversation itself. You may or may not be writing down the result of that conversation. For example, you might document it on a Note physically posted onto a Story card, document it elsewhere and referred to it from the Story, or not document the conversation at all and rely on verbal communication.

User Story Templates

You should express stories in the language of the customer. You don't have to follow any particular format, but one template has become popular because it captures the main points about a story. It's referred to as the **Role-Feature-Reason**, or **Connextra**, template, and it looks like this:

> "As a [user role] I want [function] so that [business value]."

3.4.2.24 Acceptance Criteria

Supply each feature and story with acceptance criteria that describe the conditions required for the stakeholder to consider the requirement "done." You can write acceptance criteria in an informal manner (e.g., "I can search by title"), or you can follow a template, such as behavior-driven development (BDD), discussed in Chapter 13, section 13.10.9.

3.4.2.25 Themes

Sometimes you're going to want to group stories for reasons other than that they are all components of a feature or epic. In that case, you can group them into

themes. A **theme** is a grouping that may be based on any criterion. Typically, a theme represents a business objective communicated to customers (e.g., image recognition, two-factor authentication).

3.4.2.26 Story Estimation

One way to estimate a story is to use **story points**, a relative measure of effort. Alternatively, you can use time-based units, such as **ideal developer days** (IDDs). **Velocity** refers to the number of these units a team delivers within a given amount of time (e.g., within a two-week iteration).

Points and IDDs are usually allocated using the Fibonacci series: 0, 1, 1, 2, 3, 5, 8, 13, and so on. At the start of a new venture, a point is typically set to be roughly equal to one IDD. The maximum story points for a story is often set to 8, representing the number of available days in a two-week period, or iteration. (Two of the ten workdays are assumed to be lost to planning and other work not directly related to feature implementation.)

3.4.2.27 Feature Lifecycle

As discussed earlier, a feature is a large requirement item, big enough to matter to customers. In agile processes, each feature goes through something like its own waterfall lifecycle—first analyzed, then coded, tested, and deployed. Unlike waterfall, though, this lifecycle is "porous" in that there is overlap between the steps. Figure 3.3 illustrates the lifecycle of a feature, using the feature statuses proposed by Jacobson.[12]

[12] Ivar Jacobson, "Feature State Cards," Ivar Jacobson International, https://s3.amazonaws.com/ss-usa/companies/MzawMDE1MTI2AwA/uploads/Feature_State_Cards_3_9.pdf

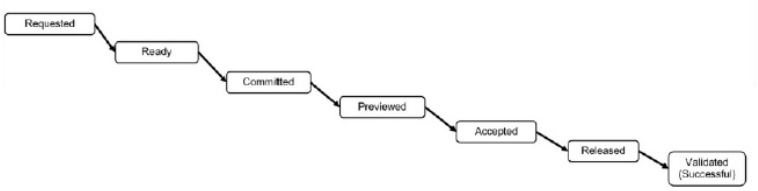

Figure 3.3 Feature lifecycle

The figure illustrates the following lifecycle:

- A feature begins as a *requested* item, at which time it is in the backlog but is not yet active, sequenced, or estimated.
- The feature is considered *ready* once it has been prepared well enough for implementation to begin. If the organization is practicing quarterly planning, a feature must be in the ready state before you can include it in the quarterly plan.
- A feature is considered *committed* when it has been assigned to a lead team, and all teams required for its implementation have committed to it.
- A feature is considered to be *previewed* once it has been discussed. This preview discussion typically begins about one or two iterations before implementation. The feature is then decomposed into stories.
- Once enough of a feature's functionality has been implemented to satisfy the customer, the feature is deemed *accepted*. Accepted doesn't necessarily mean that *all* the feature's stories have been implemented—only that *enough* have been implemented for the feature to be releasable to the market.
- Once the feature is accepted, it may quickly transition to the deployed state (presuming continuous delivery capabilities are available), or its deployment may be delayed so it can be released with other features at the end of the quarter.

Once the feature has been deployed into production and delivered to the user, the feature is considered *released*. Once a feature's value has been proven in the marketplace, the feature moves to the *validated* state.

3.4.3 Agile Planning

Agile planning is particularly challenging because it has to accommodate late changes. The way you do that is by delaying planning decisions as much as possible so that they can be made on the basis of the most current information possible. There are two broad agile planning approaches for carrying this out: the flow-based approach (e.g., Kanban), in which commitments are made just before the implementation of each item, and timeboxed planning (e.g., Scrum), in which commitments are made earlier, at the start of the planning period. As we'll see, a good rule of thumb is to use a mix of the two approaches: flow-based planning for smaller, customer-driven work items that affect one or two teams and timeboxed planning for strategic, product-level initiatives. Let's explore each of these options further.

3.4.3.1 Flow-Based Planning

In single-item flow-based planning (e.g., Kanban), you commit to features and stories on an item-by-item basis just before implementation. This approach provides the most adaptability because the customer can reprioritize a requirement right up until its implementation. It's the preferred approach for customer-driven work items, especially when they involve local changes that can be accomplished by one or two teams (e.g., a story to change the options for viewing a user's messages). This approach is also recommended when developing novel products because it accelerates learning.

The downside is that if you use only this approach, you can find yourself in a situation where teams are rarely

free at the same time to take on large product-level work items, since each one is working to the beat of its own drummer. As a result, strategic initiatives can end up sitting on the shelf for long periods. For that reason, most large organizations should reserve some of their teams' budgets for timeboxed planning, as discussed next.

3.4.3.2 Timeboxed Planning

In timeboxed planning (e.g., Scrum, SAFe), all of the work items for a given period (the timebox) are committed to up front at the same time. It's common to define a long timebox of about a quarter and smaller timeboxes within it of about one to two weeks, referred to as iterations (or sprints in Scrum). Typically, this plays out as follows: All teams within a group meet at the start of each quarter to plan and commit to multiteam features over the quarter. They meet again at the start of each iteration to commit to the team-level stories that they'll be working on.

This is the preferred approach for large, strategic initiatives involving multiple teams because it ensures that all teams are free to take on large work items at the same point in time. An example of a product-wide initiative that benefits from this approach is the implementation of a consistent user experience across the product. Frameworks that use the timeboxed approach include Scrum, XP, and SAFe. The Scrum framework includes sprint planning but not quarterly planning.

I don't recommend quarterly planning, however, for localized, customer-driven features. The reason is that it requires commitment at the start of the quarter based on the information available *at that time* rather than the more current information available just prior to implementation. It's true that even with this approach, all parties are *supposed* to agree that upfront

commitments to features are provisional, not absolute—and that last-minute changes *should* be accommodated. In practice, though, teams tend to be reluctant to make changes once a plan is written down. Consequently, if the customer wants to add a new feature based on learning that occurred partway through the quarter, the feature often has to wait until the start of the next quarter. Even worse, if the learning occurs in the latter half of a quarter, the new feature won't be included until the following two quarters because the upcoming quarter will already have been planned.

3.4.3.3 Why a Mixed Planning Approach Is Preferred

As noted earlier, organizations that use flow-based (Kanban) planning for all of their work can find that big initiatives end up sitting on the back burner. Their products also tend to become disjointed because each team is focusing only on localized issues. On the other hand, organizations that rely too heavily on timeboxed planning suffer from inordinately long learning cycles and decision making that isn't based on the most current knowledge. For these reasons, as a rule of thumb, most large development organizations do best with a mix of the two approaches.

> For more on agile planning in a scaled organization, see Chapter 17, section 17.8.

3.4.3.4 What About the Delivery Cadence?

Everything we've been talking about relates specifically to the intake side of the equation: the acceptance of requirements items into an implementation plan. The considerations are very different with respect to the output side—the delivery of releasable items to the customer. On that score, the recommendation is more straightforward: wherever possible, use the flow-based approach to deliver requirements items. With this approach, each work item is integrated and tested as

soon as it's done, so that it is in a releasable state should the customer choose to deploy it immediately to the market. To accomplish this at scale with confidence, the organization needs to adopt continuous integration and continuous delivery (CI/CD) and DevOps practices. If these capabilities are not present, it may be necessary to wait until the end of the iteration, quarter, or release cycle to integrate and test completed stories.

> For more on DevOps, see Chapter 17, section 17.4.2.1

3.4.4 Agile Frameworks

Let's now turn to the agile frameworks you need to know about and their impact on analysis and planning. Keep in mind that it's most likely that your organization will be using a hybrid of best practices from various sources as opposed to strictly adhering to a single framework. For example, it might use Kanban's approach to flow-based planning, add in Scrum tools such as daily standups and burndown charts to monitor progress, and use XP stories to specify work items.

3.4.4.1 Kanban

Kanban was developed in the 1950s for automobile manufacturing and is now widely used for software development. For example, Kanban is listed as one of the twenty-two tools of lean software development. Kanban doesn't include the concept of fixed iterations, since items progress on a piece-by-piece basis.

One of the important insights of Kanban is that simple rules and visual cues can effectively govern sophisticated behavior. The Kanban process plays out as follows: You represent each work item (e.g., a feature or story) as a card on a Kanban Board. You create a column, or *queue*, for each status that needs to be differentiated and tracked. Then you move the cards across the columns

from left to right, as their status changes. You move them up or down within a column to indicate relative priority.

The flow across the columns is governed by a few simple rules: A work-in-progress (WIP) limit is set for each queue. Whenever the number of items in a queue falls below the WIP limit, a developer associated with the queue pulls the topmost work item from the queue to its left. Once the number of items in a queue reaches the WIP limit, no more items may be added to the WIP column until an item has left the queue.

3.4.4.2 Scrum

Scrum was created by Ken Schwaber and Jeff Sutherland and was presented at the OOPSLA (Object-Oriented Programming, Systems, Languages, and Applications) conference in 1995. Its creators described Scrum as a "framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value."[13] The approach is defined in the *Scrum Guide*, which you can download for free at Scrum.org.

[13] Ken Schwaber and Jeff Sutherland, *The Scrum Guide: The Definitive Guide to Scrum—The Rules of the Game*, Scrumguides.org, 2017, 3, https://www.scrumguides.org

Scrum is very popular in IT departments of mainstream businesses. Its popularity is likely due to a confluence of factors:

- The Scrum process provides a structure for planning and status reporting.
- Scrum sets a development cadence (one or two weeks to one month) that many mainstream organizations can comfortably live with.
- Scrum has a well-marketed certification process.

Scrum is out of favor in some agile quarters because items are committed at the start of an iteration (referred to as a *sprint*) rather than later on the basis of more current priorities. Another complaint is that the framework is seen as overly accepting of longer delivery cycles rather than viewing CD as the norm.

Despite these objections, Scrum is an excellent choice for large product-level work items because it results in teams who are free at the same time (the start of a sprint) to make commitments. If you choose Scrum, use timeboxing to govern the flow of items going *into* an iteration (sprint)—but not to delay the flow of items through integration and testing at the tail end of their lifecycle.

Even if your organization does not explicitly use the Scrum framework, it likely uses some of its elements and ceremonies, such as the daily standup, product backlog, and retrospectives. These are summarized next.

The Sprint

Scrum uses the term **sprint** to refer to an iteration. (In this book, unless referring specifically to Scrum, I use the more generic term *iteration*.) A sprint is a period of up to one month during which a potentially releasable increment is created. Typical sprint durations are one or two weeks. An **increment** is an updated version of the software containing all the enhancements implemented during the sprint and including enhancement made in all previous sprints.

Note that an increment does not *have* to be released to the general market by the end of every sprint; it just has to be potentially *releasable*. For example, it might be released into a staging environment or released into production but with the changes concealed from the general user population until a later date. You might decide to delay exposure to the user because the feature

is not yet competitive enough for general release or because a completed feature is not marketable until other stories in the value stream are also available.

The Product Backlog

In Scrum, requirements reside in a product backlog. The **product backlog** is an "ordered list of everything that might be needed in the product and is the single source of requirements for any changes to be made to the product. . . . The Product Backlog lists all features, functions, requirements, enhancements, and fixes that constitute the changes to be made to the product in future releases."[14]

[14] Schwaber and Sutherland, *The Scrum Guide*, 12.

Each requirement item in the backlog is referred to as a **product backlog item (PBI)**. Scrum places few restrictions on what a PBI can be or how it should to be documented. The product backlog is a dynamic artifact, with requirements items added, dropped, re-sequenced, and refined throughout the development cycle.

Scrum and the Business Analysis Competency

Scrum does not allow designated roles for team members because this would run counter to its support for self-organizing teams. However, it does refer to required competencies. Scrum calls out BA as one of the "particular *domains* that need to be addressed"[15] within the development team.

[15] Schwaber and Sutherland, 6.

The Product Owner and Business Analysis

The product owner (PO) is the sole person responsible for managing and prioritizing the product backlog. The first responsibility, backlog management, lies within the scope of BA. The second responsibility, to prioritize, lies outside it.

The ScrumMaster and Business Analysis

The focus of the ScrumMaster role is to ensure that the Scrum guidelines are carried out and that impediments are removed from the team's path. Responsibilities that overlap with BA include communicating requirements to the team. We look at these more closely in the section on BA roles in agile.

Transparency

In place of control through comprehensive planning, Scrum offers control through **transparency**.[16] Examples of transparency in Scrum include its requirement that working code be demonstrated every sprint and that the progress of PBIs be visible to the business and team members.

[16] Schwaber and Sutherland, 4.

> Recently I learned that a group of stakeholders were told by developers that they wouldn't be able to see the product backlog because "you don't use one in agile." I later found out that there *was* a backlog but that developers were hiding it from stakeholders to avoid being harassed about timelines. This is a prime example of how *not* to practice transparency.

Definition of Done

The **definition of done (DoD)** is a set of conditions that must be satisfied for a requirements unit to be considered completely implemented—or "done." The DoD acts as a minimum set of conditions that *every* PBI must comply with; in addition, each individual PBI also has its own acceptance criteria. Common DoDs include that the requirement be discussed with users, coded, tested and that the code be checked in, refactored and integrated.

Readiness

Scrum requires that PBIs be *ready* prior to their selection in sprint planning. In Scrum, **ready** means that the PBI is sufficiently prepared to be immediately

worked on and completed by the team within a single sprint.

Many organizations go further and define formal **definitions of ready (DoRs)** to specify the conditions that must be true before teams commit to a work item. You can define a DoR at the feature level to specify what must be true before a feature is accepted into the plan and at the story level to clarify what must be true before a team commits to a story.

If the organization is using Scrum with quarterly planning, verify all features under consideration against the feature DoR before quarterly planning; check stories against the story DoR before iteration (sprint) planning. If the organization is using flow-based planning (e.g., Kanban), check each feature and story individually against the corresponding DoR as it comes up in the backlog.

Sprint Goal

The sprint goal is a statement of "an objective that will be met within the Sprint through the implementation of the Product Backlog [providing] guidance to the Development Team on why it is building the Increment."[17] As an analyst, you help the customer and development team collaborate to craft the sprint goal. If the goal is in jeopardy at any time once the sprint is underway, you work with the business and developers to explore options to deliver it in other ways (e.g., by delaying all or part of a PBI to a later sprint).

[17] Schwaber and Sutherland, 9.

Backlog Refinement

The work to prepare PBIs for implementation is referred to, in Scrum, as **backlog refinement**. In the rest of this book, however, unless referring to Scrum, I use the more generic term **preparation** (e.g., backlog preparation, feature preparation, story preparation). Scrum defines

backlog refinement as "the act of adding detail, estimates, and order to items in the Product Backlog."[18] It recommends that a maximum of 10 percent of the team's time be spent on backlog refinement.[19]

[18] Schwaber and Sutherland, 13.
[19] Schwaber and Sutherland, 15.

Refinement in Scrum is an ongoing, collaborative process between the PO and the development team. Refinement activities include all preparation work needed for the PBI to be actionable—such as removal of dependencies and initial design work. Much of the work is analysis related: incrementally adding granularity to PBI requirements and acceptance criteria as implementation nears.

The goal of refinement in Scrum is to prepare PBIs in the backlog enough that they are "ready" to be included in sprint planning. (If you're using a Kanban approach, you also prepare work items. However, you do so individually, as each individual item comes up in the backlog, as opposed to preparing a group of items for the sprint.)

Sprint Planning

A **sprint planning** event is held before each sprint to determine what will be accomplished during the upcoming sprint and how. In the rest of this book, I use the more generic term **iteration planning** to refer to the activity. During the event, the team creates a *sprint backlog*—a plan for what will be accomplished in the next sprint and how it will be accomplished. The *what* is expressed in the sprint goal and a set of requirements items (PBIs) for the sprint. The *how* is expressed in a list of **developer tasks** to implement those items—with owners and estimates assigned. Developer tasks include *all* work done on PBIs—not only design and coding but also analysis and testing. For example, during the event, you plan for any remaining analysis work on PBIs

planned for the current sprint as well as for analysis work on upcoming PBIs to make them ready for a future sprint.

Daily Scrum, or Daily Standup

The **daily scrum**, commonly called the **daily standup**, is a short meeting to plan the day's work. It should be held at the same time every day—often occurring in the morning. All team members meet to discuss what they have done toward the sprint goal since the last meeting, what they will do until the next meeting, and any impediments they foresee. As an analyst, you update the team on analysis activities related to the work that is being implemented. *The Scrum Guide* is strict about applying a 15-minute time limit to the standup, but you should extend the meeting if the update requires a longer discussion.

Keep in mind that the standup is only meant for updates; hold a follow-up meeting, if necessary, to resolve issues that have arisen during the standup. In contrast to the standup, only those members who can contribute to a solution should attend the follow-up.

> For more on the daily standup, see Chapter 17, section 17.8.

Sprint Review and Retrospective

At the end of every sprint, a **sprint review** is held to review and demonstrate the work that was done and to readjust the product backlog as necessary.

A **sprint retrospective** is held afterwards as an internal meeting to support continual improvement. As the business analyst, you're an important attendee at the retrospective, contributing an external perspective on the team's performance.

3.4.4.3 Extreme Programming

XP is an iterative-incremental framework for software development developed in the 1990s by Kent Beck and described in his book, *Extreme Programming Explained*.[20] As a timeboxed framework, XP shares many features with Scrum. Differences include its greater focus on technical issues and its short iterations of approximately one week. In addition, while Scrum does not address planning beyond the sprint, XP includes quarterly planning.

[20] Kent Beck and Cynthia Andres, *Extreme Programming Explained: Embrace Change*, 2nd ed., The XP Series (Boston: Addison-Wesley, 2005).

Primary practices of XP include the following:

- Pair programming—two programmers working together
- Sitting together as a team in an open space without cubicles
- Energized work—sustainable work hours, with avoidance of regular overtime
- Slack—allowance for nonstructured time
- Test-first programming—writing the test before the code
- Continuous integration (CI)
- Incremental design—designing for current needs and growing the design as the business grows
- Automated testing

Many of these primary practices are widely accepted today, even in organizations that haven't adopted XP as their framework.

A number of XP's techniques relate directly to agile analysis and planning. These include

- The practice of organizing small, vertically integrated teams (referred to as **whole teams**) of

no more than twelve members.[21]

[21] Beck and Andres, *Extreme Programming Explained*, 9. The authors reference Malcolm Gladwell's *The Tipping Point*, which sets twelve as the maximum number of people who can comfortably interact with each other in a day.

- The use of stories as a basic unit for requirements management.
- **Informative workspaces** whereby an observer can get an idea of the project in fifteen seconds. Examples of this practice in this book include the use of the product canvas, story maps, and the GO (goal-oriented) product roadmap.
- XP's **Planning Game** guidelines that set out the terms for quarterly and iteration planning.

3.4.4.4 Rational Unified Process (RUP)

The Rational Unified Process (RUP) is an iterative-incremental process developed by the Rational Corporation in the late 1990s and is based on the objectory process. Objectory was created by Ivar Jacobsen in the late 1980s.

In the RUP, analysis, design, coding, and testing occur during all phases of development—although the relative effort varies over time. The RUP lifecycle begins with a short **inception** phase during which the business case and high-level use-case model are created, and prototypes and proofs of concept are built. Next is an **elaboration** phase during which the majority (often up to 80%) of the requirements are analyzed up front, and the architecture is tested and established. In this phase, most of the use-case scenarios are specified. It's followed by a **construction** phase during which the remaining analysis, design, and implementation activities occur in an iterative, incremental fashion over a number of iterations. Finally, a **transition** phase occurs during which the solution is migrated to production.

3.4.4.5 Use Cases

The use-case approach was developed by Ivar Jacobson and described in his 1992 book, *Object-Oriented Software Engineering*.[22] Use cases have since become a popular tool for analyzing requirements—absorbed as part of the Unified Modeling Language (UML) standard. They are the basis for many widely used iterative-incremental software development lifecycle frameworks and methods, such as RUP.

[22] Ivar Jacobson, *Object-Oriented Software Engineering: A Use-Case Driven Approach* (Reading, MA: Addison-Wesley, 1992).

A use case is defined as follows: "A use case is all the ways of using a system to achieve a particular goal for a particular user. Taken together, the set of all the use cases gives you all of the useful ways to use the system and illustrates the value that it will provide."[23] A simple way to think of a use case is that it represents a user task—a unit of work, or goal, that a user expects to accomplish through an interaction with the system.

[23] Jacobson et al., *Use-Case 2.0*, 4.

A **use-case narrative** (also referred to as a use-case *specification*) is a "description of a use case that tells the story of how the system and its actors work together to achieve a particular goal. It includes a sequence of actions (including variants) that a system and its actors can perform to achieve a goal."[24] The narrative may be specified at various levels of granularity depending on circumstances—from briefly described (where only the user and goal are specified) to fully described (where a comprehensive description is provided).

[24] Jacobson et al., 51.

An entire use case, containing *all the ways* to achieve a goal, usually takes longer than a single iteration to implement. Consequently, rather than assigning an entire use case to an iteration, you assign specified scenarios of the use case. A **use-case scenario** is a

complete pass through a use case, describing one way it might play out.

Jacobson adapted the use-case approach for agile development with *Use Case 2.0*. One of its additions to the original approach was the invention of the **use-case slice** as the primary atomic requirements unit for planning. A use-case slice is "one or more stories selected from a use case to form a work item that is of clear value to the customer."[25] To be suitable for inclusion in iteration planning, a use-case slice must be small enough to be implemented within a single iteration.

[25] Jacobson et al., 15.

Use Cases versus User Stories

Any requirements unit that would qualify as a small-sized use-case slice would also qualify as a user story in XP's approach. In an ideal world, I would prefer to see the use-case slice used as the atomic agile requirements unit because it always provides a usage context (the use case) for a requirement. Another reason is that the approach provides a single, unified framework for organizing requirements, not only during development but after development as well—something for which user stories are not well suited. However, in the real world—and this book—I most often use user stories during development because they are much more widely used by agile teams and better supported by agile tools.

Post-implementation, the better requirements unit to use for **persistent** (long-lasting) documentation is the use case because user stories are too small to be used as an effective reference. For example, consider that one user story in a backlog might allow a shopper to order gift-wrapping, another might add the option for drone delivery, and a third might have nothing to do with shopping. For reference purposes, it's useful to gather all the stories in the backlog related to purchasing under

one roof. That's precisely what happens when you consolidate them into a *Purchase items* use case.

3.4.4.6 Lean Thinking

Lean thinking was developed by Taiichi Ohno and Shigeo Shingo at Toyota in the 1950s. It is the source of many guidelines that have found their way into agile practice, such as small batch sizes, just-in-time production, and short cycles.

Lean thinking underlies the agile principle, "Simplicity—the art of maximizing the amount of work not done—is essential."[26] In lean thinking, this guidance is expressed in its aim of reducing *muda* (waste) in the value stream.

[26] Agile Alliance, "12 Principles."

3.4.4.7 Lean Software Development

Lean software development, first coined in 1992, is an approach pioneered by Bob Charette for achieving perfection by eliminating waste.[27] It has been developed further into various branches by contributors such as Tom and Mary Poppendieck, who laid out their framework in 2003 in their book *Lean Software Development: An Agile Toolkit*.[28] The toolkit includes seven lean principles and twenty-two **thinking tools**, many of which have a direct bearing on BA and the guidance in this book.

[27] David J. Anderson, "Lean Software Development," 2015, https://msdn.microsoft.com/en-us/library/hh533841(v=vs.120).aspx

[28] Mary Poppendieck and Tom Poppendieck, *Lean Software Development—An Agile Toolkit* (Boston: Addison-Wesley, 2003).

The first and central principle is to *eliminate waste*. In lean software development, waste is "anything that does not add value to a product, value as perceived by a customer."[29]

[29] Poppendieck and Poppendieck, *Lean Software Development*, xxv.

The Seven Wastes

Lean software development recognizes seven "wastes":

1. Partially done work
2. Extra processes
3. Extra features
4. Task switching
5. Waiting
6. Motion
7. Defects

Let's examine these wastes and how you can reduce them through effective agile analysis and planning.

1. Partially Done Work[30] Work that has begun but that may never be completed by the time it's needed is considered wasteful. The same is true if the partially done work will be out of date by the time it's needed. In agile analysis, this waste is minimized using a just-in-time approach—delaying the analysis until just before implementation, when we can be sure it will be current and when the immediacy of the upcoming development means that stakeholders are more likely to give the item the attention it needs.

[30] Poppendieck and Poppendieck, 5.

2. Extra Processes The waste of extra processes[31] is minimized in agile analysis through the use of low-overhead processes for managing and changing requirements, the elimination or reduction of requirements sign-offs, and reliance on direct communication over extensive documentation to communicate requirements.

[31] Poppendieck and Poppendieck, 5.

3. Extra Features Extra features[32] represent the wasted effort to develop features that are never used. Agile analysis reduces this waste by using a data-informed approach to decide which requirements to invest in and which to drop. Waste in the form of extra features is also reduced through just-in-time analysis, since it ensures that analysis effort is spent only on requirements that have current value and that prioritization decisions are based on up-to-date information.

[32] Poppendieck and Poppendieck, 6.

4. Task Switching Task switching represents the effort wasted when a worker switches back and forth between different work items. In agile analysis, this waste is minimized by defining small stories that can be developed one after the other rather than concurrently and by keeping work-in-progress limits low.

5. Waiting Waiting represents waste caused when work cannot proceed until something else occurs. Agile business analysts reduce waiting by defining minimally dependent stories and by preparing features and stories before implementation so that all dependencies are identified and planned for in advance.

6. Motion Motion[33] represents the movement of people and artifacts. Agile analysis reduces motion through the use of *information radiators*—publicly posted displays of information—to communicate requirements information freely to the team and by eliminating or reducing handoffs and sign-offs. Information radiators are discussed further in the next section.

[33] Poppendieck and Poppendieck, 7.

7. Defects Defects are reduced in agile analysis through the early definition of acceptance criteria—ensuring that the developers know what test scenarios to look out for before they begin. Wasted effort due to defects is also

reduced by decomposing requirements into small units that can be developed and tested quickly, thereby exposing bugs early when they are easiest to fix.

Tools of Lean Software Development

The Poppendiecks' approach to lean software development contains twenty-two "thinking tools." In the following sections, we focus on some of the tools that relate most strongly to agile analysis and planning.

Tool 1: Seeing Waste The first step in removing waste is to see it. In the previous discussion of the seven wastes, you learned how agile BA helps the team see each of these kinds of waste.

Tool 2: Value Stream Mapping Value stream mapping is the charting of steps in an end-to-end process in order to optimize it (e.g., by reducing wait times). As an agile business analyst, you perform value stream mapping on *operational* business processes in order to improve them. You may also perform value stream mapping on the *development* process itself in order to identify bottlenecks and inefficiencies in the development lifecycle.

Tool 3: Feedback As an agile business analyst, you support fast feedback by defining short, valuable stories that can be implemented and tested out within a single iteration.

Tool 4: Iterations As an agile business analyst, you support the guidance "to implement a coherent set of features in each iteration"[34] by using visual tools like customer journeys, process maps, and story maps to ensure the stories in each iteration fit well together.

[34] Poppendieck and Poppendieck, 29.

Tool 7: Options Thinking Options thinking means keeping one's options open by delaying irreversible

decisions until uncertainty is reduced and by investigating alternatives (e.g., co-developing two options). You practice options thinking by facilitating the exploration of alternative ways to deliver value and delaying requirements analysis and commitment to the last responsible moment.

Tool 8: Last Responsible Moment (LRM) Delay decisions to the LRM, defined as "the instant in which the cost of the delay of a decision surpasses the benefit of delay, or the moment when failing to take a decision eliminates an important alternative."[35] The agile business analyst practices LRM by minimizing upfront analysis and delaying detailed analysis until close to implementation—when any further delay would incur unacceptable costs.

[35] Lean Construction Institute, "LCI Lean Project Delivery Glossary," 2017, https://www.leanconstruction.org/media/docs/LCI_Glossary.pdf

Tool 10: Pull Systems Lean development favors pull systems in which workers determine what work to do next rather than rely on centralized command-and-control systems that don't respond well to change. Lean uses signs (placards), known as Kanbans, to represent work items. As a work item advances from station to station, so, too, does its Kanban card. Workers at each station make their own decisions about advancing work items based on simple rules. In agile analysis, you use this approach to manage the progression of features and user stories across their development lifecycle.

Tool 12: Cost of Delay The cost of delay is the difference in total profit between a project that is started immediately and the same project begun after a delay. Cost of delay is an especially significant concern when profitability depends on being first to market. In agile analysis, you determine the cost of delay of PBIs and use

it to help determine an item's priority sequence in the backlog.

> For more on the cost of delay, see Chapter 6, "Preparing the Process," section 6.5.4.4.

Tool 20: Testing As an agile business analyst, you support testing by following acceptance test–driven development practice—negotiating and crafting acceptance criteria for features and stories before their implementation and teaching others to do so. At the feature level, you specify acceptance criteria for user acceptance testing (UAT) and describe how much of the feature must be implemented before the feature as a whole is considered done. You also specify acceptance criteria at the story level to determine the tests that an individual story must pass in order to be considered done.

Principles of Lean Software Development

Lean software development also includes a set of principles. We've already examined the impacts of many of these have on BA, such as to eliminate waste and to decide as late as possible, so I focus here on principles that provide additional guidance.

Deliver as Fast as Possible. This principle advises that features be delivered to customers immediately, or as close to immediately as possible. Speed is highly valued because it enables many other lean principles, such as "decide as late as possible." As an agile business analyst, you contribute to this principle by specifying small Stories that can be implemented and delivered quickly. At the product level, you contribute by helping the customer and team define a minimum marketable product (MMP) that can be released rapidly into the market. For the whole team and organization, it means using CI/CD, and DevOps practices to provide the capacity to deliver changes reliably as soon as the business wants them released.

Build Integrity In. This principle means that the software, architecture, and interfaces should possess coherence, with all parts working together cohesively as a whole. As an agile business analyst, you actualize this principle using story maps to ensure that the stories implemented within each iteration add up to useful functionality.

See the Whole. This principle states that teams should focus on optimizing the whole product, not just the aspects of the product within their areas of expertise. You support this principle by organizing teams around value to the business (vs. competencies), by supporting team ownership of commitments, and through such practices as the use of a single product backlog and product owner across all teams working on a product.

Information Radiators

An **information radiator** is "a display posted in a place where people can see it as they work or walk by. It shows readers information they care about without having to ask anyone a question."[36] In lean terms, information radiators lessen the *waste* of *motion* by reducing handoffs and sign-offs.

[36] Alistair Cockburn, *Crystal Clear: A Human-Powered Methodology for Small Teams*, (Boston: Addison-Wesley, 2004), 54.

The audience for information radiators is not only internal—it's for those outside the teams as well, to provide answers to questions without having to interrupt those doing the work.

Cockburn describes the following attributes of a good information radiator:

- Legible at a distance
- Easy to understand at a glance
- Updated continually

- Posted in a public place
- Is (ideally) a physical artifact (e.g., flipchart or whiteboard). Large display monitors may also be used (e.g., to communicate status information).

When working with non-colocated teams, you need to slightly adapt the guidance about a physical artifact. In such cases, members and stakeholders should first assemble in the real world for planning purposes if possible. During this phase, they use physical flipcharts. After they disperse, the flipchart content is transferred to digital form and posted online so that all interested parties have access to the same version.

As an agile business analyst and planner, you use information radiators extensively. We'll be using the following radiators in this book:

- Product portrait (or product canvas), providing an overview of the product
- GO (goal-oriented) product roadmap, indicating long-term implementation plans
- Story maps, showing incremental rollout of stories across a release (program increment) cycle
- Developer task boards, showing task assignments and progress
- Kanban boards, indicating the current status of work items

3.4.4.8 Lean Startup

Lean startup was developed by Eric Ries and described in his book *The Lean Startup*.[37] The lean startup approach is based on validated learning, whereby investment decisions are based on feedback from controlled experiments in the marketplace. Despite its name, the approach doesn't apply only to organizations customarily thought of as startups. It can also refer to established telecommunications companies, insurance

companies, and even government agencies. What's important is that the organization is developing a product or service that is novel.

[37] Eric Ries, *The Lean Startup* (New York: Crown Publishing Group, 2011).

Lean startup practices that are incorporated in the analysis guidance within this book include the following:

- Minimum viable product (MVP): The minimal version of a product needed to facilitate learning in the marketplace
- Actionable metrics: Metrics that provide actionable guidance on where next to invest
- Leap-of-faith hypotheses: Unproven theories about a product's value and growth that need to be tested quickly

3.4.4.9 Test-Driven Development, Acceptance Test-Driven Development, and Behavior-Driven Development

Test-driven development (TDD) refers to a "style of programming in which three activities are tightly interwoven: coding, testing (particularly, unit testing) and design."[38] TDD was popularized by Kent Beck in *Test-Driven Development: By Example*, published in 2002. According to TDD, tests are written before programming so that they may focus the work of developers. The TDD process begins with writing a unit test, followed by a failed execution of that test. Coding is performed until the software passes the test, and finally, refactoring is carried out to clean up the code.
Acceptance test–driven development (ATDD) extends this practice beyond unit and low-level integration testing to include full end-to-end user interface testing. The tests are often automated.

[38] "TDD," *Agile Glossary*, 2015, https://www.agilealliance.org/glossary/tdd

As an agile business analyst, you practice ATDD by specifying acceptance criteria for features and stories before implementation in collaboration with business stakeholders, quality assurance (QA) team, and developers. These collaborations are often referred to as the Triad meetings.

Following best practice, you create executable specifications for feature-level acceptance criteria in feature files using the BDD/Gherkin syntax.

> For more on BDD, Chapter 13, section 13.10.9. For more on the Triad, see Chapter 13, section 13.6.3.

3.4.4.10 DevOps

DevOps is an approach for reducing time to delivery by minimizing bottlenecks in the delivery pipeline. The guidelines focus on automating the build/test/deploy so changes can be made frequently and reliably. DevOps is recommended for all scaled agile organizations because its practices, particularly its guidelines for automating the build, test, and deploy steps, are essential for safe, rapid deployment at scale. It's also key to enabling many of the other agile practices and benefits described in this book. For example, by shortening the development cycle through automation, DevOps accelerates learning and supports evidence-based MVP planning.

DevOps guidance is to shift left: to move activities, such as testing and deployment, to an earlier stage in the delivery pipeline in order to find problems earlier, when they are easier to correct. As an agile business analyst, you support this practice by defining acceptance criteria for features and stories before implementation. The criteria are not only sources for test specification but also serve as requirements, guiding the work of developers through specific examples. You specify the acceptance criteria in feature files using the BDD/Gherkin syntax—a natural-language format that facilitates test automation.

3.4.4.11 SAFe

The Scaled Agile Framework (SAFe) was developed by Dean Leffingwell as a framework for applying agile, iterative development to large enterprises. We look at SAFe more closely in Chapter 17. Since we'll be using some SAFe terms in this book, and because SAFe concepts are pretty widely used even by organizations that haven't adopted the framework, we'll preview some of the terminology here.

SAFe Concepts and Terms

SAFe planning is built around the **program increment (PI)**. A PI is a planning cycle of eight to twelve weeks—typically a period of four 2-week iterations followed by one 2-week iteration for innovation and planning. It's roughly equivalent to a quarter release cycle. One advantage of calling it a PI is that you can separate the planning period (the duration of the PI) from the release schedule. For example, you can develop requirements implementation plans based on a ten-week horizon so that teams have enough notice to commit and coordinate plans for large work items. At the same time, you can release on a different schedule based on business and market concerns.

A **program** is a long-range enterprise mission that spans multiple teams and multiple iterations (sprints).

In SAFe, teams aligned to a common mission are organized into a long-lived team of teams, referred to as an **Agile Release Train** (**ART**). Teams across an ART synchronize their planning, test the product's conceptual integrity, and so on, at each PI.

SAFe defines a **feature** as a work item that can be implemented within a single PI by one ART (team of teams).[39]

[39] "A feature is a service provided by the system that fulfills stakeholder needs. Each is developed by a single Agile Release

Train. They are maintained in the Program Backlog and are sized to fit in a program increment so that each PI delivers conceptual integrity." Scaled Agile, "Features and Capabilities," last updated January 2, 2020, https://www.scaledagileframework.com/features-and-capabilities

SAFe and This Book

This book applies to those working in SAFe organizations. Following is a mapping of terms used in the book to SAFe terms:

- A quarter is equivalent to a SAFe PI.
- The quarterly events described in this book correspond to SAFe's PI events.
- Quarterly planning and preparation activities described in the book correspond to activities in SAFe's innovation and planning (IP) iteration. However, the default practice presumed in the book is that these activities are not performed in a reserved iteration, but in parallel with development, toward the end of the prior planning period.
- This book uses the term **product area** to refer to a group of teams dedicated to a subset of the product. It's roughly equivalent to a SAFe ART.

3.4.4.12 Domain-Driven Design

Domain-driven design (DDD) was developed by Eric Evans[40] in 2003. The approach has been experiencing a resurgence lately, possibly as a reaction to the haphazard design choices that have sometimes been made in agile development. In DDD, the business domain drives the design so that the resulting code will closely match the structure and terminology of the business—making it easier to adapt the software over its lifespan to changing requirements. As an agile business analyst, you support the DDD approach by providing an analysis of the business vision, stakeholder goals and objectives, and business-perspective models such as value stream maps that drive design models. A fuller consideration of DDD,

including class diagrams and coding implications, is beyond the scope of this book. For more on business-domain object-oriented modeling, see my earlier book *UML for the IT Business Analyst*.[41]

[40] Eric Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software* (Boston: Addison-Wesley Professional, 2003).

[41] Howard Podeswa, *UML for the IT Business Analyst*, 2nd ed. (Boston: Cengage Learning PTR, 2009).

3.4.4.13 UML and Business Process Modeling Notation

The UML is a broad modeling standard that covers everything from domain models for real-world organizations and processes to technical models that describe design issues. Aspects of the standard relevant to your work as a business analyst include UML's domain models, such as UML's business domain class diagrams and activity diagrams, and use-case models that illustrate the user's view of a product or system.

The Business Process Modeling Notation (BPMN) is a widely used standard for modeling business processes. In this book, you'll learn how to use BPMN models to construct story maps and, in the reverse direction, to use story maps to build BPMN process models for reference on future change initiatives.

For more on BPMN, see Chapter 10, section 10.16.

3.5 OVERVIEW OF AGILE ROLES AND THE BUSINESS ANALYST

For most of this book, our focus will be on the agile analysis competency, not the role. I want to address it here because you might be concerned with this issue, especially if you are a business analyst with waterfall experience wondering where you'll end up after an agile transition. You might also be responsible for organizing agile teams—and wondering how your agile business analysts fit in.

Figure 3.4 is an overview of agile roles and functions related to agile BA.

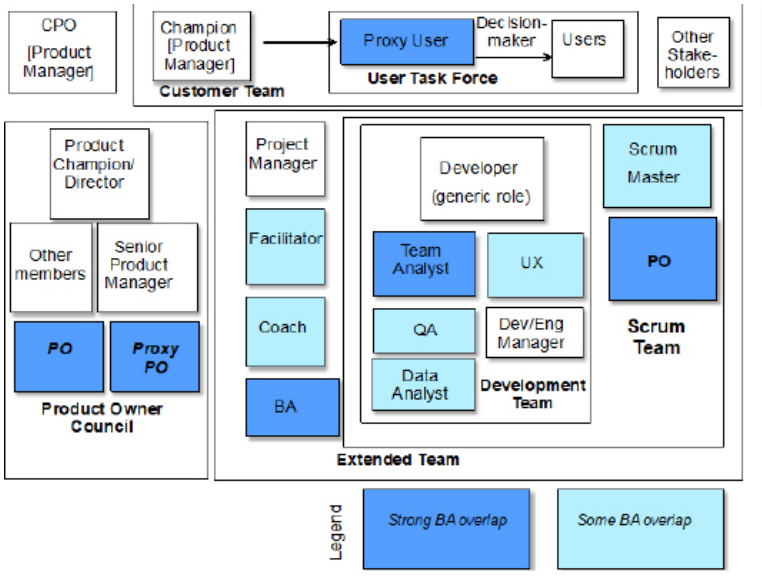

Figure 3.4 The BA function in agile

As indicated in Figure 3.4, roles and functions that overlap strongly with BA include the business analyst, team analyst (unsurprisingly), PO, and proxy PO. Roles with some BA overlap include the facilitator and coach ("free-floating" roles often not dedicated to a particular team), proxy user (representing a group of users), user experience (UX) designer, ScrumMaster, QA professional, and data analyst.

The development team includes all functions necessary to deliver work items (stories). The term **developer** is used in a generic sense to refer to any member of the team. The **extended team** includes nondedicated members, such as business analysts, who are made available to multiple teams. **Scrum team** refers to the development team plus the PO and ScrumMaster.

3.5.1 The Product Owner's BA Responsibilities

Most agile development organizations include a role that works closely with development to represent the business. The role is referred to by different names, including the PO (*product owner* comes from the Scrum framework), customer (an XP term), and product manager. I'll be referring to the role as the PO.

The PO has both inward- and outward-facing responsibilities. Inward-facing responsibilities include

- Expressing PBIs.
- Optimizing the value of work performed by the development team.
- Ensuring transparency and clarity of the backlog.
- Ensuring the development team understands PBIs to the level needed.[42]

[42] Schwaber and Sutherland, *The Scrum Guide*, 6.

Outward-facing responsibilities include meeting with customers, marketers, sales representatives, and strategic planners.

Many POs are chosen for their outward-facing competencies. They tend to be product managers from the business side and business subject matter experts (SMEs) who may have minimal background in requirements management. The solution to this competency gap is to either provide them with coaching and training in agile analysis or to support them with a business analyst well-versed in the competency. Often, there aren't enough POs to fill the role at the team level. As a business analyst, you might help solve that problem by acting as team PO or proxy PO, while the SME serves as area PO over a group of teams.

3.5.2 The Agile Team Analyst

According to frameworks such as Scrum, the team analyst is a competency, not a formal, dedicated role. On

agile teams in the new-technology sector, the analysis competency is often filled by someone other than a dedicated analyst (e.g., the PO or members of the development team who have other primary areas of expertise, such as in data analytics or UX design).

Mainstream businesses often include a designated analyst on their agile teams—either as a dedicated member or as a member of the extended team, shared among a group of teams. As someone in this role, you support the PO by eliciting and managing detailed requirements—a responsibility similar to the IT business analyst role in traditional organizations. We look at considerations for and against dedicated business analyst roles later in this chapter.

3.5.3 The ScrumMaster's BA Responsibilities

While the main focus of the ScrumMaster (a role within the Scrum framework) is on coaching the team on the Scrum process and removing impediments, many aspects of the role overlap with BA:

- Finding techniques for effective product backlog management
- Helping the Scrum team understand the need for clear and concise product backlog items
- Understanding product planning in an empirical environment
- Ensuring the PO knows how to arrange the product backlog to maximize value
- Facilitating Scrum events as requested or needed[43]

[43] Schwaber and Sutherland, 6–7.

3.5.4 Proxy User

When there are many diverse users, a user task force may be formed to represent a *group of users*, with a proxy user as its head, representing the group's needs. Because the required competencies overlap with BA, business analysts often occupy this position. The proxy user also has final decision-making authority on behalf of the group—something outside of the traditional BA function.

3.5.5 BA Responsibilities of the Product Champion (Director)

The customer team (an XP entity) is a body that includes everyone needed to ensure the software meets the needs of stakeholders. Members may include the product manager, proxy product owners, users, QA professionals, and UX designers. The product owner council (POC) is a similar organization, consisting of the POs and proxy POs that lead the teams working on an initiative.

A customer team or POC is led by a product champion (also known as a director)—a first among equals. The product champion often has to resolve prioritization conflicts between the technical and business sides (e.g., when deciding how much time to spend on technical debt vs. new product development). For this reason, the champion is often someone with a foot in each world. Many are former software developers who have moved over to the business side. This is also the background of many business analysts, making them a natural fit for this role.

For more on the POC, see Chapter 17, section 17.7.8.

3.5.6 Coach

Coaches are assigned to work with new agile teams until they are able to go it alone—at which point the coach gradually hands off control to a local member (e.g., a

ScrumMaster). Many aspects of the role relate to BA issues, such as guiding the team on how to represent requirements and when to analyze them.

3.5.7 When Are Dedicated Business Analysts Advised?

The short answer is "often." Here's why: An agile organization requires more resources from the business side than waterfall does because it requires one business decision-maker per agile team of about seven to ten people and because they are involved *throughout* development instead of only at its endpoints. There are often insufficient business decision-makers, such as product and program managers, to fill these positions. If you're responsible for organizing the teams, what do you do? There are many solutions to this problem, but most involve the inclusion of an agile business analyst in some capacity.

3.5.7.1 Dedicated Team Analyst as Assistant to the Rotating PO

One effective solution was first described to me by Angus Muir of Travelers Insurance. I've been recommending it to teams ever since. Find business SMEs to serve as *rotating* team POs and empower them to make decisions and set priorities. This frees the more senior product managers to act as area POs across a group of teams. At the same time, the rotation of SMEs in and out of the team-level PO role prevents them from going stale and losing touch with the market.

The disadvantage of this approach is that a new, inexperienced SME is continually moving into position as a team PO. The team analyst, in this case, provides continuity with respect to the requirements, jockeying the requirements through the development process as POs are swapped in and out. The analysts provide each new PO with an understanding of why stories have been

prioritized the way they have, what business objectives they support, the dependencies between them, and the processes and tools used to manage the requirements.

3.5.7.2 Business Analyst as Proxy PO

A different solution is to assign a proxy PO rather than a full PO to each team. The proxy PO is similar to a PO but with limited authority to make local prioritization decisions. Business analysts are often chosen to fill the role because they have the necessary requirements-analysis capabilities.

3.5.7.3 Team Business Analyst Supporting a Shared PO

Another solution advocated by scaled agile approaches such as LeSS is to do away with the team-level PO. Instead, one PO, shared among several teams, focuses on the role's outward-facing aspects. Detailed requirements analysis is offloaded to the team, making it especially important that it includes someone with strong analysis competencies.

3.5.7.4 Other Scenarios Where a Dedicated Business Analyst May Be Advised

Other scenarios where high-performing business analysts are especially valuable include

- Complex business initiatives, such as mergers, transformative changes, and process reengineering initiatives, that demand in-depth analysis skills.
- Third-party or vendor solutions where the financial risk is heavily front-loaded.
- Large, multiple-team initiatives where business analysts help track and manage requirements dependencies and communicate requirements across all teams.

3.5.8 Business Analysts Provide Requirements Leadership

As an agile business analyst, your function is to provide leadership in the BA competency to the rest of the team: you facilitate requirements workshops with business stakeholders and developers present, and you coach other team members in agile requirements activities—such as writing, splitting, and negotiating stories. You are also engaged in analysis tasks, such as eliciting and defining the business rules for a story. During the Triad meetings, you work with the PO, testers, and developers to

- Define acceptance criteria for features and stories.
- Challenge developers to explain which aspects of a story's requirements contribute most to high estimates.
- Consider workarounds that achieve similar business goals at lower cost.

Throughout the development process, you help ensure that the team is working on the requirements of the highest value to the business.

3.5.9 The Distinction between Business, Business Systems, and IT Analysts

Many organizations have different types of business analysts—such as **business analysts** (when referring to analysts with a higher-level business perspective) and **business systems analysts (BSA)** (for those with a more technical orientation) as well as varying levels of analysts (e.g., senior, intermediate) within each type. I try to avoid spending too much time distinguishing between these roles, as the effort goes against the cross-functionality preferred in an agile organization. In general, though, if these subroles are used in your agile organization, the "business" business analyst is usually

the person responsible for product visioning, the business case, and defining business goals and objectives (i.e., the higher-level business requirements for an initiative). The IT business analyst or the BSA is responsible for the more detailed IT requirements (functional requirements) while working daily with the team. As a BSA, you are usually expected to have some technical background. For example, you may be asked to map required data fields to existing data tables.

3.6 SOFT SKILLS OF THE AGILE BUSINESS ANALYST

So much of this book is concerned with tools such as user stories and story maps that you might think that BA is all about technique. It's not. It's very much a *people* competency. In fact, the most common reason reported to me for including business analysts on agile teams is that they fill a *soft skills* gap. Many of the sharpest technical minds I've worked with in the industry were missing these qualities—and turned out to be poor business analysts despite their exemplary qualifications. One BA had rare technical expertise in precisely the geographical mapping systems my customer was looking for but had a terrible way with people. Another had a strong background working on scaled agile projects, but stakeholders perceived him as arrogant. I never worked with either of them again and have since learned to screen much more carefully for soft skills.

If you're selecting people to fill the BA competency on an agile team, look for the soft skills described in the following sections.

3.6.1 Making the Unconscious Conscious

It's not a coincidence that business analysts and psychoanalysts both have "analyst" in their names. The main job in both cases is to *make the unconscious conscious*. A good business analyst can take knowledge

that is deep inside the brains of customers and stakeholders and bring it into conscious expression so that developers are able to act on it.

3.6.2 Curiosity

A great business analyst should be naturally curious about people, their needs, and how they do things. Avoid candidates who like to dominate conversations. Prefer candidates who spend more time asking questions and listening to others.

3.6.3 Agent of Change

Business analysts are change agents. Great business analysts have the strength of personality and self-confidence to challenge assumptions, even when the issue at hand is "above their pay grade."

3.6.4 Political Intelligence

Business analysts can get themselves into trouble if they aren't quick to pick up on the politics of a situation. One business analyst who used to work for me reported to a high-level executive who was feared by the people under him. A business analyst with good political intelligence would have talked to his staff in private. This business analyst interviewed them as a group. Unfortunately, stakeholders held back information, and there were serious gaps in the solution that was developed. Those gaps added a lot of unexpected costs.

Political intelligence also means knowing what to say and what not to say—and to whom. You don't want a business analyst who betrays confidences or makes offhand commitments to the business without checking first with the team.

3.6.5 Works Well with Difficult People

Business analysts are rarely able to choose the stakeholders and developers they work with. They need to be able to manage people—regardless of the personality type in front of them. (By the way, this is one of those situations where growing up with a difficult parent can provide benefits later on in life.)

3.6.6 Negotiation Skills

Business analysts must often work with opposing parties to bring them toward shared goals and requirements priorities—whether it's business stakeholders with conflicting priorities for the product or it's "business vs. development" with business stakeholders pushing to prioritize new features while developers favor technical enhancements.

3.6.7 Facilitation

Business analysts need excellent communication and facilitation skills: They need to be ready to facilitate requirements workshops as needed, present requirements to the team, and facilitate planning sessions such as big room iteration planning events.

3.6.8 Adaptability

You'd think agile consultants and analysts would be the most adaptable people in the world. After all, they specialize in agility! Surprisingly, they can sometimes be quite rigid and doctrinaire in their views. That's not a desirable trait for agile business analysts because they often have to work in a context that isn't the ideal agile environment. For example, the organization may not have the technical infrastructure for the DevOps practices that enable agile's short development cycles. In this case, you want business analysts who have the flexibility to adapt to the situation and recommend the best approach given the context.

3.6.9 Not Afraid to Ask Questions

A good business analyst is not afraid to ask “dumb” questions. These often turn out to be the very questions that many others also don’t understand but are embarrassed to ask about. A great business analyst knows her primary value is not that she knows the business better than anyone else, but that she knows what questions to ask and when.

3.6.10 Sense of Humor

Business analysts often have to deal with contentious situations (e.g., when managing conflicting priorities or when diffusing tension between the business’ and developers’ estimates of effort). Humor is often the key that opens the door to resolution.

3.7 13 KEY PRACTICES OF AGILE ANALYSIS AND HOW THEY DIFFER FROM WATERFALL

Let’s summarize some of the key practices and properties of agile analysis and how they differ from BA as practiced in a waterfall context.

3.7.1 A Competency, Not a Role

In waterfall, BA is often carried out by a dedicated role, such as a business analyst, IT business analyst, or BSA. In agile development, BA is a team *competency*, but it may be a formal role. Formalized roles within the team are discouraged because they lead to bottlenecks.

3.7.2 A Facilitator, Not a Messenger

On waterfall projects, business analysts are often seen as intermediaries who “throw the requirements over the fence” to the developers. As an agile business analyst, your role is to be not a messenger, but a *facilitator*,

enhancing the quality of direct discussions between business stakeholders and solution providers.

3.7.3 Changes to Requirements Are Welcomed

In a waterfall process, you freeze requirements before development and strictly control subsequent changes. As an agile business analyst, you welcome changes to the requirements at any time with minimal ceremony.

3.7.4 Collaboration with Developers vs. Contractual Relationship

As an agile business analyst, your goal is to support a collaborative relationship whereby the customer and solution provider collaborate throughout development to find the best solution that delivers the desired business outcomes. It's not to lay down a set of comprehensive requirements that can serve as the basis for a contract, as is the case for waterfall development.

3.7.5 Just-In-Time Requirements Analysis

As an agile business analyst, you analyze requirements incrementally at the LRM—as close as possible to when they will be implemented. This is in contrast to waterfall development, where the requirements are analyzed upfront.

3.7.6 Conversation versus Documentation

In a waterfall process, you rely on written specifications as the primary means for communicating requirements to developers. As an agile business analyst, you rely primarily on conversation. This reduces waste (the added effort to create and maintain the documents), speeds the development process, and enhances the ability to

respond to change, since it is much easier to change direction when little is written down. This doesn't mean you won't create any requirements documentation in agile, just less of it.

3.7.7 Specification by Example: Acceptance Test–Driven Development

As an agile business analyst, specific examples and narratives are your preferred vehicle for explaining requirements and test cases. You specify acceptance criteria for each requirements unit before implementation so that the criteria can be used as input to the developers. The acceptance criteria also serve as the basis for specifying automated and manual tests.

3.7.8 Small Requirements Units

As an agile business analyst, you split requirements into small units (user stories) that can be comfortably implemented and delivered within a short period. There is no need for this activity in a waterfall approach.

3.7.9 Vertical Slices of Functionality

Because agile processes aim for each requirements unit to deliver real value to the user, you have to specify each requirements unit so that it provides a "vertical slice" of functionality. For example, a vertically sliced requirements unit would call for implementing a couple of fields right through the architecture (UI layer, business layer, database layer) rather than all the fields of a single layer, such as the graphical user interface.

When complex products are involved, one small agile team is often unable to include all the necessary competencies to deliver value. Consequently, you may need to define the vertical slice at the higher level of a feature (multiteam) and then split it into smaller team-level stories.

3.7.10 Lightweight Tools

Agile analysis favors lightweight solutions to requirements management problems over solutions that require extensive overhead. For example, in this book, you'll learn of two approaches for managing requirements dependencies between two teams. The heavyweight solution is to track dependencies using a requirements management tool. The lightweight solution is to assign someone to sit in on the other team's planning meetings—an investment of about fifteen minutes per day that achieves the same objectives.

3.7.11 Business Analyst and Business Stakeholder Engagement across the Complete Development Lifecycle

Because analysis proceeds incrementally over the course of an initiative, you need to prepare stakeholders to be involved *throughout* the development lifecycle, not only at its ends, as is the case for waterfall development.

3.7.12 Mix of BA Classic and Agile BA Tools

To be effective as an agile business analyst, you need to be familiar with a mix of agile requirements techniques and tools—such as story maps and user stories—as well as traditional BA tools, such as domain modeling and business process modeling.

3.7.13 Meet Them Where They Are

In the course of your work as an agile analyst, there are many opportunities where you'll need to choose between options that seem to work best for the situation and those that are more ideologically pure from an agile standpoint. You should certainly not lose sight of the optimum solution and should advocate for it as a long-term solution. But for immediate guidance, choose the

option that provides the best value to the business based on where the business is right now. Remember that in any given situation, *the goal of the agile business analyst is to do well by the business—not to do well by agile.*

3.8 AGILE BUSINESS ANALYSIS RULES OF THUMB

Whenever I work with a group that is new to agile, many of the same questions pop up time and again. How many user stories are typical for an iteration? How many acceptance criteria are typical for a story? How many points are typical? I've assembled many of these questions and their answers in the rules of thumb listed in Appendix A2.1. Keep in mind that these are broad guidelines, not strict rules. They are particularly valuable as a quick reference when you're starting out as an agile analyst and don't have a guide for what would be "normal" in an agile development organization. They will give an idea about what's typical for velocity, story sizes, iteration length, and other parameters. Once you've been at it for a while, let experience be your guide, since practices will vary according to circumstances.

3.9 CHAPTER SUMMARY

Here are the key points covered in this chapter:

- The atomic requirements unit in agile development may be known as a product backlog item (Scrum), story (XP), or use-case slice (*Use Case 2.0*). The most widely used unit is the story.
- A feature is a requirement item that may be delivered by one or more teams within a quarter, release cycle, or program increment. A story is a work item that is small enough to be implemented by a single team in about one or two weeks.

- To be accepted, a story must satisfy its acceptance criteria as well as the definition of done (DoD).
- Analysts contribute formally to an agile organization as dedicated team analysts, proxy POs, and extended team members.

3.10 WHAT'S NEXT?

In this chapter, we looked at the fundamental concepts behind agile BA. In Chapter 4, "Overview of Analysis and Planning Activities across the Agile Development Lifecycle," we focus on how to put these ideas and related techniques into practice over the course of an agile development lifecycle.

Chapter 4. Overview of Analysis and Planning Activities across the Agile Development Lifecycle

This chapter provides an overview of the agile analysis and planning process. It begins with the Agile Analysis and Planning Map, depicting analysis and planning tools used during an agile product development lifecycle. The tools include activities, artifacts, and events.

The map is divided vertically into zones of activity—clusters of related work. For example, the Quarterly Inception zone includes activities to launch the quarter, such as planning which features will be implemented. Horizontally, the map is divided into three lanes:

- Short Lane, for planning horizons up to about six months
- Long Lane, for horizons from six months to five years
- Grand Lane, for large-scale agile organizations.

The chapter explores each one through a narrative that illustrates what to do when, as development progresses along a lane.

4.1 OBJECTIVES:

This chapter will help you

- Understand how the activities and artifacts of agile analysis and planning fit together across the agile development lifecycle.
- Understand what activities apply to short-term and long-term planning horizons and the additional activities needed for scaled agile organizations.

- Understand how to use the map to plan analysis and planning activities for a given situation.

4.2 OVERVIEW OF THE AGILE ANALYSIS AND PLANNING MAP

Figure 4.1 illustrates the activities and tools of agile analysis and planning across the agile development lifecycle.

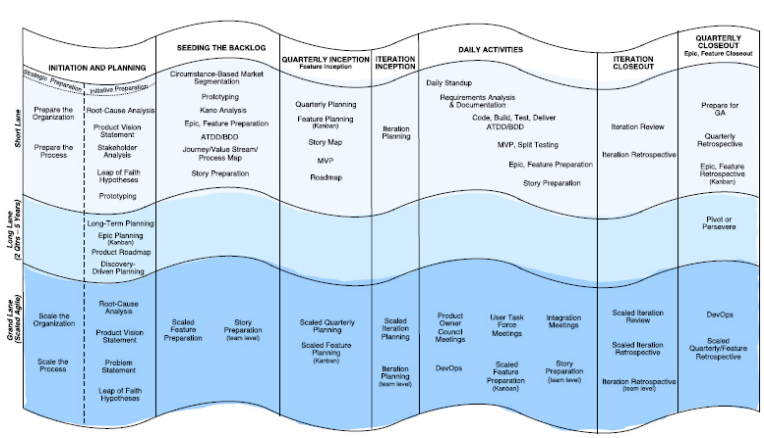

Figure 4.1: Agile Planning and Analysis Map

Figure 4.1 is divided vertically into zones and horizontally into lanes. Each zone represents a set of related analysis and planning activities, events, and artifacts. For example, the Quarterly Inception zone includes the story map and minimum viable product (MVP)—both of which are used to plan the quarter. I refer to these as **zones**, rather than phases, because agile development is not sequential: activities and artifacts are not performed to completion before the next one begins; they are refined incrementally over time. The progression is not always from left to right, either; it may reverse itself at any time. For example, an individual story may move from the testing activity back to detailed requirements analysis if testing reveals that more work is required to satisfy the customer.

A zone's left-to-right position does provide, however, a *rough* indication of when it is carried out *for the first*

time and when it *primarily* occurs. For example, the release roadmap is created at the start of the quarter, though it is updated as needed. If an activity is used in two key contexts, I've listed it twice. For example, story preparation is an aspect of seeding the backlog. It is also performed as a daily activity once the initiative is underway. To make that clear, story preparation appears in both the Seeding the Backlog and the Daily Activities zones on the map.

Each horizontal lane in Figure 4.1 represents a particular scenario. Across the lane, you'll find activities and artifacts useful for the corresponding situation. For example, in Figure 4.1, the Grand Lane includes product owner (PO) council meetings, indicating that the activity is relevant to a large-scale agile organization.

The zones and lanes form a grid: based on the position of an item, you can see which scenario to use it for and the zone it belongs to. For example, in Figure 4.1, integration meetings are shown in the Grand Lane, indicating it is used for scaled organizations. It appears in the Daily Activities zone, indicating that these meetings are ongoing.

4.3 THE ZONES

The map in Figure 4.1 contains seven zones of activity:

- **Initiation and Planning:** Activities, artifacts, and events to prepare and plan for new product development or an epic—a significant change to an existing product. This zone also contains activities to improve the organization's maturity level in agile development and analysis.
- **Seeding the Backlog:** Activities to create and prepare the initial set of work items (epics, features, and stories) in the backlog.

- **Quarterly Inception (Feature Inception):** If the organization is using flow-based planning (e.g., Kanban), this zone covers activities to plan an upcoming feature, including estimation, team commitments, and agreement on a roadmap for delivering stories. If the organization is using a timeboxed approach, this zone covers activities to plan all the features that will be delivered in the upcoming quarter.
- **Iteration Inception:** This zone includes activities for launching an iteration—a short planning cycle of about one to two weeks. The zone applies to organizations using a timeboxed planning approach, such as Scrum or XP.
- **Daily Activities:** This zone is for ongoing analysis and planning activities, such as daily standups and detailed requirements analysis. Following best practices, stories should be continuously delivered as they are done. To deliver a story means to provide it in a deployable state. Actual deployment to production or the market can occur at the time it is delivered or be delayed at the customer's discretion until a planned release date.
- **Iteration Closeout:** Final activities at the end of an iteration. This zone applies to organizations using a timeboxed planning approach, such as Scrum or XP.
- **Quarterly Closeout (Epic, Feature Closeout):** Final activities at the end of a long planning cycle—a quarter, release cycle, epic, or feature. The zone includes activities to prepare for general availability (GA)—the deployment of changes to the market or into production.

4.4 THE LANES

The map in Figure 4.1 contains three lanes, each depicting activities, events and artifacts for a particular

scenario.

- The **Short Lane** applies to a scenario with planning needs up to the next quarter, release, or program increment (PI). A PI is a SAFe planning cycle of about eight to twelve weeks. For brevity, I often use the term *quarter* as shorthand for any of these periods—a calendar quarter, release cycle, or PI.
- The **Long Lane** applies to a scenario with planning horizons of two quarters to five years.
- The **Grand Lane** applies to a scaled agile organization. Some items on this path are referred to as *scaled* events. The terms denote an event with representation from all collaborating teams, such as the teams within a product area or a SAFe Agile Release Train (ART). If the development organization has only one level of collaborating teams, the term applies to all the teams working on the product.

The lanes are not exclusive. If you have long-term planning needs, you also have short-term planning needs. Consequently, you'd use the tools in the Short and Long lanes. If the organization is large and has long-term planning needs, you'd use tools from all lanes.

> Use the map as a checklist of tools to consider when customizing the right process for your situation. Once the process is in action, use the map to find the right tools based on where you are in the development cycle, such as iteration closeout.
> The map tells you what tools to *consider*. The tools you *select*, though, will depend on your situation. Be a minimalist: use a technique only if you can specify at least one problem that it solves. Don't use it just because it's on the map.

4.5 A STORY IN THREE ACTS

Now that we've reviewed the process, let's see how to carry it out by returning to Customer Engagement One (CEO) Inc. (see Chapter 2, "Agile Analysis and Planning: The Value Proposition," section 2.4), the company developing the CEO app—a product that enables companies to manage their engagements with customers across multiple platforms from a single interface.

In our retelling of the story, you have been asked to take on agile analysis and planning responsibilities during the company's early phase as a young startup. In real life, numerous individuals and roles might carry out these responsibilities, such as business analysts, business systems analysts (BSAs), team analysts, proxy POs, and POs. In this story, we simplify things by collapsing them into a single role: the analyst. To make it more realistic, let's assume you'll fill the role initially, with the expectation that, as the company grows, you'll become a lead in the competency, and others will be hired as needed.

We follow the CEO story in three acts, each focusing on one lane of the map and highlighting some of its main activities. We begin with Act 1. It illustrates the Short Lane, the first row in Figure 4.1.

4.6 ACT 1: THE SHORT LANE

You have been informed that CEO is to be based on a freemium business model. Users will be offered a complimentary version, which they can use until the number of collaborating users reaches a preset maximum. To go beyond this number, they will be required to upgrade to a premium, paid subscription. In the early releases, the free and premium versions will have the same capabilities, except for the cap on users in the free version. The plan is to eventually add additional features to the premium version for enterprise-wide use.

The marketing team intends to use a bottom-up strategy to sell the product—pitching the product first to support agents who engage with end customers. The assumption is that enough users within a company will want to collaborate to trigger conversion to the premium version.

Your first focus is on initial preparation and planning, the Initiation and Planning zone on the map in Figure 4.1.

4.6.1 Initial Preparation and Planning

Because there is so much uncertainty about the requirements, you recommend that a prototype be tested as soon as possible. The testing is carried out, and results indicate that the most popular capabilities are the ability to manage communications across multiple platforms without having to "hop in and out" of separate social networks and the ability to analyze trends across platforms. Based on the positive response to the prototype, the company secures funding to develop CEO First Generation (CEO-1G)—the first version of the product to be released to the market.

You learn from the development lead that CEO-1G will be built as a monolithic system—simple enough for one team of eight to ten members to contain all the required competencies needed to deliver stories Due to the volume of work, two to three of these teams will be needed. (Prepare the Organization in Figure 4.1)

You choose a flow-based (Kanban) planning process because it accelerates learning at a time when there is high uncertainty about the product (Prepare the Process in Figure 4.1). You choose the Short Lane because planning needs do not extend beyond a few months at a time.

Since there is no vision statement for the product, you facilitate a workshop with stakeholders to create one

(Product Vision Statement in Figure 4.1). Together, you craft the following product vision statement: “To connect companies with their customers, anywhere, anyhow.” You perform a stakeholder analysis to understand users, customers, suppliers and other stakeholders and their objectives for the product. These include customer support agents, marketers, sales agents, and data analysts.

Next, you convene a group of business stakeholders and developers to explore the unproven assumptions that are key to the product’s viability (Leap of Faith Hypotheses in Figure 4.1). You identify the following hypotheses:

- Support agents will use the product to collaborate with other agents on messages from customers.
- Users will invite enough of their colleagues to trigger conversion to the premium model
- Enough companies will move to a subscription model for the product to reach profitability within five years.

You advise the group that it will be designing MVPs to test these hypotheses.

4.6.2 Seeding the Backlog

Next, you coach marketers to use *circumstance-based market segmentation* to anticipate the ways that people will use the product. Using the technique, they learn that the high-level usages for the product are, in order of preference, as follows:

- **Manage Messages:** Provide the ability to respond to customer messages (through support agents).
- **Marketing Campaigns:** Push marketing messages out to the community.
- **Sales:** Sell goods and services to customers.

- **Analytical Reporting:** Data analytics about engagement with the app, how marketing campaigns are performing, and so on.

You create a feature for each of these usages and add it to the backlog. You turn your attention to the first feature in the backlog: *manage messages*. To prepare it for implementation, you facilitate a value stream mapping workshop (Feature Preparation in the Seeding the Backlog zone in Figure 4.1). Attendees include business subject matter experts (SMEs) and representatives of the user roles that participate in the workflow for managing a message.

With your coaching, the group develops a workflow model that indicates the following steps:

- The system *ingests* messages.
- A Tier 1 support agent
  - *views* a message,
  - *triages* it,
  - *tags* it according to the topic,
  - *responds* to it, *resolves* it, or *assigns* it to a Tier 2 agent or queue.
- A Tier 2 agent may resolve the message or pass it to a Tier 3 agent or queue.

Using the behavior-driven development syntax (Given-When-Then), you specify end-to-end user acceptance testing (UAT) to validate that this sequence of steps can be carried out correctly (Behavior-Driven Development in the Seeding the Backlog zone in Figure 4.1).

You meet with the PO, team, and stakeholders to estimate the feature and develop a plan for implementing it (Feature Planning in the Feature Inception zone in Figure 4.1) During the meeting, you

determine the following stories for the Manage Messages feature:

- As a Tier 1 support agent, I want to ingest messages from platform X so that I can see what customers are saying about the product.
- As a Tier 1 support agent, I want to view messages with simple formatting so that I can see what customers are saying about the product.
- As a Tier-1 support agent, I want to assign a message to a Tier 2 agent or work queue so that someone with more knowledge than me can collaborate on its resolution.
- As a Tier-2 support agent, I want to assign a message to a Tier-3 agent or queue so that difficult technical support issues are addressed by someone with the necessary expertise.
- As a support agent (all tiers), I want to respond to an assigned message so that I can advance it towards resolution.
- As a support agent (all tiers), I want to resolve an assigned message so that the issue may be closed.

You identify other, lower-priority stories for the feature, such as the following:

- As a support agent, I want to assign a tag (topic) to a message so that topic-based analytics can be generated and so that the message can be routed to the agent with the appropriate expertise.

Business stakeholders have indicated that this last story is a “nice-to-have” but that the MVP as a whole would be valuable without it. Consequently, you add the story to the bottom of the backlog and don’t include it among the test scenarios required for feature acceptance.

You prepare the first stories in the backlog during the Triad sessions with business stakeholders, quality

assurance (QA) professionals, and developers. You specify story acceptance criteria. Developers gain sufficient understanding of the story to estimate it reliably.

4.6.3 Daily Activities

Planning and coordination occur daily as you and the rest of the team meet for a 15-minute daily standup (Daily Standup in Daily Activities zone in Figure 4.1). Throughout the day, you meet with users to analyze the remaining requirements for current stories and to test stories as developers implement them. When a user story has met its acceptance criteria and developers have confirmed that the definition of done has been satisfied, the user story is considered done. Stories are continuously integrated and tested, with much of the work automated (Code, Build, Test, Deliver in Figure 4.1).

As the feature is nearing completion, you begin preparations on the next feature in the backlog (Feature Preparation under Daily Activities in Figure 4.1). The next feature is for an MVP to validate whether customers will invite enough of their colleagues to trigger conversion to the premium model. You work with the development team and business SMEs to split the feature into the following stories:

- As a user, I want the capability to invite my colleagues to use the product so that I can receive loyalty rewards.
- As a prospective customer, I want to respond to the invitation so I can begin using the product.
- As a data analyst user, I want to view metrics by message topic so I can detect negative comments and reviews that are at risk of going viral.

Developers flag an impediment. The data analyst's story is dependent on the following story:

- As a support agent, I want to assign a tag (topic) to a message so that topic-based analytics can be generated.

Currently, the latter story is sequenced far down in the backlog. Everyone agrees to move the story to the top of the backlog in order to remove the impediment.

You continue to prepare stories on a rolling basis, as each story nears the top of the backlog (Story Preparation in the Daily Activities zone in Figure 4.1). To prepare a story, you meet with the PO, QA, and developers to determine its acceptance criteria and develop wireframes. Each story then undergoes coding, integration, testing, and deployment to a select group of users.

4.6.4 Feature Closeout: Prepare for GA

As soon as enough stories in a feature are done for the feature to satisfy its acceptance criteria, the feature is released for GA. You continue to prepare features and stories on a rolling basis as they come up in the backlog.

4.6.5 Quarterly Inception, Iteration Inception

Some time has passed, and you are reconsidering the analysis and planning process because of changing circumstances. For one thing, the product is now integrated with third-party providers who require a long lead for change requests. For another, the marketing team wants releases to come on a three-month cycle so they have enough lead time to plan a campaign. To accommodate these needs, you advise that each team reserve a portion of its time for initiatives planned on a quarterly basis. The remainder of the team's time will be

retained for flow-based (Kanban) work items to preserve the ability to respond quickly when necessary.

At the start of each quarter, each development team convenes to determine the features it will implement over the quarter and specify the plan for rolling them out (Quarterly Planning in the Short Lane in Figure 4.1). They reconvene every two weeks to plan stories for the upcoming iteration (Iteration Planning in the Short Lane in Figure 4.1.)

4.6.6 Iteration Closeout

At the end of each iteration, the team, PO, and interested stakeholders meet for an iteration review to demonstrate the completed stories to stakeholders. Next, the team meets for an iteration retrospective to explore what has been working well and what practices need to be changed (Iteration Closeout in Figure 4.1).

4.6.7 Quarterly Closeout

At the end of the quarter, you facilitate a quarterly retrospective to review how things went and make recommendations (Quarterly Closeout in the Short Lane in Figure 4.1). QA reports it is struggling to meet the demands of development teams to create and run automated tests. The DevOps lead suggests that the solution is not more testers but better coaching so that developers can create and run tests on their own.

4.7 ACT 2: THE LONG LANE

The free version of CEO is now well established. It includes rich features for replying to customer support messages. The company has now secured longer-term funding to develop new features over the three years in the premium version to support marketing and sales and analytical reporting. There are now more teams, but the product is still simple enough that each team can

operate, for the most part, as a self-sufficient unit. You decide to update the process to include techniques from the Long Lane.

You meet with stakeholders to develop a long-term plan for rolling out features (Long-Term Planning in the Long Lane in Figure 4.1). During this activity, you create a product roadmap, outlining the goals, dates, and features for releases over the next three years. Attendees agree to extend customer-support capabilities to additional social networks in the first release. Marketing features will be added in the release that follows; sales and analytical reporting features will be added in a later release.

At the start of each quarter, you review the product roadmap with stakeholders and adjust it as necessary. You carry out daily analysis and planning activities, as described for the Short Path.

4.8 ACT 3: THE GRAND LANE

The company has now been in business for a few years. The CEO app has grown from a simple, monolithic product to a complex system with multiple microservices and architectural tiers. It's no longer possible to cover all the required competencies within a single team. As a result, multiple teams often have to work closely together to deliver value. You decide to add tools from the Grand Lane to the analysis and planning process because they support the coordination of multiple teams.

4.8.1 Scale the Organization

You facilitate a meeting to discuss how the teams should be organized (Scale the Organization in Figure 4.1). One lead developer argues that teams be formed on the basis of competencies—one for user experience, one for the business layer, one for the database layer, and so on. You argue, instead, that teams be organized around

value to minimize coordination overhead. You gain agreement that there should be four groups of teams, each focused on a distinct product area, or *usage* of the product:

- Customer support
- Marketing campaigns
- Sales
- Analytical reports

Each feature team will include business and technical competencies to deliver features within its product area. Component teams will be set up to own commonly used microservices and software interfaces. Teams within a product area will collaborate with each other and with component teams, as necessary, to deliver value.

POs will be allocated as follows: each team will have a team-level PO, each product area will have an area PO, and there will be a product-level PO for the product as a whole. Additional business analysts will be brought in at the team level and as extended members at the product area level, as needed. You will focus on product-level needs and act as lead analyst for the customer-support product area.

4.8.2 Scaled Quarterly Planning

You convene all teams in your product area at the start of each quarter for a two-day multiteam quarterly planning session to determine the features that will be delivered during the quarter (Scaled Quarterly Planning in the Grand Lane in Figure 4.1).

4.8.3 Scaled Iteration Planning

At the start of each iteration, you bring all teams together within your product area for scaled iteration planning, using the “Big Room” approach (Scaled

Iteration Planning in the Grand Lane in Figure 4.1). Working closely with each other, the teams plan the stories they will implement in the upcoming iteration. During the session, they check in with each other regularly to ensure their plans align.

As the teams determine what work will be done over the next two weeks, feature team POs advocate for new capabilities, while component team POs promote technical improvements. With the engagement of the product-level PO, you help bring everyone toward a consensus, guided by a company rule of thumb to spend approximately 75 percent of the budget on feature development and 25 percent on technical improvements.

4.8.4 Daily Planning and Analysis

Each day begins with a team-level daily standup. Prioritization conflicts raised during the standup are addressed in the PO council meetings that convene two or three times per week.

Starting around the second week of each iteration, the team analysts in your product area focus on preparing stories for their next iteration planning session.

4.8.5 Iteration Closeout

At the end of each iteration, you gather all the teams in your product area for a scaled iteration review and retrospective.

4.8.6 Quarterly Closeout

At the end of the quarter, you bring together all the teams in your product area for a scaled quarterly retrospective to review how the past quarter went and begin preparations for the next one.

4.9 CHAPTER SUMMARY

Here are the key points covered in this chapter:

- The Agile Analysis and Planning Map is divided into zones and lanes.
- Each zone (e.g., Initiation and Planning, Quarterly Closeout) groups related activities, events, and artifacts.
- Each lane describes a scenario. It illustrates the activities and artifacts used in the scenario and when to use them.
- The Short Lane is for agile projects that require planning for a horizon up to about three months and do not require coordination between teams.
- The Long Lane is for initiatives that require long-term planning.
- The Grand Lane is for large-scale initiatives that require coordination between teams.

4.10 WHAT'S NEXT?

This chapter provided a brief overview of the whole process. Now, we're going to rewind and start again—this time pausing to learn how to use the tools that apply at each step along the way. That's the subject of the remaining chapters—beginning with Chapter 5, "Preparing the Organization," on preparing the organization.

Chapter 5. Preparing the Organization

This chapter provides guidelines for preparing small development organizations for agile software development. It includes preparations that are specific to an initiative and general guidelines for transitioning organizations toward higher maturity levels of agile analysis.

The chapter explains how to structure teams by value so that each team can deliver capabilities to customers and users, with minimal dependencies on others. The chapter also explains why team dependencies cannot be entirely eliminated. It includes guidelines for forming feature teams and component teams. Also included are guidelines for preparing non-IT divisions new to agile, including the marketing team, financial planners, and compliance officers.

This chapter focuses on small agile organizations of up to ten teams that do not require much inter-team coordination. For guidance on scaled development organizations of collaborating teams, see Chapter 17, “Scaling Agility.”

> For guidelines on scaling the agile organization, see Chapter 17, section 17.7. For guidance on preparing the enterprise beyond the context of software development, see Chapter 18, “Achieving Enterprise Agility.”

5.1 OBJECTIVES

This chapter will help you

- Use the purpose alignment model to determine whether a product or capability should be developed in-house or delivered by a third party.
- Prepare departments and stakeholders new to agile development for an agile initiative.
- Determine an organization's target agile maturity level and its readiness for agile development.
- Know what structural changes to champion if the organization is not at its target maturity level.
- Understand Kotter's eight steps for accelerating change in an organization.
- Use your knowledge of the ways the product is used to optimize productivity by organizing cross-functional teams around usage.
- Manage the expectations of stakeholders new to agile development.

5.2 THIS CHAPTER ON THE MAP

Figure 5.1 highlights the activities covered in the chapter.

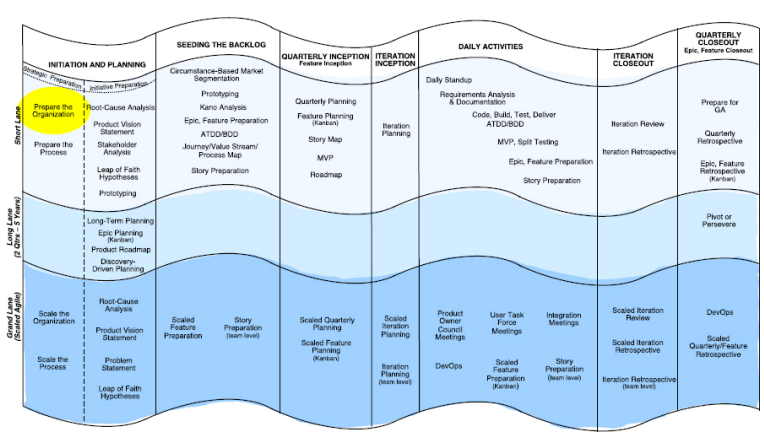

Figure 5.1 Chapter 5 agile analysis and planning activities

As shown in Figure 5.1, we'll be examining Prepare the Organization, an activity performed as part of the first zone, Initiation and Planning.

5.3 WHAT IS INITIATION AND PLANNING?

In Chapter 4, "Overview of Analysis and Planning Activities across the Agile Development Lifecycle," we grouped analysis and planning work into zones of related activities—the vertical columns in Figure 5.1. The Initiation and Planning zone covers initial analysis and planning activities for new product development or a significant change to an existing one. The activities in this zone include organizational preparation, calibration of the analysis and planning process, product visioning, initial stakeholder analysis, and the development of a product roadmap for the venture. This chapter focuses on organizational preparation.

Many of the analysis activities in this book presume that the enterprise is already following an agile approach. For organizations that have not yet reached an acceptable level of maturity in agile development or business analysis, this zone also includes activities to get them there.

If a transformation is required, it's unlikely that it will lie within your official sphere of responsibility, but that doesn't mean it's not your concern. As we'll see in this chapter, your effectiveness as an analyst or consultant lies in the ability to *influence* change to provide better results for the business. This chapter describes *what* changes to champion and introduces some of the approaches you can use to effect change in the organization. For a fuller treatment, I encourage you to explore books devoted to the topic, such as the following recommended titles:

- *Leading Change: Why Transformation Efforts Fail*[1]

[1] John P. Kotter, *Leading Change: Why Transformation Efforts Fail* (Boston: Harvard Business Review Press, 2012).

- *ADKAR: A Model for Change in Business, Government and Our Community*[2]

[2] Jeffrey M. Hiatt, *ADKAR: A Model for Change in Business, Government and Our Community* (Loveland, CO: Prosci Learning Center Publications, 2006).

- *The Heart of Change: Real-Life Stories of How People Change Their Organizations*[3]

[3] John P. Kotter and Dan S. Cohen, *The Heart of Change: Real-Life Stories of How People Change Their Organizations* (Boston: Harvard Business Review, 2012).

- *Changing the Way We Change: Gaining Control of Major Operational Change*[4]

[4] Jeanenne LaMarsh, *Changing the Way We Change: Gaining Control of Major Operational Change* (Upper Saddle River, NJ: Prentice Hall PTR, 1995).

- *Change Management: The People Side of Change*[5]

[5] Jeffrey M. Hiatt, and Timothy J. Creasey, *Change Management: The People Side of Change* (Loveland, CO: Prosci Research, 2012).

- *Master Change, Maximize Success: Effective Strategies for Realizing Your Goals*[6]

[6] Rebecca Potts and Jeanenne LaMarsh, *Master Change, Maximize Success: Effective Strategies for Realizing Your Goals* (San Francisco: Chronicle Books, 2004).

5.4 HOW LONG SHOULD YOU SPEND UP FRONT ON INITIATION AND PLANNING?

As a rule of thumb, the time spent upfront on initiation and planning activities should not exceed 10 percent of the planning horizon. So, if the planning horizon is ten months, as a *general rule*, you wouldn't spend more than one month preparing and planning for it. In practice, take the time you need. But if that turns out to be more than three months, start a separate project for initiation and planning.

At the start of every quarter or significant change, revisit the activities in the Initiation and Planning zone. Make adjustments in light of experience and changing

circumstances. For example, it may be necessary to add feature teams or reallocate resources due to changing demand.

5.4.1 The Greater the Anticipated Risks, the Greater the Need for Upfront Planning

Generally speaking, the higher the risks anticipated for a venture, the more upfront analysis and planning will be needed. For example, a high-cost, fixed-contract initiative with an external solution provider exposes the business to the risk of high, unexpected costs: missed requirements can result in cost overruns, missed launch dates, and risks to the customer experience. To mitigate these risks, you need to extend activities in the Initiation and Planning zone to allow for a comprehensive risk analysis in the plan.

5.4.2 What's Past Is Prologue[7]

[7] "Whereof what's past is prologue." See William Shakespeare, *The Tempest*, act 2, scene 1, line 217, PlayShakespeare.com, https://www.playshakespeare.com/the-tempest/scenes/act-ii-scene-1

Tap into the collective wisdom of the organization regarding similar initiatives when determining how long to spend on preparation and planning activities. Reach out to key individuals who were involved in these activities in the past. Ask them which activities can be safely postponed until development and which, in their view, should be performed up front because the cost of delaying them would be prohibitive to a successful outcome.

> One of the most extreme examples of excessive upfront planning concerned a communications company that asked my company to assess its requirements before it decided on a replacement for its geographic information system (GIS). It didn't

take me long to recognize the primary problem: the upfront analysis had gone on for *six years*, and yet nothing had been decided! As Tom Peters (author of *In Search of Excellence*) puts it, "Have a bias for action." Yes, it's vital to spend the necessary time to identify and mitigate risks. But at the same time, you have to realize that you will never have *all* the facts. At some point, you have to dive in with what you know, because the cost to delay the decision any further becomes too high.

5.5 THE PURPOSE ALIGNMENT MODEL

Before you start developing a new product or capability—and even before creating the vision for a new product—how do you know if developing it is the best strategy? Use Niel Nickolaisen's purpose alignment model,[8] illustrated in Figure 5.2, for guidance on this question.

[8] Niel Nickolaisen, "Breaking the Project Management Triangle," InformIT, August 20, 2009, p. 2, https://www.informit.com/articles/article.aspx?p=1384195

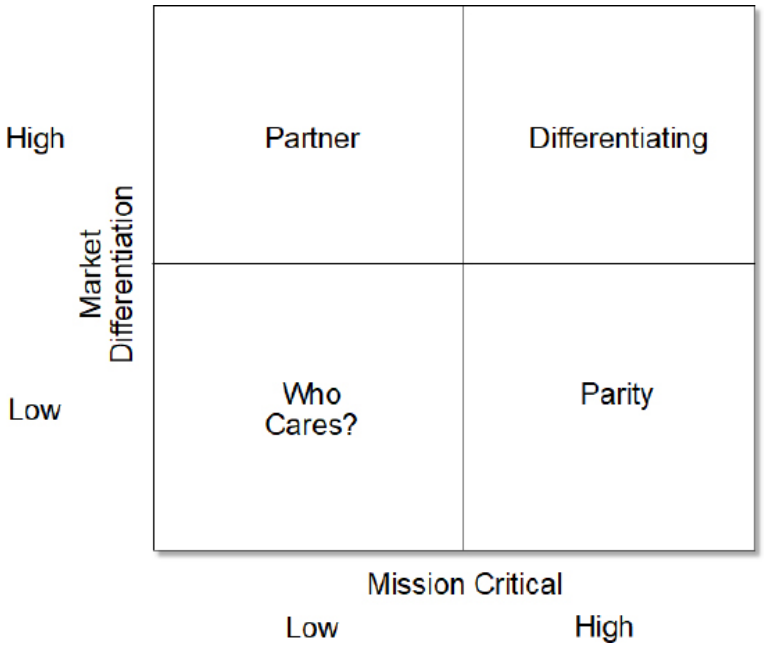

Figure 5.2 The purpose alignment model

The model can be used to evaluate an entire product, service, or capability of a product or service.

If the product or capability already exists and the question is whether to improve it, determine if it is currently at, above, or below parity with similar capabilities available in the market. If it is below parity and improvements are being considered, or it's a new product or capability, determine which quadrant it belongs to, as indicated in Figure 5.2, based on mission criticality and market differentiation. Then use the guidelines for the corresponding quadrant to determine whether to maintain parity, develop the capability in-house, outsource it, or develop it in partnership with another company.

5.5.1 Differentiating Quadrant (Top Right)

The model includes four quadrants based on levels of market differentiation and mission criticality. Use the top-right quadrant in Figure 5.2 for mission-critical products or capabilities that are market differentiators. Items landing in this category should be developed in-house by long-lived teams because they have to be continually improved for the product to remain competitive.

5.5.2 Parity Quadrant (Bottom Right)

The bottom-right quadrant in Figure 5.2 is for mission-critical capabilities that are not core differentiators. *Parity* with the market standard is sufficient for these items, since excellence is unlikely to affect customer choice.

Develop items in this quadrant in-house, or outsource them. Weigh the immediate savings of outsourcing against future costs. Keep in mind that choices made

early on can have a significant impact on the cost of future changes.

As a general rule of thumb, use the following guidelines for capabilities in the quadrant:

- Use an outsourced solution if the product or capability is a novel innovation. Outsourcing enables accelerated learning by providing the ability to test the market for the capability quickly at a low cost.
- Use an in-house solution if uncertainty is low and high volumes are expected—for example, when improving an established product with a large customer base. While upfront costs and time to market are higher with an in-house solution than with outsourcing, total costs will be lower once volumes are high.

One example of a parity capability is eBay's payment processing. Payment processing is a parity capability for eBay because it's mission-critical but not a differentiator. Early on, when the business case for eBay was, as yet, unproven, it outsourced payment processing to PayPal in order to deliver the capability quickly with minimal upfront investment. By 2002, eBay's volume grew to the point where it no longer made financial sense to outsource the service, and the company brought it in-house by purchasing PayPal.

Many businesses face a similar decision regarding cloud services, such as AWS (Amazon Web Services). At the outset of product development, the use of these services speeds time to market, provides scalability, and doesn't require a substantial upfront investment. However, once the business grows, the cost of these services often tips the scale in favor of in-house development.

If an outsourced solution is to be used, explore ways to reduce future costs in case the solution needs to be

replaced when volumes rise. Costs can be significantly reduced by building adaptability into the design from the start, such as through strategies like service-oriented architecture (SOA) and the use of outgoing APIs for messages to third-party services.

5.5.3 Partner Quadrant (Top Left)

Use the top-left quadrant in Figure 5.2 for capabilities and products that are differentiators but outside the company's core mission and competency. For example, consider a training company whose value lies in its intellectual content—the courseware, workbooks, and so on, that it provides to its customers. Suppose that it decides to launch an eLearning platform. The platform is not the company's core competency, but it is a crucial differentiator. The preferred option is to develop the services collaboratively through a long-term partnership with a third party. That option provides a stable basis for ongoing innovation in a critical area without diverting the company from its core mission.

5.5.4 Who Cares? Quadrant (Bottom Left)

Use the bottom-left quadrant for capabilities that have low mission-criticality and market differentiation. Spend as little resources as possible on these capabilities. Consider excluding them from product development due to their low value to the business.

5.6 PREPARING THE INFRASTRUCTURE

For the full potential of agile development to be realized, you have to ensure the right development and testing infrastructure exists—one that supports DevOps and continuous integration and continuous deliver (CI/CD) practices, including automated testing, automated builds, and provisioning. Figure 5.3 illustrates what happens when that infrastructure is lacking.

Unfortunately, I still see this situation in some development organizations.

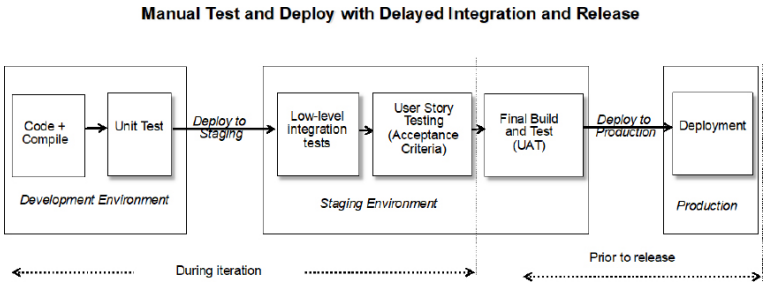

Figure 5.3 Testing without automation

In Figure 5.3, user stories undergo unit tests and low-level integration tests as soon as they're completed. Once they pass those tests, they are tested against their acceptance criteria in the presence of the development team. If the story fails its acceptance criteria or just does not "feel right" to users, it is sent back for immediate rework. Once the story is accepted (and satisfies the definition of done), it is considered done. So far, so good. However, in this scenario, because the final build-and-test steps involve time-consuming manual tasks, they are delayed until the end of the release cycle.

This approach is better than pure waterfall because low-level technical testing occurs continuously and early instead of at the end of the development lifecycle. But it's not the ideal approach because it delays testing under realistic production conditions until the end of the release cycle—a period of up to three months. In addition, this approach doesn't provide the option for the timely release of changes the business wants to implement immediately, such as bug fixes and small enhancements to existing features.

The *recommended* agile approach, instead, is to use DevOps, incorporating CI and CD practices. With this approach, a work item undergoes its build and final testing steps automatically, as soon as the item has been coded and has passed its initial tests. This automation

enables changes to be delivered frequently and reliably to the customer, while actual market deployment is based on the demands of the business. For example, the company may want to hold back a feature until it is mature enough to be competitive in the marketplace or until there are enough supporting features to allow a user to have a better experience.

> For more on DevOps and other technical practices, see Chapter 17, section 17.4.2.1.

5.6.1 Transitioning from Manual to Automated Testing

Test automation is key to enabling a company to deliver changes to the market with speed and confidence. If your organization is transitioning toward automated testing, use the test pyramid in Figure 5.4 as a guideline to determine where to focus the automation effort for maximum benefit.

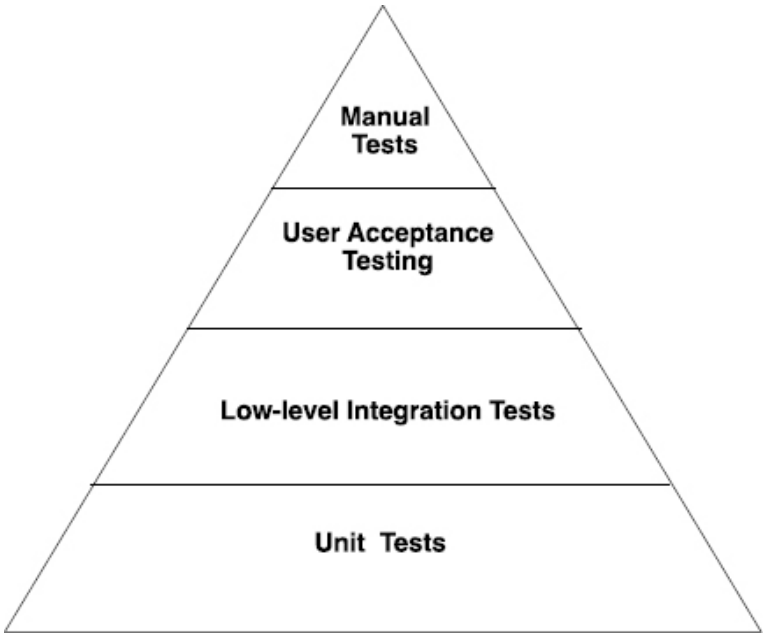

Figure 5.4 Test pyramid

The figure illustrates the relative prevalence of tests by test type. The type with the most tests is at the bottom of

the pyramid. The type with the fewest is at the top. (Other versions of the pyramid, such as Martin Fowler's, may vary from that shown in Figure 5.4.)[9]

[9] Fowler, Martin, TestPyramid, martinFowler.com, Nov. 15, 2017 (originally published May 1, 2012, https://martinfowler.com/bliki/TestPyramid.html

At the top of the pyramid are manual tests. These should be the least numerous because they are the most time consuming and expensive to run.

Next are automated user acceptance tests. There should be fewer automated tests of this type than the types below it on the pyramid. There are several reasons for this:[10]

[10] Fowler, TestPyramid.

- User acceptance testing (UAT) tends to be brittle. Changes to the user interface can easily break existing tests.
- Automated UAT scenarios are expensive to write.
- User acceptance tests have a propensity for nondeterministic behavior (i.e., they provide inconsistent results for the same test, making them difficult to diagnose).

Automated lower-level integration tests verify the connection between one software component and another, such as the connection between a programming unit and an API to which it sends messages. These automated tests should be more numerous than user acceptance tests because they are less brittle. Most automated tests should be unit tests, because

- There are more low-level units to test than higher-level components.

- Unit tests are frequently reused in regression testing and are a critical capability for enabling reliable, continuous delivery.
- Automated unit tests are less subject to change than other test types.

Use the testing pyramid to guide the transition to test automation for software that's *already* been written:

- Focus test automation, at first, on end-to-end UAT (at the top of the pyramid), since you won't need many of these relative to other types of tests.
- Next, focus on automating lower-level integration tests.
- Finally, create automated unit tests (bottom of the pyramid) for existing software units. There are typically so many existing software units that efforts to automate unit testing for some have minimal impact on overall outcomes.

When prioritizing the automation of *new* software units, advance through the pyramid in the opposite direction, from the bottom up. Developers should begin creating automated unit tests and low-level integrations tests as part of their coding process. All new software units should include these tests and pass them before being accepted. Expect testing-automation activities to take up 50 percent of a developer's coding time. As an analyst, you specify automated UAT in feature files by specifying scenarios in the Gherkin syntax.

5.6.2 Timing the Automation of the Build and Distribution Processes

When transitioning to automated build and distribution processes, focus investment on new products, because decisions made early in the product's lifecycle have a disproportionate impact on future automation costs. For example,

- A system designed at the outset to use Web Services (e.g., AWS) for its build and distribution processes can be built and scaled later at much less cost than one that relies heavily on manual intervention.
- New releases of a product designed at the outset as a website are easier to distribute reliably than a product designed as a mobile app or as a laptop application. The former requires updates only at the server end, whereas the latter involves users' participation—introducing unreliability because some users may not participate.

5.7 ORGANIZING DEVELOPMENT TEAMS

I once attended an address given by a vice president of software development before his retirement. "If I had to do it all over again," he said, "I would have organized teams differently." He had realized that he had made a mistake organizing his teams by competency: one team for business analysts, a data layer team, an interface team, and so on. This approach had caused team dependencies that, in turn, led to delays whenever a team had to wait for others before it could deliver stories into production. In this section, we look at the guidelines for forming agile teams that reduce dependencies and optimize productivity.

5.7.1 Guidelines for Forming Agile Teams

Use the following guidelines to form teams that can respond quickly to change.

- Everyone works for the customer
- Foster a whole-team culture
- Maximize self-sufficiency
- Self-organization
- Keep teams small
- Favor full-time membership

5.7.1.1 Everyone Works for the Customer

Encourage a culture where all team members, whether they're business SMEs or programmers, work for the *customer*, not for internal stakeholders or departments.

5.7.1.2 Foster a Whole-Team Culture

Encourage a *whole-team* culture—where the *entire team* is responsible for reaching team goals, and everyone, regardless of job title, shares responsibility for success and failure. Encourage team members to have a collaborative mindset in which everyone contributes to shared goals according to their skill set, not their formal roles.

5.7.1.3 Maximize Self-Sufficiency

Where possible, form teams that include the competencies (skills, knowledge, and abilities) required to deliver value. Include business stakeholders and practitioners on the *same* team. Based on their knowledge and experience, business stakeholders may participate as SMEs, business analysts, testers, or product owners.

By including as many business and technical capabilities as possible within the team, you reduce dependencies and handoffs and accelerate decision making. The cross-functional composition of the team also prevents the emergence of silos and the tendency to create factions.

Note that in many cases, you may need to share resources between teams. For example, there may not be enough work to occupy a UX designer or business analyst on a full-time basis, so that person may be made available to a group of teams. These shared team members form part of the *extended team*.

Some agile frameworks, such as Scrum, advise that team dependencies be eliminated by organizing self-

sufficient teams, where each team contains all the skills required to deliver value.[11] In practice, though, it's usually not possible to remove all dependencies because of several factors:

[11] For example, the Scrum Guide states, "Cross-functional teams have all competencies needed to accomplish the work without depending on others not part of the team." Ken Schwaber and Jeff Sutherland, *The Scrum Guide: The Definitive Guide to Scrum —The Rules of the Game*, Scrumguides.org, 2017, 6, https://www.scrumguides.org

- Features are intentionally designed to be well-integrated. This results in dependencies between the feature teams that support them.
- The product is often too complex for any single team to contain all the competencies required to deliver value.
- Component teams are often formed to maintain commonly used software components. This results in dependencies between feature and component teams.

5.7.1.4 Self-Organization

Each team should make tactical decisions and decide for itself how to carry out the work, while leaving bigger, strategic decisions to senior-level decision makers.

5.7.1.5 Keep Teams Small

Each agile team should consist of about five to ten members. At least some of these members should be jacks-of-all-trades, adaptable enough to pitch in as needed (e.g., performing QA, BA, and UX tasks as required).

5.7.1.6 Favor Full-time Participation

As a general guideline, members should be fully dedicated to one team so they can focus entirely on the team goal and reduce time lost to task-switching between teams. It is likely, though, that some members

will have to be shared, perhaps due to a lack of resources or because there isn't enough work in a competency to justify a full-time person. Cohn recommends that teams form a consensus on simple rules regarding sharing, such as "No person can be assigned to more than two projects" or "Everyone on the team must be at least x% dedicated to the team."[12]

[12] Mike Cohn, *Succeeding with Agile* (Boston: Addison-Wesley, 2010), 196–197.

5.7.2 Organize around Value

Organize teams around value, not competencies, so that each team can deliver value to an end user or customer with minimal dependencies on other teams. Figure 5.5 is an example of teams organized around value in a small development organization.

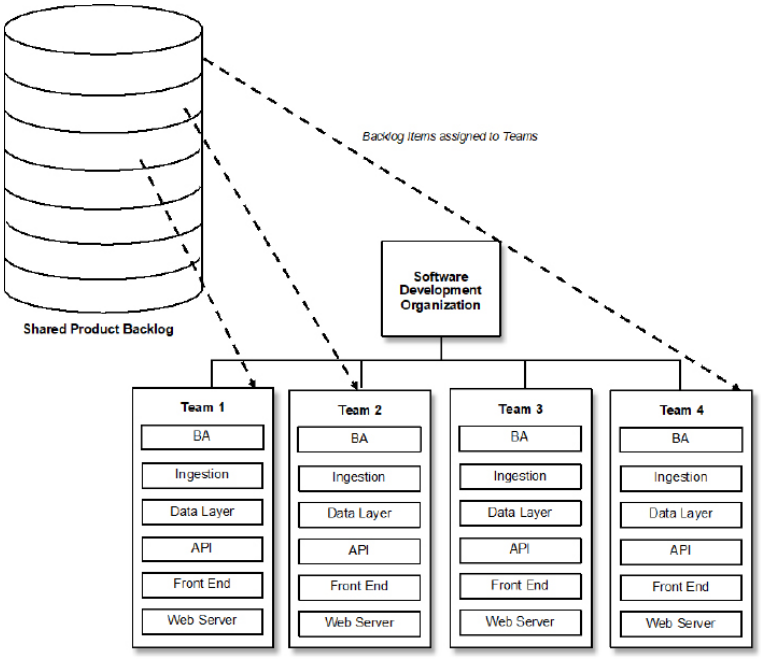

Figure 5.5 Teams organized by value

In the example in Figure 5.5, the product is small enough for each team to be able to contain all the required competencies.

Note that the boxes within each team in Figure 5.5 refer to competencies, not individuals. An individual may have more than one competency, and a competency may be practiced by more than one individual on the team.

The benefit to the business in organizing by value is that it can adapt the product quickly and reliably in response to changing circumstances, new opportunities, and threats.

5.7.3 Feature Teams versus Generic Teams

As a general rule, the best organizational strategy is to create long-lived **feature teams** that each specialize in a product area. A **product area** is a subset of the product representing a high-level use case (usage) or feature set. Over time, the team gains expertise in its product area, resulting in optimized performance and reduced turnaround time.

The exception is during the early stages of new product development when teams should be generic. Generic teams at that time enable managers to allocate resources dynamically in response to changing demand. Once the product has stabilized, though, and demand is more predictable, teams should organize into long-lived feature teams.

5.7.4 The Extended Team

Once a product matures and becomes more complex, you typically can't contain all the required competencies within a small agile team of about ten or fewer members. Fully self-sufficient teams are no longer possible. Instead, teams need to collaborate. They often need to share members, too (as noted in section 5.7.1.6).

Figure 5.6 illustrates the structure of a feature team for a mature product, highlighting dedicated and shared team members. The following is one example. It's not a general prescription. Individual team structures will vary.

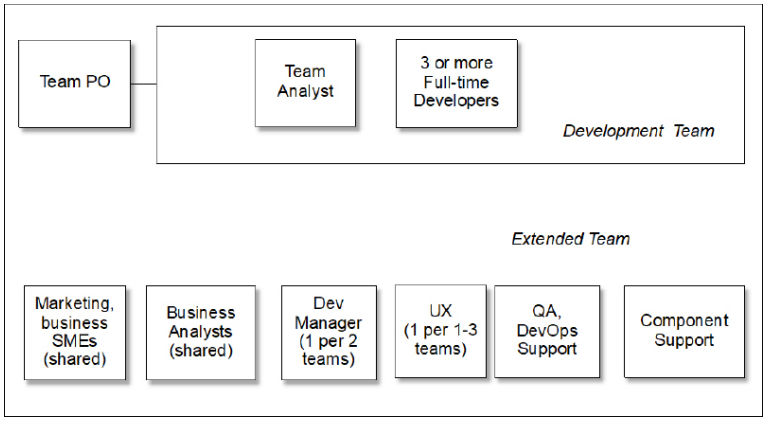

Figure 5.6 Structure of a feature team

The core development team in Figure 5.6 consists of one team analyst and three to four full-time developers. The **extended team** includes shared members: these are marketing and business SMEs, high-level business analysts, a development manager shared among no more than two teams, a UX designer shared among one to three teams, QA and DevOps professionals to coach the team in automated testing and integration, and support for the software components used by the team.

> See Chapter 15, "Rolling Analysis and Preparation—Day-to-Day Activities," section 15.3 for more on organizing teams in scaled agile organizations.

5.7.5 Why Organizing by Competency Is Bad for the Business

As noted earlier in the chapter, it's a common mistake to organize teams by competency instead of by value. Team organization may sound like an issue of little concern to the business or business analyst, but it's not, because poor organization delays time to market.

Figure 5.7 illustrates why. It depicts a development organization for a messaging product, organized by competency.

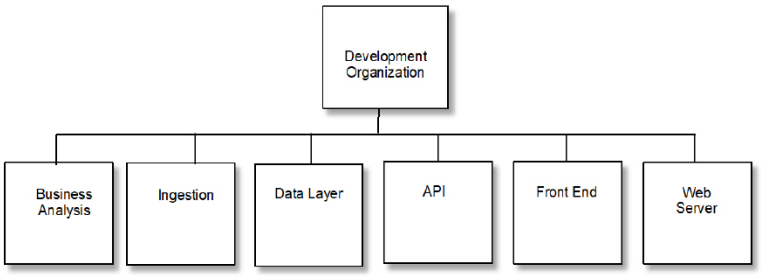

Figure 5.7 Teams organized by competency (not advised!)

In Figure 5.7, there is one team to handle high-level BA and a team for each technical layer of the application: ingestion (loading messages into the app), data layer, API, front end, and Web server. The trouble with this arrangement is that no single team can deliver value on its own. For example, a feature to add a new message type requires

- Analysis by the BA team.
- Work by the ingestion team to load the new message type into the system.
- Work by the data layer team to enable the new message type to be stored in the database.
- Work by the front-end team to allow the new messages to be displayed.

If any of these teams finishes early, the feature can't be released any sooner. The feature is only releasable once *all* the other teams have completed their work. This structure also increases the number of handoffs and approvals per feature. Each handoff introduces a time lag to wait for approvals, to plan, and to allocate tasks for the next team—increasing the overall time it takes to bring change requests to the market. If teams are currently organized this way, champion transformational change to prevent these negative

impacts. As we've seen, that means organizing teams by value into feature teams.

5.8 MANAGING STAKEHOLDER EXPECTATIONS ABOUT AGILE DEVELOPMENT

If you set appropriate expectations about agile development from the start, stakeholders are more likely to take the long view when problems arise—as they inevitably will. Let's look at examples of common stakeholder expectations and how to manage them.

5.8.1 The Incorrect Expectation That Requirements Delayed Are Requirements Denied

A common expectation of stakeholders new to agile development is that requirements that don't make it into the first cycle will never get implemented. Stakeholders have often learned this lesson from negative experiences with waterfall development. The consequence is stakeholders who want to assign high priorities across the board to requirements items. This outcome is anathema to agile development, since the approach is based on the phased rollout of requirements over time.

To head off this problem before it becomes an issue, meet with stakeholders early on to explain the agile trade-off: the customer gets the benefit of shortened time to market and continuous delivery of features instead of the long lag time that is typical of waterfall processes. However, to obtain those benefits, business stakeholders have to prioritize some requirements over others because agile development is, by nature, incremental. Another advantage of agile development to stakeholders is that they can change their minds at any time and reprioritize requirements based on changing circumstances.

> To help stakeholders focus on what's most important, *ask them to imagine what core features they would want to deliver if the initiative ran out of time or money at some point in the development cycle*.

Sometimes it helps to flip the terminology: instead of speaking about *prioritizing* requirements, talk about *sequencing* them—reframing the conversation from one about *values* to a practical one about *problem solving*.

Make sure stakeholders also understand that they don't have to make prioritization decisions on entire features. For example, they may decide to implement some stories of Feature X first, followed by some stories for Feature Y, followed by more stories for Feature X.

There's nothing more persuasive than the evidence of one's own eyes, though, to convince stakeholders that agile's incremental approach works and that requirements delayed are not requirements denied. Provide opportunities for stakeholders to meet teams and customers who have been using agile development successfully. Arrange for experienced agile teams to invite new stakeholders to sit in as observers during product demonstrations.

5.8.2 Productivity Expectations

A few years ago, I was speaking with an executive about his company's planned agile transition. He told me that stakeholders were very enthusiastic about "agile getting their products to market much quicker." I should have been pleased, but I was concerned. Business stakeholders sometimes mistake agile's promise of *early delivery* to mean that they can get *all* the requirements they want in much less time than waterfall. The reality is not so straightforward. For example, in Chapter 2, "Agile Analysis and Planning," I quoted a QSM study that found that agile projects received a 16 percent

productivity increase compared to the industry average.[13] It is a significant benefit—but far from the expectations of "several 100 percent" gains in productivity made in the early days of Scrum.[14]

[13] QSM Associates, *The Agile Impact Report, Proven Performance Metrics from the Agile Enterprise* (Boulder, CO: Rally Software, 2015), 5, https://www.rallydev.com/sites/default/files/Agile_Impact_Report.pdf. Productivity was measured as an aggregate of tooling and methods, technical difficulty, personnel profiles, and integration issues.

[14] Ken Schwaber and Mike Beedle, *Agile Software Development with Scrum* (Upper Saddle River, NJ: Prentice Hall, 2002), viii.

On the other hand, the same study found that "Agile Projects are 37% faster to market than [the] industry average."[15] How can agile development result in faster time to market without corresponding increases in productivity? The answer lies in *what* is delivered: agile development enables a business to bring *something* to market quickly—but not the whole wish list. What agile development delivers quickly is a *minimum marketable product* with a limited set of features. Make sure stakeholders understand that the approach speeds time to market by focusing effort on the *highest-value* features, not by delivering the *same* set of features in less time. The value of agile development lies in learning that *not all the requested features are actually needed*—or that less-costly versions will be able to get the job done just as well.

[15] QSM Associates, *The Agile Impact Report*, 4.

Another common but incorrect expectation is that productivity improvements will come quickly as an organization transitions to agile development. More commonly, it can take a few months to a few years for teams to get good enough at agile development for improvements to be realized.

Stakeholders also can have mistaken expectations about what that will be delivered frequently. Many expect that software updates will be delivered to the *market* at least every one to two weeks (the length of a typical Scrum sprint). That may be the case for some updates, but it's often not so for major features. Although deployment to a *testing* environment occurs continuously, it can often still take two to three months to deploy significant changes to the *market*. The benefit of agile development, in this case, is not reduced time to market but that the delivered solution will be

- More *relevant*—since it is based on current priorities and requirements.
- More likely to *delight the user*—since it relies on user feedback throughout the development process.
- More cost-efficient—because it weeds out low-value requirements and eliminates administrative waste.

5.9 PREPARING THE CUSTOMER–DEVELOPER RELATIONSHIP

In Chapter 1, "The Art of Agile Analysis and Planning," you read about a company whose meetings kept ending in acrimony because business stakeholders didn't believe the developers' estimates. The problem, at its heart, had to do with the relationship between business stakeholders and the development organization. In this instance, the relationship was contractual. In a truly agile organization, it's collaborative, with both sides working together to problem-solve as circumstances change.

Prepare both parties in the relationship by gaining early consensus regarding norms of behavior. Following XP guidance, coach them to codify this consensus in a bill of rights and responsibilities for the customer and the development team. The *customer*, in this context, refers to the entity representing customers and stakeholders (e.g., a Scrum product owner (PO) or a customer team).

5.9.1 Customer's Bill of Rights and Responsibilities

Following is an example of a customer bill of rights and responsibilities.

Customer Bill of Rights

- The customer has ultimate responsibility for managing and prioritizing the stories in the product backlog, subject to the following constraints:
  - The customer does so with the approval of stakeholders and in collaboration with development and operations.
  - If a conflict can't be resolved through collaboration, a higher-level role may be called upon to make a decision.
- The customer may add, drop, and change stories and acceptance criteria.
- The customer may change priorities and functionality without undue penalties.

If the organization is using iterations for planning (e.g., as indicated by Scrum), these rights are subject to the following limitations.

The customer

- May not add stories, acceptance criteria, or functionality to an iteration once it begins unless the development team agrees.
- May not change the length of an iteration once it begins.

Customer Responsibilities

The customer agrees to

- Accept the estimates of the development team.
- Participate in team planning.

- Respond to developers' questions in a timely fashion.
- Collaborate with developers to explore options.

5.9.2 Developers' Bill of Rights and Responsibilities

Following is an example of the developers' bill of rights and Responsibilities.

Developers' Rights

The development team

- Has the sole right to make and update estimates.
- Has the right to add technical stories to the backlog.

Developers' Responsibilities

The development team agrees to

- Deliver a potentially shippable integrated increment with each work item.
- Deliver a shippable increment at least every iteration if continuous delivery is not supported.
- Accept the priorities assigned by the customer to backlog items.
- Be transparent with the customer about progress.
- Alert the customer if goals are endangered.
- Collaborate with the customer to explore alternatives and trade-offs.

5.10 AGILE FINANCIAL PLANNING

Prepare financial planners for changes in the way planning is performed on agile initiatives in contrast to traditional waterfall projects. Following is an overview of those differences.

5.10.1 Measuring Success

In traditional waterfall project management, you measure success by the degree to which the solution delivers specified outputs (the *deliverables*) at a given cost within a specified time. You can't use this approach on agile software initiatives because there is no baselined set of requirements to be delivered: by intention, agile requirements are subject to change at any time.

Agile initiatives measure success on the basis of desired business *outcomes*[16] rather than prespecified outputs. You don't ask, "Did the customer receive a product with the specified features on time and budget?" Instead, you ask, "Did the customer increase market share as much as expected?" or "Did it get the expected return on investment?"

[16] Michael F. Hanford, "Program Management: Different from Project Management," IBM Developer Works, May 14, 2004, http://www.ibm.com/developerworks/rational/library/4751.html

5.10.2 Discovery-Driven Financial Planning

Traditional financial planning approaches rely on past data to make projections into the future. This approach doesn't apply when the product is novel because there is no historical or trend data to draw from. In such circumstances, financial planning methods need to take uncertainty into account and bake it into the plan.

One approach for doing that is **discovery-driven planning**. With this method, instead of projecting an expected bottom line from historical trends, you begin by specifying the bottom line that must be achieved for the venture to be successful. Then you work *backwards* to identify the assumptions you need to make to get there (e.g., assumptions about the price customers will pay and support costs). These assumptions then drive the

planning process: you create a plan to test critical hypotheses as early as possible.

> For more on discovery-driven planning, see Chapter 18, section 18.10.2, and Appendix B.

5.11 PREPARING THE MARKETING AND DISTRIBUTION TEAMS

If the marketing team is new to agile development, you'll need to help it prepare for changes resulting from the approach. Prepare marketers to expect that only the release date and goals may be committed at the start of a release cycle; the features may not be determined until close to the scheduled release date. As a result, marketers may not be able to specify the content of advertising copy until close to the release date.[17] This is in contrast to traditional waterfall development, where marketing teams obtain a list of committed features with a comfortable lead time.

[17] Cohn, *Succeeding with Agile*, 39.

In many organizations, this plays out as follows: The marketing group sets the release date based on business conditions. It sends marketing representatives to the feature teams as extended team members. They collaborate with developers to establish the lead time required before a general release to allow for marketing, testing, and deployment activities. They then work backwards from the release date to set a cut-off date that provides the necessary lead time. When the cut-off date arrives, only those features deemed *done* are included in the release and promoted in advertising.

Prepare the marketing team for changes at the front end of development as well. Explain that they will need to use new methods to analyze the market because traditional research methods don't apply to novel products. The product class is so new that there are no comparable products to compare it to, and customers

can't reliably predict what they'll want. Discuss the use of **circumstance-based market segmentation**. The approach is based on the insight that "People don't want to buy a quarter-inch drill. They want a quarter-inch hole."[18] In other words, they're not really looking for a product; they're looking to get a job done. Following the approach, researchers recruit customers in the field and interview them to learn what jobs they are trying to accomplish, the constraints that influence their choice of product, and what other products they've sometimes purchased for the same job.

[18] Harvard Business School professor Theodore Levitt, as quoted in Clayton M. Christensen, Scott Cook, and Taddy Hall, "What Customers Want from Your Products," *Working Knowledge*, January 16, 2006, https://hbswk.hbs.edu./item/what-customers-want-from-your-products

> For more on circumstance-based market segmentation, see Chapter 8, "Seeding the Backlog—Discovering and Grading Features," section 8.4.

5.12 PREPARING CHANNELS AND SUPPLY CHAINS

If the company is an established incumbent developing a novel product, it should plan for changes to its existing distribution and supply channels. An established company's current market, distribution channels, and supply chains often are not appropriate for the new product and must be developed in an iterative, agile manner, just like the product itself.

5.13 PREPARING GOVERNANCE AND COMPLIANCE

Governance is concerned with overseeing an initiative by monitoring its progress and expenditures to determine how well it is adhering to its plan. *Compliance* is concerned with ensuring the initiative conforms to standards and guidelines—both internal to the company and external (e.g., ISACA/CMMI Institute's Capability

Maturity Model Integration [CMMI], Institute of Electrical and Electronics Engineers [IEEE], Information Technology Infrastructure Library [ITIL], and Sarbanes-Oxley Act [SOX]).

Agile development includes some practices that, at least at first glance, appear to run counter to governance and compliance in traditional organizations. Examples of these agile practices include minimal documentation, lack of formal sign-offs and the postponement of decisions to the last responsible moment. Despite this apparent conflict, agile development can usually be brought in line with governance and compliance regulations if the appropriate preparations and adaptations are made.

If governance and compliance officers are new to agile development, use the following strategies to prepare them for an agile initiative.

5.13.1 Challenge Compliance Assumptions

Be willing to challenge assumptions about compliance rules. There is often much more room for give-and-take than you might first expect. The following real example is instructive: A company was planning to use a third-party vendor for new product development. The auditor was concerned that the vendor might change agreed-upon requirements. This concern was expressed in the following rule: “If a vendor logs in to the requirements repository, the vendor is prevented from changing requirements.” The lead analyst determined that the rule would not be easy to enforce, so he pushed back. They agreed that only a “controlled environment” would be needed for compliance—and that the unique, trackable login identification already provided by the repository on changes was sufficient.

5.13.2 Do Compliance *After* Process Design

One of the most useful pieces of advice regarding compliance and agile development comes from Andre Franklin, a process and systems analysis expert in drug research and development services—a highly regulated sector—who provided this guidance: Don't have auditors determine what the process should be. Design the process first the way you want it. Then bring the process to auditors for approval. That way, practitioners do what they do best—design processes—"and auditors do what they do best—audit processes for compliance."

5.13.3 Focus on Goals, Not Means

Another effective strategy is to argue that though an agile process may use nontraditional lightweight methods, it still achieves the goals behind compliance regulations.

For example, it is sometimes assumed that testing standards strictly govern the testing process, whereas they often only define *goals* for testing; that there must be a well-defined testing process in place; that it must be followed; and that it must lead to a low defect rate. There is often a great deal of latitude, however, on *how* to achieve those goals. The DevOps automated environment used in agile development often achieves the goals of standards, because it results in high levels of testing, leading to low defect rates. For example, Cohn reports that CMMI compliance up to level 5 has been achieved with agile frameworks.[19]

[19] Cohn, *Succeeding with Agile*, 400. Cohn lists Philips Research and Systematic as examples of companies achieving CMMI compliance with agile processes.

The same is true concerning documentation: while the *types* of documents required for compliance are often nonnegotiable, there is often flexibility regarding the

form the documentation takes. Even minimalist documentation, in the form of photo images of whiteboard designs, can be sufficient to satisfy compliance regulations.

Concerning ISO 9001, a study of agile organizations concluded that "an agile process [can] produce documents that can be used (1) as proof of conformance and that (2) can be reviewed as part of ISO 9001's verification and validation" and that "there will be no problems whatsoever when a company wants to use agile development and still keep its ISO 9001 certificate."[20]

[20] Tor Stålhane and Geir Kjetil Hanssen, "The Application of ISO 9001 to Agile Software Development" (conference paper, Product-Focused Software Process Improvement, 9th International Conference, PROFES 2008, Monte Porzio Catone, Italy, June 23–25, 2008), p. 14.

5.13.4 One-Time Experiments

If you aren't successful at gaining governance and compliance adaptations for agile development at the enterprise level, try negotiating such governance changes as one-time experiments.

5.14 PREPARING FOR INCREASED DEMAND ON RESOURCES

If the organization is new to agile development, engage functional groups and alert them to expect an increased demand for resources to support agile methods. If the resources are required across multiple initiatives and projects, the organization may need to follow an extensive process to review the request against workforce capacity. A central body such as the project management office (PMO) may need to be called in to help with the allocation of talent.

Following is an example of the number of resources that might be expected to support an agile development

initiative (individual cases will vary):

- One business stakeholder, who is comfortable making decisions, for each team of five to ten people, acting in the role of team-level PO.
- Business SMEs, as needed, dedicated to feature teams or shared among a group of teams as extended team members. These individuals must be prepared to make themselves available throughout the development cycle rather than only at its ends, as is the case with waterfall development.
- Business analysts dedicated or shared among a group of teams.

In addition to these requirements, technical competencies will also be required. They may be fully dedicated to a team or shared as part of the extended team, as indicated in the example in Figure 5.6.

5.15 PREPARING AN ENTERPRISE FOR AGILE DEVELOPMENT

In the preceding sections, we focused on preparation for a specific agile initiative—a new product or significant capability. But these guidelines presume that the organization already has some degree of maturity using agile as an approach for product development. If the organization is not yet there, you, as a business analyst, should take steps to influence change. (This added work will not be necessary, however, if the organization's maturity level is high.) Start the process by preparing the business case for transitioning to a higher level of maturity in agile development and analysis. Use the recommendations in this chapter to identify the current and target levels of agility. Include industry statistics and the benefits of agile development, and find an executive who will champion the effort.

A critical strategic planning activity when transitioning to agile development is the development of a change management approach. It should describe how the change will be communicated to the organization, how people will be trained in agile techniques, how resistance to change will be managed, and how roles and responsibilities will be defined. If you're a team-level analyst, these issues are unlikely to fall within your formal sphere of duties, but they do fall within your sphere of influence. That's because, as an analyst, your role is to be an agent of change within your organization. It is your ability to influence and convince others (e.g., through stealth change management) that makes you effective. That power often extends beyond your formal duties.

I feel passionately about this issue because so much of the organizational change I have been involved in has been due to visionary BA practitioners wielding outsized influence because of the high regard in which they were held by their peers. It's worth noting, though, that while you, as an analyst, are a *catalyst* for change, the change must be supported by a coalition of committed executives across the organization and by representatives at all levels for it to be sustainable—and this support must be consistent and long-lived. Without a broad and deep commitment, it's difficult to maintain successes over the long period, and old habits eventually return—an unfortunate situation I have seen in other organizations.

If the organization's level of maturity in agile development is already high, a full change management process may not be needed. If an extensive transformation is called for, then a process should be instituted with a full-time change management lead. The following sections describe some of the activities to consider in your change management approach.

5.15.1 Agile Fluency Model

The first step in getting from a current to a desired level of agility is to determine what those levels are. Use the agile fluency model, devised by Shore and Larsen,[21] to grade your organization's current and target levels of agile fluency. It characterizes each successive level as a zone, as follows:

[21] James Shore and Diana Larsen, "The Agile Fluency Model: A Brief Guide to Success with Agile," ThoughtWorks, March 6, 2018, https://martinfowler.com/articles/agileFluency.html

- Zone 1: *Focusing*. Teams produce business value.
- Zone 2: *Delivering*. Teams deliver value on market cadence (i.e., based on demand from the market).
- Zone 3. *Optimizing*. Teams lead their market.
- Zone 4. *Strengthening*. Teams make their organizations stronger.

Zone 4 is the highest agility level, but it's not necessarily the correct destination for every organization. Your goal should be to choose the right target zone (or zones) for the situation. The entire enterprise doesn't need to set the same target.

5.15.1.1 Zone 1: Focusing

Assign the Zone 1, Focusing, level if teams have absorbed agile fundamentals and can work collaboratively and transparently. These teams are benefiting from agile practice, but their processes are not repeatable. Zone 1 is an appropriate target for teams at the start of an enterprise transition when they are experimenting with the process. This zone is also a sufficient target level for teams working on short-lived software systems.

5.15.1.2 Zone 2: Delivering

Assign the Zone 2, Delivering, level if processes are repeatable, benefits are sustainable, and productivity and adaptability are high. This zone is an appropriate target

for teams that will be working on systems or products that will be enhanced over a long period.

5.15.1.3 Zone 3: Optimizing

Assign the Zone 3, Optimizing, level if the organization is developing disruptive products and services. Advancement to this level typically requires structural change.

> See Chapter 17, section 17.7, for guidance on structuring organizations in this zone.

5.15.1.4 Zone 4, Strengthening

Zone 4, Strengthening, is for organizations using innovative approaches to management theory and practice.

> See Chapter 18 for guidance on applying agile approaches at the enterprise level.

5.15.2 Transitioning the Team

If the development team has not practiced agile development before, a coach should mentor the team for the first couple of months, after which you (the analyst) or another full-time team member, such as a ScrumMaster, should take over the role. At this time, the coach moves to the sidelines, returning to work with the team as needed.

The aim should be to advance the team's agile fluency level to the desired target zone, according to the agile fluency model. The focus should be on improving the structural aspects of the team. The Tuckman model,[22] which describes the stages of a maturing team, may be used to guide these improvements at the team level. Teams should be guided through the *forming* stage (where there is a high dependence on the leader), to *storming* (where team members jostle for position), to

norming (where consensus forms), to *performing* (where the team is united in purpose and self-sufficient).

[22] Denise A. Bonebright, "40 Years of Storming: A Historical Review of Tuckman's Model of Small Group Development," *Human Resource Development International* 13, no. 1 (2010): 111–120, https://doi.org/10.1080/13678861003589099.

5.15.3 Transition Activities at the Enterprise Level

If the organization is new to agile development or is practicing it but struggling with the approach, a full change management process will be needed to improve agile practices across the enterprise. The following sections provide guidelines for creating and accelerating that change according to Kotter's eight-step process.

5.15.3.1 Kotter's Eight Steps for Accelerating Change

John Kotter has described the following eight steps for accelerating change in an organization:[23]

[23] John Kotter, *8 Steps to Accelerate Change in Your Organization: With New Insights for Leading in a COVID-19 Context*, Kotterinc.com, 2020, pp. 10–38, https://www.kotterinc.com/wp-content/uploads/2020/06/2020-8-Steps-to-Acceperate-Change-eBook-Kotter.pdf

- "Create a sense of urgency." Generate and sustain a sense of urgency about the transformation. Identify and communicate a big opportunity that is available today but may be gone tomorrow. Explain what steps need to be taken to get there, the advantages in succeeding, and the costs of failure.
- "Build a guiding coalition." Create a diverse coalition to guide the transformation. Seek out change agents. The alliance should represent all levels of the organization, functions, and locations.

- "Form a strategic vision and initiatives." Strategic initiatives are "targeted and coordinated activities that, if designed and executed fast enough and well enough, will make your vision a reality."[24] The vision communicates how the future will improve upon the present, motivates product developers and stakeholders, and helps them align their efforts toward a common purpose.

[24] Kotter, *8 Steps*, 18.

- "Enlist a volunteer army." Create a movement of people within the organization motivated by the vision for change. Kotter reports that 15 percent of an organization adopting the change is enough to generate momentum. At 50 percent, the change is sustaining.[25]

[25] Kotter, 24.

- "Enable action by removing barriers." Remove bureaucratic practices that deter and decelerate change. Find out which barriers have stood in the way of past improvement efforts.
- "Generate short-term wins." Generate and communicate tangible, small wins that customers and stakeholders care about in order to motivate volunteers.
- "Sustain acceleration." Sustain a sense of urgency over the long term until the vision is achieved.
- "Institute change." Create lasting change. Where existing frameworks are effective, graft the changes onto existing structures. A center of excellence (CoE) supports, disseminates, and sustains best practices.

The approach used to integrate agile practices depends on the organization's agile maturity level. If the organization is successfully using agile methods, the integration will have already occurred. If the organization is using agile methods but is struggling,

coaches should be brought in to identify and improve practices, and a CoE should be created (if one does not exist) to promote best practices. A change management process should be instituted to determine the target agility level (or levels) and manage the transition of the organization to the desired future state.

5.15.3.2 Organizations with No Agile Experience

If the organization has no experience, institute a full change management process for the transformation with a lead change manager to head the transformation. A CoE should be created to support and sustain the change.

The best results come from using both a bottom-up and top-down approach to the transition:

- Prove the value proposition with a few pilot initiatives.
- Gain executive support at the highest levels of the organization, up to the CEO, in order to communicate the strategic importance of the transformation to the company. Develop a community of volunteers as described by Kotter to drive the change (see section 5.15.3.1).
- Let the first teams experiment with practices to learn what works best and when.
- Communicate and sustain best practices through tools, techniques, and training provided by the CoE.

The ideal pilot initiative is small and mostly self-contained—able to be implemented by a team or small group of teams with minimal inter-team dependencies. Give the first agile teams a high degree of freedom to experiment with agile development frameworks and practices. Encourage them to hold open houses to share their experiences and successes. Iteration reviews are an excellent opportunity for these events.

During the broader rollout, teams should focus on improving processes so they are repeatable and sustainable. As the lead on agile analysis and planning, you support this objective by developing techniques, tools, practices, and processes in the competency. These tools should be disseminated and maintained by a CoE in business analysis or agile development.

Don't aim for uniformity across teams. For example, teams working primarily on customer-generated requests may choose a Kanban process because it is optimized for learning. In contrast, teams working on strategic product-wide initiatives may choose a timeboxed approach because it ensures that all teams will be free at the same time to make commitments.

In the final phase, extend the agile transition out to the development organization as a whole. During this phase, establish a standard set of agile analysis and planning techniques, tools, practices, and processes.

5.15.4 Transition Timeline

It can take six months or more to get good enough at agile development to begin to realize improvements. The timeline for a complete enterprise transition varies widely according to the organization's culture, structure, and level of competency in agile development. To give you one example of how such a transformation can play out, the following is a case study, as it unfolded for one of my clients. The client reported these results at the end of Year 2 in the timeline.

5.15.4.1 Initiation (Year 0)

At the start of the transition, the total number of teams in the organization was one thousand. Five hundred of these were software teams. Aside from a few teams, none were agile. The plan was to transition all one thousand teams.

5.15.4.2 Years 1 and 2

The transition began with a single business area. At the six-month mark, the rest of the enterprise started to transition. By the end of the second year, the organization had transitioned at least seventy teams to agile development.

5.15.4.3 Year 3

In Year 3, the organization plans to bring a further thirty-five teams to Zones 1 and 2 of the agile fluency model. During this time, it will be focusing on the standardization of practices.

5.15.5 Communications Plan

The change management plan should contain a communications plan. The plans should be supported by an active and committed coalition of sponsors and volunteers. Measures in the change management plan, such as stakeholder analysis and the identification of resistance, should drive the communications plan.

The communication plan should describe the communication methods to be used. Aim to optimize the quality of communications within the team, between teams, and between teams and business stakeholders. Use the highest-quality mode possible. In order of preference, these modes are

- In-person conversation.
- Video call.
- Virtual meeting.
- Asynchronous communication: Methods include wikis, SharePoint, messaging services, email, and updates made through software applications and services.
- Conference call (audio).

Teams are increasingly synchronizing their efforts through the asynchronous approach: when CI/CD is practiced (as is advised), then updates to version control and code repositories serve to communicate changes across teams. Changes to requirements, their priorities, and their statuses can also be tracked through requirements management tools.

If teams are colocated, use in-person communication as your preferred method of contact; use the lower forms only when necessary. If teams are distributed, favor video calls because members can see who's speaking and read body language. The next best option is Web conferencing that enables shared presentations and communication via text or voice. Examples are WebEx and GoToMeeting. This is a good option if the conversation is mostly one-way (e.g., a status update). Use telephone conference calls only as a last resort.

5.15.6 Agile Enterprise Transition Team

If there is no other body in place to do so (such as an agile CoE or a PMO), create an agile enterprise transition team to champion agile practices within the organization and provide resources for those who need assistance applying them. The team should meet regularly (about every two weeks) and keep its own backlog of work items. More information on the enterprise transition team can be found in *The Enterprise and Scrum*, by Ken Schwaber.[26]

[26] Ken Schwaber, *The Enterprise and Scrum* (Redmond, WA: Microsoft Press, 2014).

5.16 DETERMINE ORGANIZATIONAL READINESS

The following checklist summarizes many of the organizational issues discussed in this chapter. It may be used to verify whether an organization is ready for agile development and to identify readiness gaps that still need

to be addressed. The evaluation should be made by a change manager, the PMO, or a CoE with the input of seasoned agile practitioners.

5.16.1 Organizational Readiness Checklist

Use the following checklist of key questions to determine organizational readiness for agile initiatives.

- Does the culture invite change and innovation and support a customer focus?
- Does the culture actively encourage participants to challenge assumptions, decisions, and the status quo?[27]

[27] Thanks to Ron Healy for suggesting this question.

- Does the current organizational structure minimize inter-team dependencies?
- Do IT and the business sit on the same teams?
- Do IT team members report primarily to the business side (vs. IT)?
- Are team members encouraged to contribute any way they can regardless of job title?
- Do people from the business side feel comfortable leading development teams and making decisions for them? Are there enough of these people available for one to be allocated per agile development team (five to nine members)?
- Are all parts of the organization (funding, marketing, supplier relationship management, etc.) prepared for frequent rollouts and ongoing changes to the requirements?
- Is the technical infrastructure in place to facilitate accelerated time to market? Is test automation supported? Are the build and deploy processes primarily automated?

5.17 CHAPTER SUMMARY

Here are the key points covered in this chapter:

- Use the purpose alignment model—a chart mapping criticality against market differentiation—to develop a high-level strategy for new features (e.g., whether to develop features in-house, through partnerships or outsourcing).
- Organize teams by *value*. Each team should be as self-sufficient as possible, self-organizing, and small (five to ten members)
- Establish an agile culture through customer and developer bills of rights: the customer is the sole person responsible for managing and prioritizing the stories in the product backlog; the development team has the exclusive right to make and update estimates.
- Use circumstance-based market segmentation to analyze the jobs customers might want to use the product for. Organize teams around those jobs.
- Integrate and test continuously, but expose features to the market based on the demands of the business.
- Use the agile fluency model to grade an organization's level of agility as focusing (teams produce business value), delivering (value delivered on market cadence), optimizing, or strengthening.

5.18 WHAT'S NEXT?

This chapter focused on *organizational* preparations for agile development. In Chapter 6, "Preparing the Process," we focus on preparing the *process*—that is, the steps to set up the agile analysis process for an upcoming initiative.

Chapter 6. Preparing the Process

In Chapter 5, "Preparing the Organization," we examined organizational preparations for agile development. This chapter covers process preparation and offers guidelines for selecting the planning approach, agile framework, techniques, and artifacts for the initiative. The chapter describes how to tune or calibrate the chosen process by customizing the product backlog, including the specification of product backlog item attributes, such as cost of delay and weighted shortest job first (WSJF). It also includes guidelines for planning requirements traceability, specifying the definition of done (DoD) and definition of ready (DoR), forecasting team capacity, and setting Kanban work-in-progress (WIP) limits.

6.1 OBJECTIVES

This chapter will help you

- Determine the right mix of agile practices for the circumstance, including considerations for low-risk and high-risk initiatives.
- Set up the product backlog for new product development.
- Determine the attributes that will be used to describe backlog items.
- Prepare for requirements traceability.
- Specify DoD, DoR, and initial Kanban parameters (e.g., WIP limits).
- Forecast initial team capacity (velocity).

6.2 THIS CHAPTER ON THE MAP

Figure 6.1 highlights the activities covered in the chapter.

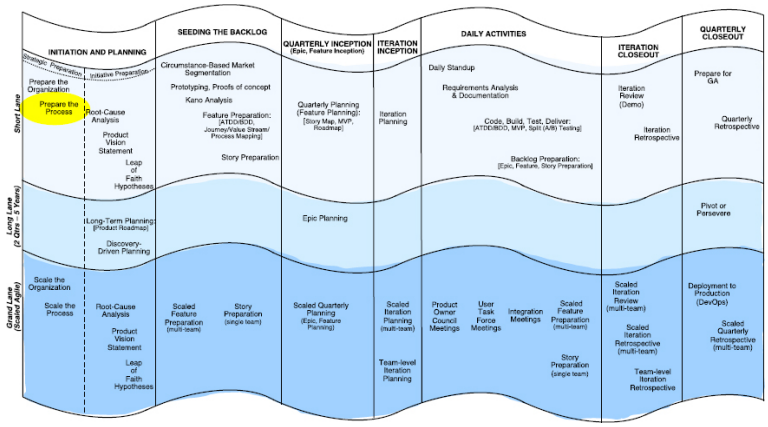

Figure 6.1 *Chapter 6 on the map*

As shown in Figure 6.1, we'll be examining *Prepare the Process*. This activity is performed as part of the first zone, Initiation and Planning.

6.3 PROCESS PREPARATION

Before embarking on new product development, numerous issues about the process must be addressed. Should teams use a flow-based or timeboxed approach to planning? Which estimation units should they use—story points or real-time estimates? Should teams use a feature DoR, and if so, what conditions should it specify? These are the issues included in the activity Prepare the Process—the subject of this chapter.

In the following sections, we look at two sides of this preparation: tailoring and tuning the process. **Tailoring** means fitting the agile analysis and planning process to the circumstance (e.g., selecting the right agile framework, techniques, and artifacts). **Tuning** means calibrating the chosen process to optimize desired outcomes. Tuning activities include setting up the product backlog, determining the PBI (requirements item) attributes that will be tracked, planning how requirements items will be traced, specifying the DoD and DoR, and setting up a Kanban board to manage the workflow of requirements items (features and stories).

6.4 TAILORING THE AGILE PRACTICE TO THE CONTEXT

Agile development is not a one-size-fits-all proposition: practices that work well in one situation may not work well in another. To *tailor* the practice is to adapt it to its context—taking it in a little here, by pulling back on an agile method, and taking it out a little there, by extending it. For example, on a fixed-contract project, you may choose to customize a hybrid analysis and planning process—one that uses a continuous deployment approach at the back end to deliver reliable improvements quickly to the market but a waterfall approach at the front end to mitigate risk. You can't use a one-size-fits-all process because there are benefits *and* costs to agile development, and they change depending on the context. In any given circumstance, you need to tailor the process to maximize the net benefit. Let's begin with the potential costs.

6.4.1 Costs of Agile Development

In agile development, the initial implementation is based on an incomplete understanding of the requirements, with the expectation that the requirements will be clarified over a number of cycles analysis, implementation, and testing. This rework adds to the cost of product development relative to a waterfall approach, where implementation occurs only after a comprehensive analysis.

Another drawback of agile development is the cost to clean up—or **refactor**—software to remove deficiencies, such as redundancies and unreachable and inefficient code. Agile development creates refactoring work because early versions of the software are often meant to deliver learning and quick wins, not market-level quality or maintainability. Agile practitioners can, however, minimize this effect by refactoring continuously rather

than allowing the work (also known as *technical debt*) to build to the point where it is difficult to manage.

6.4.2 Benefits of Agile Development

Agile development—and agile analysis and planning in particular—reduces costs because it enables the business to quickly determine a product's most valuable features and focus investment there, where it will have the most impact. Agile analysis also leads to *reduced* rework, because it occurs at the last responsible moment—as close as possible to implementation when it is most current. In contrast, waterfall analysis is more likely to be out of date by the time it's used because it is performed and completed up front.

Agile development also improves the business's ability to respond quickly to change because its methods shorten time to market. Another benefit of the approach is that it creates solutions that match or exceed expectations because it incorporates continuous feedback throughout the product's lifecycle.

6.4.3 Finding the Best Trade-off of Costs and Benefits

The costs and benefits discussed in the preceding sections will vary by circumstance. For example, iterative-incremental development provides a higher net benefit if there is extreme uncertainty about the product than if it is well-established. The challenge in any situation is to find a mix of practices that provides the best trade-off of costs and benefits.

Figure 6.2 illustrates how context affects the overall target level of analysis and planning agility from lowest to highest expected agility.

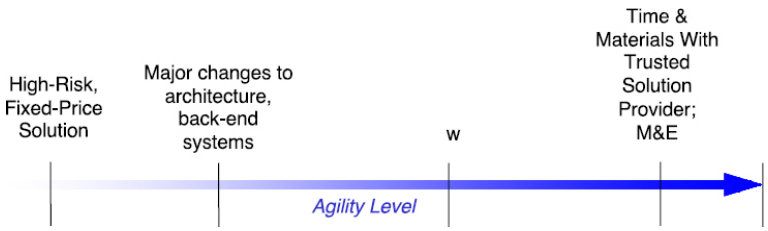

Figure 6.2 *Target agility level by context*

The figure is a general guide. Assess each situation according to its own merits. Let's examine some scenarios.

6.4.3.1 High-Risk, Fixed-Price Solution

The first scenario is a high-risk initiative to purchase a commercial-off-the-shelf (COTS) solution or a fixed-price customized solution. The risk to the business may be high because of a high upfront cost, the absence of a track record with the solution provider, or both. The target agility level for this scenario is low because incremental analysis would expose the customer to unacceptable financial risk: the customer would bear the full cost of any missed requirements not supported in the delivered product. On the other hand, an in-depth upfront analysis protects the business by minimizing the chance it will purchase a product that doesn't satisfy its needs and by transferring the consequences of a low estimate to the solution provider. Use the agile practice of incremental *implementation*, however, wherever practical. For example, a COTS eLearning product may contain components such as a community billboard, registration, academic reporting, and course development that can be turned on incrementally. The advantage of incrementally deploying these components is that exposure to risk is reduced if significant problems arise navigating to the new software. This advantage should be weighed against the possible benefits of the alternative—a Big Bang, all-at-once deployment. The latter approach can prevent costly workarounds often associated with partial deployments. For example, two

systems may have to be maintained during an incremental transition, leading to inefficiencies such as double entries.

6.4.3.2 Changes to Architectural and Backend Systems

When the initiative is a change to a legacy architecture or backend system, the target agility level will be low for technical reasons. It's usually more costly and less efficient to make architectural changes incrementally (the agile way) than it is in a single pass. However, other agile practices are advised for this scenario: use the minimum viable product (MVP) process to verify leap-of-faith design hypotheses. Use continuous integration and (CI/CD) practices to enable speedy, reliable delivery of software changes.

6.4.3.3 Changes to a Known Product with Low Uncertainty

What about a change to a well-understood product or service, where there is low uncertainty about the requirements and the requirements are nonvolatile (unlikely to change)? An example is a change to an insurance product that expands coverage. This scenario requires a medium target agility level. Analyze high- to midlevel requirements up front to ensure the solution covers essential capabilities and to provide enough information for a reliable cost and time estimation. However, you should perform *detailed* requirements analysis incrementally to reduce the lag time before implementation and shorten time to market for the most valuable features. Aim for a high agility level with respect to deployment: use DevOps and CD/CI practices to gain the benefits of early and frequent testing and delivery.

6.4.3.4 Time and Materials with Trusted Solution Provider; Maintenance and Enhancement

The cost–benefit calculation is different for a time-and-materials contract with a trusted developer—for example, ongoing maintenance and enhancement (M&E) by an internal IT department or long-term partner. The target level of agility should be high in this case because there is a foundation of trust, and the financial risk is spread out over time, not frontloaded.

6.4.3.5 Disruptive Business Model

The rightmost scenario in Figure 6.2 represents initiatives where there is extreme uncertainty. The product, service, or technology may be novel and the business model in doubt. This scenario calls for the highest level of target agility. In this circumstance, it's not just software development but the *entire* business ecosystem that is characterized by uncertainty. The affected areas can include marketing departments, distribution channels, and supply chains. As a result, **enterprise agility** is called for—the extension of agile principles and practices beyond the development team to the entire company. Enterprise agility practices for developing a business under conditions of extreme uncertainty include lean startup, discovery-driven planning, and circumstance-based market segmentation.

> For more on enterprise agility, see Chapter 18, "Achieving Enterprise Agility."

6.4.4 Determining the Framework

The next step in process tailoring is to determine the approach and framework that will be used. We'll be focusing on analysis and planning issues. Technical practices such as CD and DevOps are also prepared at this time.

> For guidance on technical practices, see Chapter 15, "Rolling Analysis and Preparation—Day-to-Day Activities," section 15.8.

In Chapter 3, "Fundamentals of Agile Analysis and Planning," we looked at flow-based versus timeboxed agile processes. In flow-based planning (e.g., Kanban), each work item is addressed as it reaches the top of the backlog and progresses independently through the development lifecycle. In a timeboxed approach (e.g., Scrum), all items for a period are accepted into the plan at the start of the period. As a general guideline, select a flow-based approach to plan customer-driven features to optimize adaptability. Use a timeboxed approach to plan strategic initiatives because it's easier to align teams to work on them when everyone is available at the same time. Use a mix of both approaches if teams will be involved in both kinds of initiatives. Keep in mind that these considerations apply to the intake of work items. Once a work item is implemented, it should be delivered continuously if the capability exists.

> For more on flow-based vs. timeboxed planning, see Chapter 3, "Fundamentals of Agile Analysis and Planning," section 3.4.3, and Chapter 17, "Scaling Agility," section 17.3.

Once you've decided on the approach, choose a framework that supports it. For flow-based planning, use Kanban. Frameworks that support a timeboxed approach include Scrum, Extreme Programming (XP), and Scaled Agile Framework (SAFe; for scaled organizations).

Most organizations choose a hybrid process that uses elements from various frameworks. For example, the process may be based on Kanban, with added Scrum ceremonies such as the daily standup and the retrospective. Alternatively, it may be based on Scrum, with work planned on a two-week basis, but also include Kanban tools to manage the workflow of stories once they have entered a sprint.

6.5 TUNING THE PROCESS

Once you've determined the framework—the instrument you will use for analysis and planning—you need to calibrate it to optimize performance. Let's look at how you tune the adjustable parameters of the process, such as cadence, requirements attributes, and the DoR.

6.5.1 Business Analysis Information Artifacts and Events

If the determination has not yet been made, you'll need to select the business analysis and planning artifacts that will be used for the initiative. *BABOK v3* refers to these artifacts as **business analysis (BA) information**—"the broad and diverse sets of information that business analysts analyze, transform, and report . . . information of any kind . . . that is used as an input to, or is an output of, business analysis work."[1] These artifacts include features, stories, functional requirements, business rules, business cases, and organizational models.

[1] International Institute of Business Analysis (IIBA), *BABOK v3: A Guide to the Business Analysis Body of Knowledge*, 3rd ed. (Toronto, Canada: IIBA, 2015), 14–15.

6.5.2 Checklist of Agile BA Information Artifacts

Following is a checklist of the agile BA information artifacts discussed in this book. Use the list to *consider* artifacts for the initiative, but don't expect to produce every artifact on every occasion.

- Product vision statement
- Product canvas
- Vision statement
- Stakeholder analysis
- Goals
- Objectives

- Leap of faith hypotheses
- Metrics
- Main features
- Product roadmap
- Product backlog
- Iteration backlog
- Story map
- Stories: Themes, epics, features, user stories, functional spikes, bug-fix stories, technical stories
- Functional requirements
- Nonfunctional requirements (NFRs)
- Business rules: Behavioral rules (decision tables); definitional rules (data model, data dictionary)
- Business process models
- Use-case models
- Reverse income statement (for discovery-driven planning)
- Assumptions checklist
- Milestone planning chart

6.5.3 Defining Requirements Types

Define requirements types early. It establishes a common lexicon, ensuring that everyone means the same thing when using a term such as *business requirement*. Moreover, once you've created a set of requirements types, you can use them as a checklist to ensure you haven't missed anything in your analysis.

6.5.3.1 Checklist of Requirements Types

Use the following checklist to determine whether you've considered all requirements types. It incorporates *BABOK v3* requirements types with the addition of user requirements.

- Business requirements
- Stakeholder requirements
- User requirements
- *Solution requirements* with two subcategories:
  - Functional requirements
  - Nonfunctional requirements
- Transition requirements

See Chapter 3, sections 3.4.2.4 to 3.4.2.11, for more on requirements types.

6.5.3.2 Why Types Matter (and When They Don't)

When you get into the weeds, defining requirements types can be difficult, and I've often seen lead business analysts struggle with this issue. For example, I've heard (and participated in) arguments regarding whether NFRs should even exist as a type, since you implement them with functional requirements. Should user requirements be a type in its own right or a subtype of stakeholder requirements—or should it be a subtype of solution requirements? I've learned not to indulge these arguments for too long because they usually have little impact on outcomes.

So, why bother with requirements typing at all? It's not as much about *how* you categorize requirements; it's about *doing* it. By defining types and verifying that you've addressed them in your analysis, you reduce the risk of unexpected costs due to missed requirements (e.g., unexpected integration, data conversion, and transition costs). You can also use requirements types as a basis for

- Mapping out which roles will be primarily responsible and accountable for which type.
- Determining how each requirement type will be represented.

- Specifying the point in the development lifecycle at which each requirement type will be tested and how.

6.5.4 Tuning the Backlog

As noted in Chapter 3, the product backlog is the repository of requirements items and other work. You need to set up the backlog at the start of new product development and tune it to optimize it for your circumstance. This setup includes determining what kind of information it will track and how to organize the backlog to support multiple teams. Let's look at some guidelines for carrying out these activities.

> For product backlog guidelines for scaled agile organizations, see Chapter 17, section 17.6.

6.5.4.1 General Guidelines for Setting Up the Product Backlog

Use the following guidelines and principles when setting up the product backlog.

Single Source of Truth

The backlog is a *single* source of truth across all teams. A single product-level backlog

- Supports a whole-product view.
- Simplifies prioritization decisions: the product-level product owner (PO) can view and prioritize across teams while individual feature teams can view items within their domains.
- Simplifies the tracking of dependencies across teams

Specify a PBI attribute to identify the team to which the PBI is assigned. Each team can view its PBIs. Mike Cohn recommends that the number of items viewable to any team be limited to about 100 to 150.[2] This

maximum is based on Dunbar's number of 150,[3] presumed to be a limit to the number of relationships a person can maintain.

[2] Mike Cohn, *Succeeding with Agile* (Boston: Addison-Wesley, 2010), 331–33.

[3] Aleks Krotoski, "Robin Dunbar: We Can Only Ever Have 150 Friends at Most," *The Guardian*, March 13, 2010, https://www.theguardian.com/technology/2010/mar/14/my-bright-idea-robin-dunbar

> For more on establishing product backlogs for scaled agile organizations, see Chapter 17, section 17.6.

Varying Levels of Granularity and Sizes

At any time, items in the backlog will be at varying levels of granularity. In general, the closer a PBI is to the front of the backlog (i.e., the nearer it is to implementation), the higher the level of detail.[4] PBIs also vary in size from large epics to small stories. PBIs near the top of the backlog should be small: the estimate to implement a PBI that is close to implementation should not exceed the size limit for a story.

[4] Ken Schwaber and Jeff Sutherland, *The Scrum Guide: The Definitive Guide to Scrum—The Rules of the Game*, Scrumguides.org, 2017, 13, https://www.scrumguides.org.

The Backlog Is Dynamic

The backlog is a dynamic, not static, list. It is expected that, over time, items will be removed, re-sequenced, changed, and added to the backlog.

The Backlog Is Comprehensive

The backlog includes requirements in the form of epics, features, and stories as well as other work items, such as bug fixes and technical debt payment.

6.5.4.2 Physical Form of Backlog Items

During planning, represent PBIs using physical cards if the teams are physically together; otherwise, use virtual cards. Cards are recommended at this time because they

are intuitive, easy to create, and easy to move around. They also feel temporary, so the team is less likely to see them as set in stone than if they were entered into a requirements tool. If it's a small initiative and team members are colocated, continue with physical cards throughout development. Otherwise, transfer the items to a requirements management tool, once the planning session is over, to provide visibility across teams and locations.

6.5.4.3 Defining PBI Attributes

Determine the attributes (fields) that will be tracked for each item in the backlog. Consider the following checklist of PBI attributes.[5]

[5] Scrum offers the following guidance: "Product Backlog items have the attributes of a description, order, estimate and value." See Schwaber and Sutherland, *The Scrum Guide*, 15.

- Item type: Feature epic, user story, functional spike, and so on
- Description: Text description of the requirement (e.g., use the Role-Feature-Reason format where appropriate)
- Acceptance criteria
- Estimate
- Order: Sequence in the backlog (assigned by the PO)
- Value: The value delivered by the item; may be expressed in "so that" clause
- WSJF prioritization (see section 6.5.4.5)
- Cost of delay: Value of the item, taking into account time criticality (see section 6.5.4.4)
- Owner: Person with primary expertise and sign-off authority for the item
- Planned quarter/release cycle/program increment (with timeboxed planning)

- Designated iteration (with timeboxed planning)
- Projected delivery
- Designated team
- Designated developer(s)
- Status (e.g., ready, done)
- Dependencies and other relationships: Dependencies and relationships with other PBIs and configuration items
- Change log: Record of changes to the item

6.5.4.4 Determining Cost of Delay

The **cost of delay** expresses the value of a backlog item. Determine the cost of delay for an item as follows:

[Cost of Delay] = [User and Business Value] + [Time Criticality] + [Risk Reduction and Opportunity Enablement Value (RE & OE)]

where:

- **Time criticality** is a measure of the decay of value over time.
- **RE & OE** account for the value of the feature in reducing risk or due to an opportunity it enables.

In the pure form of the WSJF method, you assign a relative point value for each term, in much the same way you do for estimating effort with story points.

6.5.4.5 Determining WSJF

To determine an item's priority based on its cost of delay, calculate its **weighted shortest job first** (WSJF) value according to this formula:

WSJF = [Cost of Delay]/[Job Duration]

The higher the WSJF of the item, the higher its priority sequence in the product backlog. (In other words, the PBI with the highest WSJF is sequenced first.)

Popularized by SAFe, WSJF is a useful and recommended aid to prioritization, particularly for features. Use it as a guide to *inform* decisions, though, not to dictate them, because the approach doesn't account well for all scenarios, especially black swan events[6]—relatively rare but impactful occasions when a feature is wildly over- or underestimated. (For example, a black swan may occur when significant technological problems only become apparent once development is underway.)

[6] "SAFe and Weighted Shortest Job First (WSJF)," Black Swan Farming Inc., 2017, https://blackswanfarming.com/safe-and-weighted-shortest-job-first-wsjf

Some have proposed adaptations to WSJF to make it useful not only for prioritization but also for financial planning.[7] Dahlstrom and Lund recommend that the WSJF method be adapted for this purpose as follows:

[7] Gustav Dahlström and Jesper R. Lund, *Is It SAFe to Use WSJF for Prioritisation in Financial Software Development? A Case Study of Prioritisation Needs at a Swedish Bank* (master's thesis, KTH Industrial Engineering and Management Industrial Management, 2019), 5, http://www.diva-portal.org/smash/get/diva2:1372030/FULLTEXT01.pdf

- Establish standard values at the enterprise level for each term in the formula.
- Use a prioritization model that uses absolute, monetary values for cost of delay evaluation at the strategic level but uses relative estimates (e.g., points) at the operational level.

According to their report, these adaptations improve the percentage of satisfied requirements from "50% to 89% on a strategic level and from 85% to 90% on an operational level."[8]

[8]

[8] Dahlström and Lund, *Is It SAFe*, 5.

6.5.5 Determining Requirements Granularity Levels

The reason the level of granularity provided for each item in the backlog varies over time is that you perform agile requirements analysis incrementally. The lean principle applies: wait until the last responsible moment (LRM) to perform analysis tasks. If the development organization uses flow-based planning (e.g., Kanban), wait to prepare large features until about six weeks before their planned implementation, two to four weeks if they're small. Begin to prepare stories one to four weeks before implementation.

If the organization uses timeboxed planning (e.g., SAFe, XP, Scrum), begin preparing features for the next quarter (or SAFe PI) about halfway into the previous one. Begin story preparation about one or two iterations before the planned implementation.

The risk when postponing preparation past these guidelines is reduced team productivity because teams don't understand requirements items well enough to reliably estimate them or correctly implement them.

Because they are prepared incrementally, the items in the backlog will vary significantly in granularity. The following granularity levels have been adapted for stories from the *Use Case 2.0* guidelines for specifying use cases.[9]

[9] Ivar Jacobson, Ian Spence, and Kurt Bittner. *Use-Case 2.0: The Guide to Succeeding with Use Cases* (London: Ivar Jacobson International SA, 2011), 47–48.

- Briefly described
- Bulleted outline
- Steps outlined (essential outline)

- Fully described

Let's examine these granularity levels and when they are advised.

6.5.5.1 Briefly Described

Use this level for low-order items at the end of the backlog. At this level, only a short description of the item is provided. The text may be informal (e.g. "Submit order") or may follow a template (e.g., "As a customer, I want to submit an order so that I can receive my requested items").

6.5.5.2 Bulleted Outline

To achieve this granularity level, outline the item in enough detail to estimate its size and complexity. Specify acceptance criteria indicating the scenarios to test and the expected outcomes. Prepare stories to this level of granularity prior to commitment—approximately one to four weeks before their planned implementation (one or two iterations, if the organization is using a timeboxed framework such as Scrum). If the team is in close collaboration with the business, this target level of documentation granularity may suffice, with the remainder of the understanding achieved through conversation, user testing, and discussions among the Triad—business, quality assurance (QA), and developers.

6.5.5.3 Steps Outlined (Essential Outline)

At this level of granularity, you provide detailed story acceptance criteria and low-level (step-by-step) test-case specifications. If you're using *Use Case 2.0*, you reach this level by specifying the steps in use-case flow specifications. This target level is appropriate when working with third parties that don't have a close, long-term relationship with the product, such as offshore third-party solution providers.

This level is usually not required if there is a long-term relationship with the solution provider, such as a partner or internal IT department, since developers have a good understanding of the business and there is more likely to be a foundation of trust.

6.5.5.4 Fully Described

To reach this level, add detailed notes and functional requirements to the requirements item. Use this level of granularity

- To communicate complex requirements and business rules related to a feature or story.
- Where the relationship between the solution provider and the business is short term.
- When required by regulatory and compliance regulations.

6.5.6 Tracing Requirements and Other Configuration Items

At various points in the life of a story, you need to be able to answer questions such as the following: What objectives does the story support? What tests were run against it? Which stories are affected if we deprioritize it? You use **traceability** to answer these questions. To **trace** an item is to link it with another item. Each traceable item is referred to as a **configuration item**.

You can set up a traceability system to trace a story to configuration items, such as the story's owner, stakeholders, business objectives, tests, graphic user interfaces (GUIs), software components, data tables, and other stories with which it has dependencies.

6.5.6.1 Provide Just Enough Traceability

Traceability's benefits come with a cost—the administrative effort required to keep relationships up to

date. On an agile initiative, this can add up to a significant effort, since requirements are continually in flux. Use automated methods whenever possible. Only trace an item if doing so serves a purpose. If you can't identify at least one important question you can answer by tracing an item, don't trace it.

Once you've identified the questions, determine what you need to trace to what in order to answer them.

6.5.6.2 Provide Traceability in Different Directions

You can trace a configuration item in three directions: downstream (forward), upstream (backward), and cross-stream (horizontally). Let's look at these kinds of traceability and the benefits they provide.

Downstream (Forward) Traceability

To trace an item **downstream** is to link it to configuration items further along in the development cycle. For example, you can trace a user story downstream to the GUIs, test cases, databases, microservices, and software components created, changed, or used as a result of the user story.

Use downstream traceability to answer the following kinds of questions:

- To which team, release, iteration is the story assigned?
- Which software components will be affected by the user story?
- What test cases were run for the user story? What were the test results?
- What actions were taken relative to the story?
- At the beginning of each day: Which user stories are ready for testing."

Upstream (Backward) Traceability

Trace an item **upstream** to link it back to an earlier item. For example, you can trace a user story upstream to business objectives, personas, business stakeholders, business processes, features and epics, and objectives.

Use upstream traceability to answer the following questions:

- Why was this feature added to the backlog?
- What business objectives does it support?
- What feature does this story support?
- What business process does this feature support?

Cross-Stream (Horizontal) Traceability

Cross-stream traceability links items of similar types to each other. For example, you can trace stories to other stories they depend on.

Provide cross-stream traceability stories to answer the following kinds of questions:

- If we delay implementation of a user story, what other user stories will it hold up?
- If we postpone story X, what other stories could be delayed as a result?
- If we deliver story Y, what stories do we need to regression-test as a result?

You can also trace a story to the business rules it supports, its functional requirements, and its NFRs. In addition, you can trace the history of changes that were made to a configuration item over time, by whom, and why.

6.5.6.3 Determining Traceability Mechanisms, Tools

We've talked about the *relationships* that should be traced. If a mechanism has not yet been established, the next step is to determine *how* traceability will be implemented. Before we explore the options, let's summarize the main guidelines for a typical, large distributed agile development organization.

If the organization is practicing DevOps/CD, provide traceability wherever possible through the configuration management system (CMS) and version control in order to obtain current and reliable information with minimal overhead. If CMS tracing is unavailable or inadequate (e.g., it doesn't provide the necessary granularity), use a requirements management tool.

> During colocated planning sessions, manual traceability approaches are best. For example, SAFe uses red strings to trace dependencies between stories. Once teams disperse, however, enter stories and their relationships into the CMS or requirements tool to provide access across all locations, as described previously.

Now let's explore the mechanisms further. Use the leanest traceability mechanism for the situation—one that delivers the information, accuracy, and access needed with the minimum overhead cost.

Traceability through the CMS

If the development organization follows best practices in CI/CD, traceability between configuration items is provided by the CMS and version control. A history of changes to configuration items is also provided.

Each traceable configuration item is represented as a record (row) in the configuration management database (CMDB). The CMS maintains relationships between configuration items by managing the connections between corresponding records in the database.

Figure 6.3 is an example of relationships maintained between configuration items on a claims project. The configuration items include user stories, requestors, objectives, and software components such as classes, services, and subsystems.

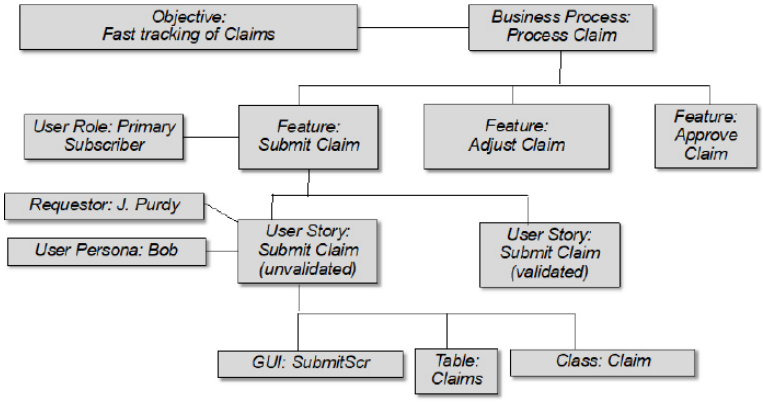

Figure 6.3 *Tracing configuration item relationships in a configuration management database for a claims project*

Traceability from features to automated tests (not shown) is achieved by organizing behavior-driven development test specifications into feature files.

> For more on behavior-driven development, see Chapter 10, "Quarterly and Feature Preparation," section 10.9.5.

Traceability through Requirements Management Tools

As noted earlier, when a CMS is not available or does not provide sufficient traceability, use a requirements management tool. Aside from traceability, requirements management tools also offer the capability to generate burndown charts and support Kanban workflow management processes and artifacts. One of the most popular tools for agile teams is Jira Software by Atlassian.

> For guidance on selecting requirements management tools, see Chapter 17, section 17.9.

Traceability Spreadsheets

If the preceding methods are unavailable or their cost is prohibitive, a simple, low-cost solution is to trace requirements elements using a spreadsheet application such as Excel or Google Sheets. Table 6.1 is an example of a traceability matrix for BLInK, the insurance product in our case study that provides benefits to customers based on their behaviors.

Table 6.1 Product Backlog Item (User Story) Traceability Matrix

PBI/Story	Sequence	Description	User	Estimate	Team	Business Rules	User Interface	Tables
PBI100	1	Receive BLInK marketing on invoices	Primary subscriber	2	Orange		RF003	
PBI030	2	Earn premium reduction for sign-up	Primary subscriber	3	Orange	BR012	UI004	Subscription table Application table
PBI022	3	Offer BLInK with Quick Quote to select, preapproved customers	Broker/agent	5	Blue	BR012	UI030	

Table 6.1 shows that story PBI1030 supports business rule BR012 and that implementing the story affects the subscription data table and an application table.

Now let's turn to some lightweight, manual traceability options.

Visual Cues

With this approach, you represent stories as cards (or their virtual equivalents), and use *visual cues* to signal their relationships. The most common use of a visual cue is the placement of a story on a story map: stories are generally dependent on stories placed above them and the stories to their left.

Other visual cues include the use of color-coded dots and red strings to indicate dependencies on a program board.

> For more on dependencies on story maps, see Chapter 12, "MVPs and Story Maps," section 12.5.4.

> For more on the red string and colored dot approaches, see Chapter 11, "Quarterly and Feature Planning," section 11.11.3.5.

Conversation

The most lightweight manual option for managing dependencies is through conversation. Those working on related items communicate directly to each other about their plans and progress. Use discussion to communicate inter-team dependencies during *big room iteration planning meetings*. As dependency issues arise, people move between teams to notify those affected and resolve problems. Another conversation option is for teams to send envoys to sit in on the planning sessions of those they work with closely or for collaborating teams to conduct their planning sessions together.

> For more on lightweight solutions for coordinating dependent teams, see Chapter 17, section 17.10.

6.5.7 Setting Process Parameters

An agile analysis process has configurable parameters—such as cadence, the DoD, and WIP limits. Tune these parameters for optimal performance at the start of an initiative, and readjust them over time.

6.5.7.1 Determining the Cadence

If the teams are using a timeboxed planning approach (e.g., Scrum), they need to decide on the duration—or cadence—of the timeboxes. This step does not apply if teams are using a flow-based planning framework (e.g., Kanban), because work is not planned by time period but by work item.

Two timeboxes to consider are the iteration and quarterly. The iteration cadence is typically one or two weeks. The maximum duration of an iteration varies by framework. In XP, it's one week. Scrum sets an upper

limit of one month. Most timeboxed frameworks require a fixed cadence. (XP and SAFe demand it. Scrum recommends it.[10])

[10] "Sprints have consistent durations throughout a development effort." Schwaber and Sutherland, *The Scrum Guide*, 9.

The iteration cadence determines the planning horizon—how far ahead you're planning. However, it should *not* determine when stories are delivered. Stories should be delivered continuously, as they are completed, not at the end of the iteration. Note that to *deliver* a story is to provide it in a releasable state; however, the customer may decide to delay its release into production until a later date.

If your development organization has not adopted DevOps practices, it may not be able to integrate and test stories reliably at that pace. In that case, the cadence acts as an upper limit on delivery. Development commits to deliver a releasable integrated increment *at least* once per iteration.

If there are longer-term planning needs, you will need to set a cadence for the longer horizon. You can set it to the duration of a release cycle (one to six months), the length of a SAFe program increment, or a calendar-based period (e.g., a quarter). For simplicity, I refer to the long-term planning horizon as a **quarter**. When you see the term in this book, it means any of these variations.

6.5.7.2 Set Estimation Standards (When Relative Estimation Units Are Used)

You can use this method regardless of the estimation approach. It applies to story points, ideal developer days (IDDs), Gummi Bears, and Nebulous Units of Time (NUTs).[11] Let's assume the unit is the story point—the most popular choice. Gain consensus from the team on a story, or a set of stories, as the standard for a 1-point

requirement item. If you're using story points or IDDs, the story should require one day's effort by one dedicated developer with the appropriate skills for the work item. Teams may also find it useful to establish an 8-point standard so that they can better gauge where a given story lies within the acceptable point range of 1 to 8.

[11] "Points (estimates in)," *Agile Alliance Glossary*, 2020, https://www.agilealliance.org/glossary/points-estimates-in

Once the team has established a standard, estimate the other stories in relation to it. For example, the team assigns 3 story points to story X, indicating that they estimate the effort to implement it to be three times that of the standard one-unit story.

It's worth noting at this point the secret about these estimation units. Unless you're using real-time estimates, it doesn't matter, in the end, what standard you used to represent one unit of effort. Within a few weeks or iterations, the team establishes a record of the number of units it *actually* can deliver in a given timeframe. That's what you use—not the original time associated with a one-unit story—to make forecasts once some time has passed.

6.5.7.3 Determining Initial Capacity (Velocity)

The team's **capacity**, or **velocity**, is the number of estimating units (e.g., story points or IDDs) that it can deliver within a given period. Calculate velocity on a per-team basis, not for the entire group of teams.

You can determine velocity for various estimation units and planning horizons. The most common usage is to measure the total number of story points a team delivers per week or iteration.

Use velocity for forecasting purposes; don't use it as a performance metric or to compare teams. Following is a simple-to-use formula for converting estimates to time:

(Remaining Time) = (Remaining Estimating Units)/Velocity × Cadence

For example, suppose there are 400 remaining story points in the product release backlog, the velocity is 40 story points per iteration, and the cadence is two weeks/iteration. The remaining time is 400/40 × 2 = 20 weeks.

For a more accurate forecast, make projections based on the team's burndown chart.

For guidance on forecasting using a burndown chart, see Chapter 15, section 15.7.5.5.

If the team is using a timeboxed planning approach (e.g., Scrum), you need to know the team's past velocity going into iteration planning, since you use it to forecast future capacity—the number of estimating units that the team will deliver in the upcoming iteration. To quote Shakespeare, "The past is prologue."[12] The best forecast for what the team *will* achieve is based on what it achieved in the past. If the team has worked on a comparable initiative, set the initial velocity for the new venture based on what it accomplished previously, adjusting for changes in team sizes, holidays, availability, and so on.

[12] William Shakespeare, *The Tempest*, act 2, scene 1, line 217, PlayShakespeare.com, https://www.playshakespeare.com/the-tempest/scenes/act-ii-scene-1

If the team is new or hasn't worked on a comparable initiative before, use one of the methods that follow to forecast capacity.

Determining Initial Capacity, Method 1: Based on Availability

Base your projection on experience, if possible, but if there is no comparable example to draw from, you may use the following formula:

$$\text{(Initial Capacity)} = \text{(Total Potential Person-Days)} \times \text{(Availability)} - \text{(Slack)}$$

The **total potential person-days** represents the entire budget if all members are fully dedicated to the work. For example, for an eight-person team and a two-week cadence (ten working days), the potential person-days are 8 × 10 days = 80 days. **Availability** is the fraction of time members can spend on the planned work while accounting for time lost to other activities and events. It includes time lost to work on other projects, meetings, demos, illness, vacations, unplanned bug fixes, administrative activities, and email. A rough guideline is to expect availability to be within one-third to one-half of potential time.[13] Following lean guidance,[14] you should also set aside some **slack** time so that team members will be available to pitch in and unplug bottlenecks when they occur.

[13] Mike Cohn, *User Stories Applied: For Agile Software Development* (Boston: Addison-Wesley Professional, 2004), 104.

[14] Mary Poppendieck and Tom Poppendieck, *Lean Software Development: An Agile Toolkit* (Boston: Addison-Wesley, 2003), 81.

Determining Initial Capacity, Method 2: Guesstimate and Decompose

An alternative method for forecasting the initial velocity (see Cohn)[15] is to guesstimate how many stories can be delivered and verify the guesstimate by decomposing the stories into tasks. To use this method, proceed as follows:

[15] Mike Cohn, "How to Estimate Velocity as an Agile Consultant," Mountain Goat Software, July 25, 2013, https://www.mountaingoatsoftware.com/blog/how-to-estimate-velocity-as-an-agile-consultant

- Ask the team to review the first stories in the product backlog—items that have been sequenced and estimated.
- Ask the team to guess how many of these stories they will be able to deliver in the first iteration.

- Ask them to decompose each story into individual tasks and estimate the time to perform each task.
- Add up the task times for all the selected stories.
- If the sum of the task times exceeds the available time in the iteration, drop stories from the back of the list until the total task time is within the available time.
- Set the initial capacity to the sum of the points given to the stories that remain.

This alternative process may seem like a lot of extra work, but it's not. Many teams would be decomposing stories into estimated tasks, in any case, during iteration planning.

Adjusting Capacity Once Implementation Begins

Once the first period or iteration has passed, forecast future capacity according to the following formula:

[Capacity] = [Past Velocity] ± [Changed Circumstances]

As the formula indicates, the future capacity is based on past performance (past velocity), making adjustments for changed circumstances such as holidays, illness, changes to team composition, and technical challenges[16].

[16] While slack is incorporated indirectly in the above formulation as a component of past velocity and changed circumstances, there is an argument for explicitly subtracting slack in forecasting capacity based on past velocity to ensure the team isn't always operating at peak capacity and to account for unknown unknowns. (Karl Wiegers in personal communication with the author.)

For forecasting purposes, use *recent* velocities achieved by the team during the past three or four iterations. Don't use a historical average because the team's performance can be expected to vary, fluctuating at first but settling down after three to six iterations.

Estimation and Velocity for Multiple Teams

When working with multiple teams, have all teams meet at the start of the initiative to agree on a standard story or stories to represent a 1-point estimate. You can also establish standard stories for an 8-point story. Afterwards, the team that is likely to implement an upcoming story estimates it relative to this standard.

Velocity has relevance only at the team level. Where there are multiple teams, each team tracks its own velocity and uses that to determine team capacity—the number of estimation units it can accept into the iteration.

Estimating Features versus Stories

The preceding considerations about estimates are primarily a concern at the story level. When estimating features, it's often sufficient for a team to specify the number of iterations or weeks it will take to deliver the item.

> For more guidance on estimating features, see Chapter 11, section 11.11.2.

6.5.7.4 Tuning the Definition of Done

The **DoD** specifies conditions that must be true before a story can be considered "done." It applies to all stories. In addition to the DoD, each story must satisfy its acceptance criteria to be deemed done. By specifying a DoD and enforcing compliance with it across all stories, the organization achieves a consistent level of quality for all delivered requirements.

Work with the development organization to specify a DoD if one does not yet exist. Review it and update it over time to achieve higher levels of quality,[17] For example, at the start of a transition to automated testing, the following condition is added to the DoD: "Automated unit and low-level integration tests have been created and passed." Later in the transition, another condition is

added: “Automated UAT tests have been created and passed.”

[17] Schwaber and Sutherland, *The Scrum Guide*, 18.

Review the effectiveness of the DoD and consider improvements during the iteration retrospective[18] and quarterly retrospective.

[18] Schwaber and Sutherland, 14.

Definition of Done Examples

There is no standard DoD that works for all organizations; it will depend on circumstances, such as the degree to which CI/CD and DevOps are practiced. At a minimum, the DoD should stipulate that the story has been demonstrated and tested by the customer.

The following is an example of a DoD.

- Coding is complete.
- The code has been checked in.
- Automated tests created and passed: unit tests, integration tests, UAT.
- The story has passed tests satisfactorily: unit-tests, low-level integration tests, functional tests, acceptance criteria, end-to-end integration testing, system testing (regression testing, performance testing, etc.).
- Compliance, governance, and NFRs have been satisfied.
- The code has been refactored.

6.5.7.5 Tuning the Story Definition of Ready

Deem a requirements item **ready** when it is understood well enough for the team to make an informed commitment and begin implementation without undue delay or need for subsequent rework. It’s a good idea to formalize what “well enough” means by specifying

conditions in a DoR. In Kanban, you can indicate that a work item has satisfied these conditions by moving it to a Ready column.

If your team uses timeboxed planning, require that stories must pass the DoR to be considered for iteration planning.[19]

[19] Schwaber and Sutherland, 15.

To enable high team productivity, stories should be more than "simply items on a list."[20] They should satisfy additional conditions, such as being estimable. To propagate best practices and to ensure a standard level of preparation, specify these conditions in a story DoR.

[20] Jeff Sutherland, "Scrum: What Does It Mean to Be Ready-Ready?" (OpenViewVenture, 2011), https://www.youtube.com/watch?time_continue=3&v=XkhJDbaWojo

Definition of Ready Example

The following DoR example incorporates INVEST guidelines and others.[21],[22]

[21] Sutherland, "Scrum."

[22] For example, Roman Pichler, "The Definition of Ready in Scrum" [blog post], December 16, 2010, http://www.romanpichler.com/blog/the-definition-of-ready

> INVEST is a mnemonic for writing quality stories. The acronym stands for independent, negotiable, valuable, estimable, sized-appropriately, testable. For more on INVEST guidelines for stories, see Chapter 13, section 13.12.1.

- The story is independent.
- The story is refined.
- The story is well articulated.
- The story is actionable.
- The story is negotiable.
- The story is contextualized.

- The story is unique.
- The story is consistent with other stories.
- The story's value is known.
- The story has been discussed and prioritized recently.
- The story is estimable.
- The story is testable.
- The story is small.
- The story is vertically sliced.
- The story is non-solutionized.
- The story is complemented with supporting documentation, including a wireframe.
- The story and acceptance criteria have been approved.
- The customer and team have confirmed that the story satisfies the previous conditions.

Let's look more closely at these DoR conditions to understand them better.

The Story Is Independent. The story has minimal dependencies on other stories. Independent stories reduce waste because they remove the administrative cost of tracking dependencies. In addition, they prevent delays that occur when a story isn't done in time for another that depends on it.

The Story Is Refined. The story has been sufficiently analyzed and prepared to be completed ("done") within an iteration or a specified size limit.

The Story Is Well Articulated. The story is described in a way that clarifies who wants it and what they want. You can satisfy this condition through the use of a standardized template, such as the Connextra template, "As a [user role], I want [function] so that [business

value].” A template helps remind story writers to articulate critical issues, but its use should not be enforced too strictly: it’s not worth the effort trying to fit every story into a predetermined format if you can more clearly and naturally express it in another way.

> For more on story templates, see Chapter 13, section 13.9.2.

The Story Is Actionable. The story must be immediately **actionable** by the team:

- It should have no outstanding dependencies on other stories that would delay implementation.
- Preparatory analysis and technical work for the story are complete enough for the story to be completed during the implementation iteration or within the time estimate.
- The **architectural runway**, including specification of service communication protocols, creation of infrastructure, and identification of components, has been sufficiently prepared.

The Story Is Negotiable. The story is ready to be negotiated. The PO and team agree that the requirements for the story are not fixed but will evolve on the basis of on dialogue. Business stakeholders and the development team understand the value and costs of the story well enough to negotiate trade-offs.

The Story Is Contextualized. The context for the requirement is clear. As an example of a story with unclear context, consider “As a customer, I want the system to present the service fee to me up front.” It’s unclear what the user is doing while this requirement is active. To contextualize the story, be specific: “As a Web banking customer, I want the system to present the transaction service fee up front whenever I perform a banking transaction.”

The Story Is Unique. The requirements in the story should not duplicate those in another story. Duplication leads to overestimation of work and duplicated effort.

The Story Is Consistent with Other Stories. The requirements in the story do not contradict those in another story. Inconsistencies in the requirements result in discrepancies in the product if they're not caught and rework if they are.

The Story's Value Is Known. Mary and Tom Poppendieck quote statistics[23] that show that 45 percent of developed features are never used, and 19 percent are rarely used—adding up to a whopping 64 percent of features with no or minimal value. Sutherland[24] has reported that, if one were to examine a typical backlog, at least one-third of its items would be "junk stories"—stories that have no value that anyone can identify. To improve these statistics, the value of a story must be clear before the team commits to it.

[23] Mary Poppendieck and Tom Poppendieck, *Lean Software Development: An Agile Toolkit* (Boston: Addison-Wesley, 2003), 32. The statistic is attributed to the Standish Group.

[24] Sutherland, *Scrum*.

You can communicate the story's value qualitatively through the "so that" clause of the Connextra template. You can also specify it quantitatively by calculating the story's cost of delay—its overall value—as described in section 6.5.4.4.

The Story Has Been Discussed Recently. It's not enough to determine a story's value when it was written. Its value may have changed over time. To be ready, the story must have been discussed and its value confirmed *recently* by business stakeholders.

The Story Is Estimable. The story is understood well enough to be estimated by the team if called upon to do

so. It isn't necessary that the story actually has been estimated—just that it *could* be.

The Story Is Testable. The story's acceptance criteria are understood well enough so that tests can be specified, a sufficiently reliable estimate can be made, and implementation can begin.

It is expected that acceptance criteria will evolve. The higher the uncertainty about the product or feature, the more general the acceptance criteria should be to allow more room for the customer to determine what is acceptable through trial and error.

The Story Is Small. The story fits within the maximum limit set for a story. A common restriction is for most stories not to exceed 5 story points, with a strict maximum of 8 points. Where timeboxed planning is practiced (e.g., Scrum), there is the added stipulation that the story must be small enough to be comfortably implemented during an iteration.

One useful tip for keeping stories small is to include only one distinct capability per story. Not only does this practice help limit size, but it also makes it easier to cost and prioritize each capability individually.

The Story Is Vertically Sliced. The story is structured so that, to the degree possible, it cuts a vertical slice through architectural layers to deliver usable functionality to the user with minimal or no dependencies on other stories.

The Story Is Non-solutionized. The story has been written in a non-solutionized manner to leave room to determine what solution works best through trial and error. The story should describe need and value rather than a design solution. For example, the story "As a customer, I want to be able to search for products by

type" is better than "As a customer, I want the system to present a dropdown box of product types."

The Story Is Complemented with Supporting Documentation, including a Wireframe. Other artifacts required for the team to begin implementing the story, including wireframes, are available. A **wireframe** is a rough layout of the screen, showing items and how they will be grouped, without indicating design choices.

The Story and Acceptance Criteria Have Been Approved. The story, including its functionality and acceptance criteria, has been approved according to the agreed-upon approval mechanism. That mechanism may be anything from an informal discussion to a formal sign-off by a designated authorizer, such as the PO or superuser.

The Customer Has Confirmed That the Story Meets the Preceding Criteria. When I ask teams about their DoRs, they often show me very well-articulated ones. The trouble is that they don't use them! Require that every story be evaluated against the DoR before it is accepted into an iteration.

How to Avoid Gating in the Definition of Ready

A poorly expressed DoR runs the risk of becoming a gating mechanism[25] nudging the team back toward a sequential waterfall approach. To avoid this tendency, define conditions that do not require completion but instead allow for concurrency. Rather than stating a flat rule that a particular activity or artifact must *always* be complete for the story to be ready, state the *circumstances* in which the rule applies, and describe the desired *degree of completion*. For example, consider the following DoR condition: "If the story is part of a larger process, the process has been analyzed well enough to understand the impact on other stories and other steps of the process." This condition specifies the circumstance

("If the story is part of a larger process") and the degree of completion required ("enough to understand").

[25] Mike Cohn, "The Dangers of a Definition of Ready" [blog post], Mountain Goat Software, August 9, 2016, https://www.mountaingoatsoftware.com/blog/the-dangers-of-a-definition-of-ready

6.5.7.6 Tuning the Feature Definition of Ready

Use a **feature DoR** to specify the conditions a feature must satisfy before it is included and committed in the plan. When the team is using a flow-based approach such as Kanban, this commitment occurs feature by feature, as each one nears the top of the backlog. When it is using timeboxed planning, features are committed to as a group at the start of the quarter, release cycle, or PI.

The following feature DoR incorporates guidelines from Ivar Jacobsen[26] and others.

[26] Ian Spence, "Preparing Features for PI Planning—What Does It Mean for a Feature to Be Ready?" Ivar Jacobson International, https://www.ivarjacobson.com/publications/blog/preparing-features-pi-planning-what-does-it-mean-feature-be-ready

- The feature has no (or minimal) dependencies on other features.
- All dependent teams are committed: All teams upon which delivery of the feature is dependent have committed to the feature. No feature is accepted into a quarter (or release cycle) unless this condition is met.
- The feature is negotiable: All parties understand that everything is negotiable concerning the feature—its priority, functionality, acceptance criteria, and estimate. The costs and benefits of the feature are understood well enough for trade-offs to be negotiated during PI/release planning.
- The feature is valuable: The feature's value is understood and has been confirmed recently through discussion with the customer.

- The feature is estimable: The feature is understood well enough to be estimated. The closer the feature is to implementation, the more accurate the estimation.
- The feature is right-sized: The feature is small enough to be implemented within a quarter yet significant enough to be of value if released to the market.
- The feature is testable: Acceptance criteria for the feature are known. The feature's acceptance criteria communicate how far the feature's implementation must go for it to be releasable. Business stakeholders, QA, and developers share an understanding of how the feature as a whole (vs. its individual stories) will be tested and the test procedures that will be used.
- The feature is clear: The customer can explain the feature to the team.
- The feature is feasible: It is possible to implement the feature within the time frame provided by the estimate. Financial and technical risks are understood and within acceptable limits.
- The feature is owned: Someone has been designated as the primary person for discussing and approving the feature.
- Stakeholders are engaged: Stakeholders impacted by the feature have been identified and commit to being involved in its development.
- The cost of delay for the feature is understood. (See section 6.5.4.4 for cost of delay calculation.)
- The feature is prioritized: Priority sequencing may be determined using the WSJF method. (See 6.5.4.5 for WSJF calculation.)

- If the feature impacts a business process, the process is well-enough understood to identify dependencies on other stories and effects on other steps in the process.

6.5.7.7 Tuning Workflow Parameters (Kanban)

The Kanban approach contains several configurable parameters. These include target *item sizes*, the *states* that will be tracked (columns of the Kanban Board), and *WIP limits*.

Determining Item Size

Kanban's guideline regarding work items is that they be sized so that they are *understandable* and *doable*. Typical sizes in Kanban range from about half an hour to two days, with a preference for items sized at one day or less.

In a context where the work items represent stories, the item sizes are often larger. As we've seen, a story can take up to five days to implement (and even eight days, on occasion) because it has the added constraint of delivering useful value—and that's not always possible within a limit of one or two days.

Determining Kanban States (Columns)

Kanban does not specify the states that must be tracked for a work item. Typical statuses for a story are *Backlog* (i.e., the item is present in the backlog but not yet ready), *Ready* (satisfies the DoR), *In Progress*, and *Done*. Other statuses often added before *Done* include *Code Review* and *Test*. Table 6.2 is an example of a Kanban board to track the states of stories as they make their way through their lifecycle.

Table 6.2 Example of Kanban Board

Backlog	Ready	In Progress	Code Review	Testing	Done
PBI101	PBI045	PBI077	PBI057	PBI088	
PBI087	PBI099	PBI127	PBI152	PBI126	
PBI141		PBI131		PBI044	

> For examples of Kanban boards used in conjunction with timeboxed iteration planning, see Figures 14.5 and 14.6 in Chapter 14, "Iteration and Story Planning."

Determining WIP Limits

Once you've determined the Kanban columns, you need to set a WIP limit for each one. The following are guidelines for setting WIP limits in software development.

Keep the WIP limit for the *In Progress* column lower than the number of team members to reduce inefficiencies. This practice limits the need for multitasking and allows for slack time so that someone is more likely to be available to help unplug bottlenecks.

When using timeboxed planning, make sure the WIP limit for the Ready column (containing stories queued up for iteration planning) does not exceed the number of team members. Anything higher would result in more ready stories than the team can work on, assuming each member takes one story. That, in turn, would mean you had prepared more stories to be ready for the iteration than you needed to—violating the last-responsible-moment principle.

Start with What You Do Now: Agree to Pursue Incremental, Evolutionary Change

In setting WIP limits, follow the Kanban principles: "Start with what you do now" and "Agree to pursue incremental, evolutionary change."[27] Begin by letting the development process play out, tracking the movement of work items across the Kanban board without seeking to intervene. Once you've allowed the process to run for a while, set WIP limits by observing the number of items that have tended to accumulate in each state. Over time, make changes to WIP limits and work item sizes with the aim of reducing the time items spend in each state.

[27] "Kanban," Agile Alliance Glossary, 2020, https://www.agilealliance.org/glossary/kanban/#q=~(infinite~false~filters~(postType~(~'page~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'kanban))~searchTerm~'~sort~false~sortDirection~'asc~page~1)

6.6 OPTIMIZING THE PROCESS USING VALUE STREAM MAPPING

Once the development process is up and running, use value stream mapping to locate opportunities to remove waste.

To use this lean technique, first determine the steps that a work item (e.g., user story) goes through as it passes through the development process, then research how long it spends at each stage. Figure 6.4 is an example of a value stream map resulting from this analysis.

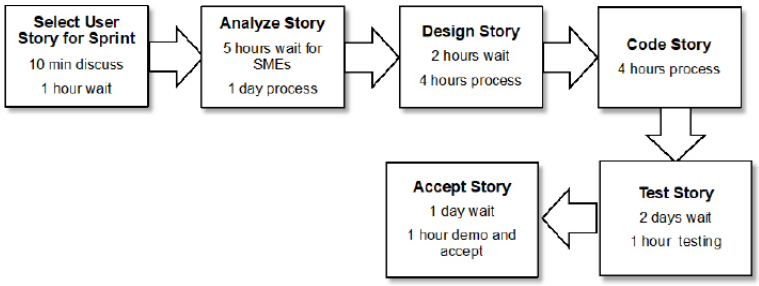

Figure 6.4: *Value stream map for implementing a user story*

Figure 6.4 indicates two significant sources of waste in the development process: five hours in Analyze Story (while the analyst waits for SMEs to become available) and two days in Test Story (while a story waits to be tested). The first can be improved by managing stakeholder expectations up front regarding participation and working with managers to obtain the required time commitment. The second waste item may be an indication that more testers are needed on the team.

6.7 DETERMINING PROCESS READINESS

Use the following checklist to determine whether the agile planning and analysis process is ready. Create a plan to address any remaining gaps.

- Has a base process been developed or selected (e.g., Kanban, Scrum) and tailored to the context?
- Has a product backlog been set up, and is it ready to be used?
- Have standard requirements types and attributes been established?
- Has a traceability process been established?
- Have BA artifacts and events for the process been established?
- Have Kanban workflow parameters and artifacts been set or initialized (e.g., WIP limits)?
- For timeboxed planning, has the cadence been determined?
- Has a shared DoD been established?
- Has a shared DoR been established?

6.8 CHAPTER SUMMARY

Here are the key points covered in this chapter:

- Trace stories backward to features, business objectives, business processes, and so on; horizontally to other stories; and forward to tests, software components, and so on.
- Specify a DoD to indicate conditions that must be satisfied before a story is accepted as complete.
- Specify a DoR to indicate conditions that must be satisfied before committing to a story during planning.
- Forecast capacity based on past velocity, adjusting for changes in availability.

6.9 WHAT'S NEXT?

This chapter focused on *process preparation*—preparing the agile process that will be used to analyze and manage requirements. In Chapter 7, "Visioning," we begin that process by working with stakeholders and developers to envision the product and its impact on customers and the business.

Chapter 7. Visioning

This chapter examines visioning activities. In visioning, stakeholders articulate a vision of the future state and the reason for embarking on the endeavor. Visioning is carried out when developing a new product or undertaking a change initiative. It is revisited quarterly and as required by changing circumstances, such as a market opportunity—or loss of an opportunity—due to unexpected regulatory changes.

The chapter begins with the first step in the process, root-cause analysis. Root-cause analysis is a set of methods for tracing the causes of an effect back to a root cause. An example of an effect is poor customer retention. An example of a root cause is an *inefficient customer-service process*. Root-cause analysis techniques covered in this chapter include Five Whys and cause–effect diagrams. The chapter explains how to use this analysis to inform the crafting of product vision statements, epic vision statements and problem statements. The identification of root causes helps direct development investment to areas that have the greatest potential to impact outcomes.

Stakeholder analysis begins as early as possible in the initiative and is continually updated and developed as more stakeholders are discovered and their needs become better understood. The chapter includes guidelines, templates, and checklists for performing this analysis and using it to inform the communication plan.

The next step covered in the chapter is the identification of leap of faith hypotheses—critical assumptions that underlie the vision and must be valid for the undertaking to be viable (e.g., the hypothesis that users will pay for a subscription to a new service). Leap of faith hypotheses

are identified and tested early so the company can decide whether to invest its resources in the endeavor or pivot to another hypothesis. The chapter explains the agile approach for doing so using the minimum viable product (MVP) process. An MVP is a low-cost version or facsimile of the proposed product or feature used for learning. Customers interact with MVPs, and their feedback is used to test hypotheses, features, changes, and design solutions.

The chapter ends with guidelines for specifying metrics to validate hypotheses and measure progress toward goals and objectives. It explains how to define **actionable metrics**—measurements that can be used to make data-informed decisions.

The chapter also marks the beginning of the BLInK case study workshops that run throughout the book. Each workshop provides a snapshot of agile analysis and planning tools as you step through the development lifecycle.

7.1 OBJECTIVES

This chapter will help you to

- Identify root causes and needs.
- Articulate the vision statement for the product or epic.
- Craft a problem statement.
- Analyze stakeholders, goals and objectives.
- Discover leap of faith hypotheses.
- Specify the actionable metrics that will be used to validate hypotheses.

7.2 THIS CHAPTER ON THE MAP

Figure 7.1 highlights the activities included in this chapter.

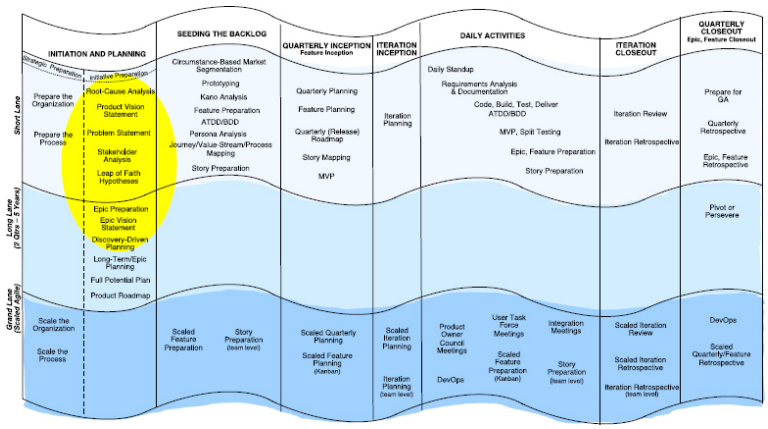

Figure 7.1 Chapter 7 on the map

Figure 7.1 highlights the visioning tools we cover in the first zone, *Initiation and Planning*. These are root-cause analysis, product vision statement, epic vision statement, stakeholder analysis, problem statement, leap of faith hypotheses, and circumstance-based market segmentation.

7.3 OVERVIEW OF PRODUCT VISIONING AND EPIC PREPARATION

In visioning, stakeholders articulate a shared vision of the future and the reason for undertaking the endeavor. This activity is performed at the product level (product visioning) to communicate the rationale for building the product. It also occurs at the epic level to articulate the rationale for a major change initiative and prepare the epic for planning and implementation.

Visioning occurs primarily at the beginning of an undertaking, but the activities are revisited quarterly and as required by changing circumstances (e.g., because a hypothesis for the product no longer holds true or because of a new opportunity or threat).

Individuals typically responsible for product visioning include the following:

- Director of product

- VP of product
- Chief product officer
- Product-level product owner (PO), Portfolio
- Senior business analyst

Junior business analysts are not typically involved in the *formation* of the product vision, but they share the responsibility to *communicate* the product vision and objectives to the team.

Visioning is an essential activity for achieving any bold target that requires sustained effort, such as Nike's moonshot challenge to "double business with half the [environmental] impact."[1]

[1] "NIKE, Inc. Sets Bold Vision and Targets for 2020" [press release], *Business Wire*, May 11, 2016, https://www.businesswire.com/news/home/20160511005885/en/NIKE-Sets-Bold-Vision-Targets-2020

The envisioned target may be for

- an enterprise (enterprise visioning),
- a product (product visioning), or
- an epic or change initiative **(epic visioning)**.

In this chapter, we focus on product and epic visioning.

The result of visioning is a vision statement. Internally, a product vision statement motivates the team and supports the development of a cohesive product. Externally, the product vision statement communicates the product's potential value to investors and early adopters (earlyvangelists) at the start of a venture and drives marketing messages to customers. Similarly, epic vision statements and problem statements communicate the reason for a change initiative.

7.3.1 An Example of Product Visioning and Why It's Important

A couple of years ago, I worked with a team of consultants tasked with choosing an incident management solution for a public transportation agency. The incidents ranged from small schedule delays to fatal accidents. (Suicides were a particularly common and unsettling incident type; one unfolded in real-time as I was viewing the monitoring system with the team.)

After I'd introduced them to the benefits of visioning, they realized that they had never gone through this vital step even though they were well into the preliminary analysis. And so, they convinced their manager to convene a visioning workshop with stakeholders. During the workshop, the participants defined a vision of the product that was very different from the one assumed by the consultants. The participants expressed a vision of a solution that would free up customer support agents from the mundane incidents that occupied most of their time so that they could focus on those tasks best handled by humans. The consultants, on the other hand, had been working under the assumption that the vision was of an all-encompassing tool that could manage *any incident* an agent might have to deal with. This insight resulted in significant savings because it meant that the team could now exclude incident types that required integration with police and government systems. As it turned out, those were the types that had contributed most to the initial cost estimates.

7.3.2 Visioning Readiness Checklist

By the time you get involved with the initiative as a business analyst, the product vision may already have been defined or completely missed, as was the case in the preceding example. If you were not part of the initial analysis, your first step should be to check if any steps in the product visioning process were skipped. Use the checklist in Appendix A4 to determine if that's the case.

As a business analyst, you are a change agent. If you discover that important steps were missed during product visioning, it is your responsibility to raise your concerns, highlighting the risks of not carrying them out. Though you may lack the authority to act on those concerns, you *can* influence those who do.

7.3.3 Initial Stakeholder Identification

As soon as possible, identify the primary individuals and groups affected by the initiative and those who impact it, such as approvers. Key stakeholders include business subject matter experts (SMEs) in the areas affected by the initiative, the PO, sponsor, and steering committee members.

Stakeholder identification and analysis is ongoing. As the initiative progresses, expect to discover more stakeholders and learn more about their needs. We examine these activities more fully in section 7.9.

7.3.4 Facilitation Tips

Product visioning and root-cause analysis workshops typically take place through group facilitation led by a PO or business analyst. As a facilitator, the soft skills you bring to the table to these and to subsequent workshops are at least as important as the analysis techniques covered in this book. Tips for facilitating stakeholder events can be found in the appendix.

> See Appendix A3 for tips on facilitating events.

7.4 ROOT-CAUSE ANALYSIS

If you don't diagnose the real cause of a problem, you can't solve it. Use Lean Six Sigma's **root-cause analysis** techniques early in the development process to aid in the diagnosis. For example, following a merger of two banks, data discrepancies keep appearing despite repeated efforts to fix them. Root-cause analysis traces

the problem back to data duplication between the two banks' systems. The problem is solved by normalizing the data or merging the systems.

Sometimes a need that stakeholders bring forward is really a proposal for a solution. In this case, you use root-cause analysis to identify the underlying need. For example, one of my clients was a software consulting company looking to enhance a software system used by its compliance auditors. The auditors had told analysts they needed reporting features they'd seen in a competitive product. Root-cause analysis revealed the *real* need: to mitigate the risk of missing the opportunity to bid on government contracts if the company didn't meet a compliance deadline. Once this problem was correctly identified, the team was able to focus enhancements on those features the software system needed for compliance, significantly reducing the risk of missing the deadline. A similar approach can be used to trace an opportunity back to a core benefit (e.g., tracing the opportunity presented by new mobile payment technologies to greater customer convenience).

Table 7.1 is an overview of the root-cause analysis approach.

Table 7.1 *Root-Cause Analysis: At a Glance*

What?	**Root-cause analysis**: A set of techniques for uncovering the root causes of a presenting symptom. The approach encompasses a family of tools such as Five Whys (asking Why? repeatedly) and cause–effect diagrams.
When?	At the start of an initiative.
Why?	Ensure the real problem is addressed so that symptoms do not recur.
Tips	Create analysis artifacts live, during facilitation events, so that attendees can visualize cause–effect relationships as they are discovered.
Deliverables	Root causes, Five Whys graph, cause–effect diagram, cause–effect tree

We look at the following root-cause analysis tools:

- Five Whys

- Cause–effect diagrams
- Cause–effect trees

7.4.1 Five Whys

The Five Whys method is just what it sounds like. You ask, “Why?” repeatedly until a root cause is found. There is nothing magic about the number five; it’s just a rough approximation of the number of times one has to ask the question to get to the root cause.

As an example, suppose stakeholders present the problem of decreasing revenues. You ask the first why question: “Why has the decrease occurred?” You learn that it’s due to an increase in voluntary churn (customers choosing to leave the company). Next, you ask, “Why has voluntary churn increased.” You learn that it is because customer loyalty is weak. Again, you ask why. You learn that competitors have more robust customer loyalty programs. Figure 7.2 shows the Five Whys diagram you create during this conversation.

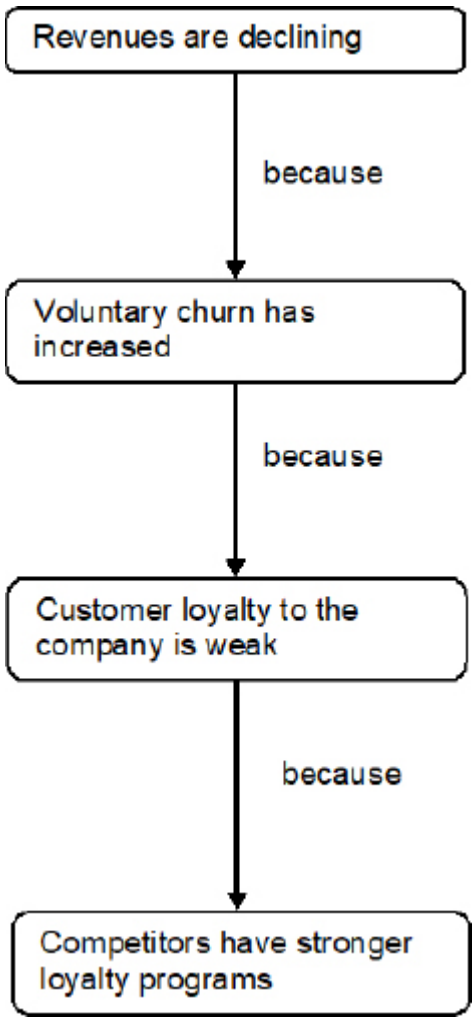

Figure 7.2 Example of a Five Whys diagram for decreased revenue

The figure illustrates the train of causes from the effect "Revenues are declining" back to the root cause of the problem, "Competitors have stronger loyalty programs." In a similar way, you can trace an expressed want back to a root need.

Since root-cause analysis is one of the first analysis activities in new product development, it's also a fitting place for our case study to begin. We follow this case through most of the book as we walk through the Agile Analysis and Planning Map (see Chapter 4, "Overview of Analysis and Planning Activities across the Agile Development Lifecycle," Figure 4.1).

BLInK Case Study Part 1: Five Whys

Background

In the introduction to the book, we learned of Better Living (BL) Inc., a company that offers insurance products in personal and commercial lines. The commercial lines products offer employer-sponsored group health insurance to employees. Employees enroll in a group insurance plan. All those in the plan pay a community rate, set for the group on the basis of factors such as age, geography, and smoking habits. The premium is sponsored by the employer, with the employee paying any remaining costs.

According to industry reports, costs to cover an employee and family with group health insurance have nearly tripled in the last fifteen years.[2] To slow this increase, the federal government (in our fictional example) has passed legislation that permits insurance companies to provide enrollees with insurance benefits, such as rate reductions, for making healthier lifestyle choices. This presents BL with the opportunity to develop innovative usage-based insurance (UBI) products that personalize benefits on the basis of healthful behaviors.

[2] Christina Merhar, "Small Business Q&A's: How Group Health Insurance Works," PeopleKeep, Inc., March 25, 2015, https://www.peoplekeep.com/blog/small-business-health-insurance-qa-guide

The Ask

As a senior business analyst, you've been asked to facilitate a meeting of product managers, marketing executives, the sponsor, and other business stakeholders to identify the potential benefits of exploiting the opportunities provided by the new legislation.

> **Tip**
>
> Use the Five Whys approach to structure your questions during the event. Record responses live during the event using a Five Whys diagram.

Preparation

To prepare for your meeting, review the business-domain terminology that follows. (It is used in this and subsequent case studies.)

Business Terminology

- *Group insurance*: Insurance provided to a group of people (e.g., employees of an organization) under a joint group plan
- *Policy owner*: Person who pays for the policy
- *Subscriber*: Someone covered under an insurance policy
- *Loss ratio:* The total amount paid out to subscribers divided by the amount paid in

What Transpires

You begin by asking attendees why the opportunity to offer behavior-based insurance products benefits BL. Stakeholders reply that it would provide a treasure-trove of behavioral data on enrollees.

You ask, "Why does behavioral data benefit the business?" You learn it's because it improves BL's ability to predict undesirable health events.

You ask, "Why does a better ability to predict health events benefit the business?" You learn that it allows BL to forecast loss ratios for different premium price points more reliably.

You ask, "Why does reliable loss-ratio forecasting benefit the business?" You learn that getting the loss ratio right is the key to competitiveness and profitability in the industry: If it's too low, you lose customers to competitors. If it's too high, you lose money on each customer.

Deliverables

Deliverable 1: Five Whys Diagram

You create the deliverable shown in Figure 7.3 during this step, based on stakeholder answers to your questions. You identify the root benefit as *increased*

competitiveness and profitability due to better loss ratio forecasting.

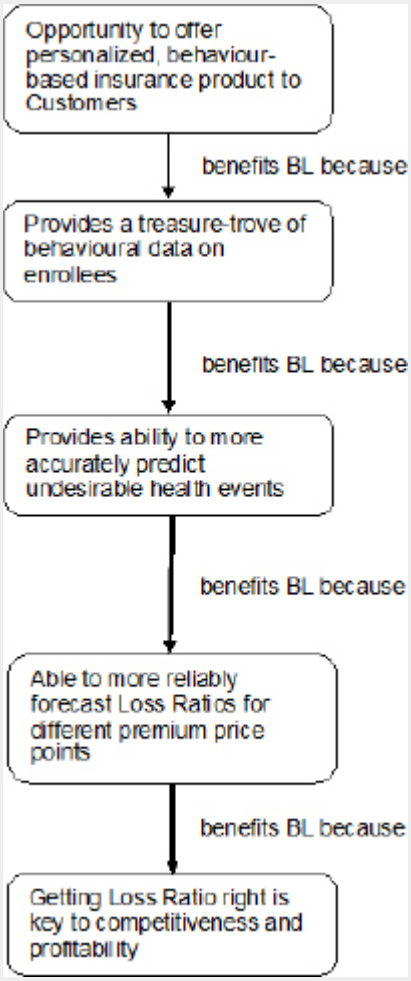

Figure 7.3 *BL insurance: Five Whys diagram*

Case Study Retrospective

As a result of this root-cause analysis, it is now clear that the core business benefit to the company is increased competitiveness and profitability due to the enhanced capability to forecast loss ratios. The analysis will help direct future development on features that generate rich datasets that will improve financial forecasting.

7.4.2 Cause–Effect Diagrams

Whenever you ask why, you can get more than one answer—and those answers may lead to more questions with more answers. Though elegant in its simplicity, the Five Whys method doesn't have a way to map multiple causes. A **cause–effect diagram** does. Use it when there are multiple answers to the question, "Why?"

The diagram is also referred to as an **Ishikawa diagram** or **fishbone diagram**. Figure 7.4 is a template for this type of diagram.

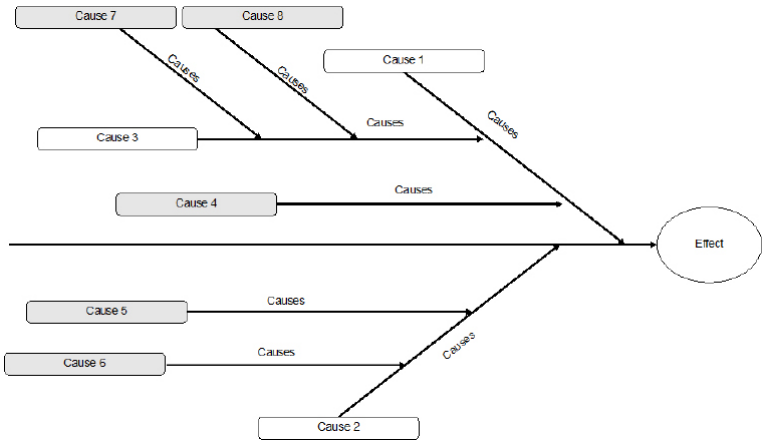

Figure 7.4 *Cause–effect diagram template*

7.4.2.1 Facilitation Tips

Start the diagram by drawing a horizontal line—the spine—with an ellipse at one end—the fish head (indicated at the right edge of Figure 7.4). Ask stakeholders to specify the **effect**—the presenting symptom that triggered the initiative. The effect can be an undesirable symptom you wish to trace to its root causes, a presenting opportunity traced to root benefits, or an expressed need traced to a core need. Indicate the effect inside the head of the diagram, as indicated in Figure 7.4.

Now, ask stakeholders *why* the effect occurs. For each cause you discover, draw a new line away from the spine. Label the tip of the new line with the cause, as shown for causes 1 and 2 in Figure 7.4.

Next, ask stakeholders why *each* of these items occurs and map those answers as indicated in Figure 7.4 for cause 1 (shown to be due to causes 3 and 4) and cause 2 (linked to causes 5 and 6).

Continue asking about each cause until you come up against a dead end. For example, causes 7 and 8 are dead ends. Additional why questions do not yield useful knowledge. For example, any further causes are outside of the organization's sphere of influence.

The dead ends on the diagram are the *root causes*. In Figure 7.4, they're the shaded causes: 4, 5, 6, 7, and 8.

7.4.2.2 Example of a Cause–Effect Diagram

Let's suppose you've convened a group of stakeholders to examine the reason for declining revenues. You soon learn there are multiple causes, so you use a cause–effect diagram for the analysis. Figure 7.5 illustrates the diagram developed during the meeting.

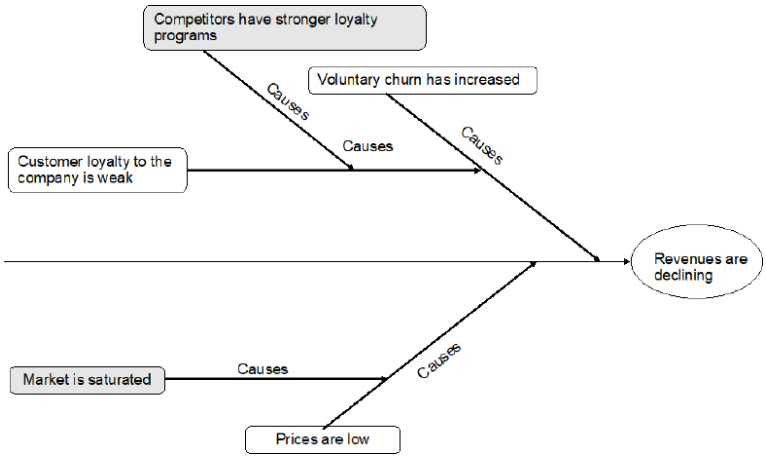

Figure 7.5 *Cause–effect diagram declining revenues example*

Figure 7.5 traces declining revenues to two root causes: market is saturated and competitors have stronger loyalty programs. The company may decide to address these causes by seeking out new markets and by strengthening its loyalty programs.

BLInK Case Study Part 2: Cause–Effect Analysis

Background

Recent industry reports have highlighted growing concerns in the public about rising healthcare costs. BL Inc. is searching for ways to attract new customers

and improve customer retention by addressing those concerns through novel product offerings.

The Ask

You've been asked to facilitate a meeting of business SMEs to identify the root causes of rising healthcare costs.

Following are the deliverables for the event:

- Deliverable 1: Cause–effect diagram tracing the causes of rising healthcare costs
- Deliverable 2: List of root causes

> **Tip**
>
> Use a cause–effect diagram instead of Five Whys whenever there is likely to be more than one reason for an effect, as is the case here with respect to rising healthcare costs.

What Transpires

As attendees arrive, you draw the spine of the diagram, writing "Rising healthcare costs" at its head.

You ask stakeholders, "Why are healthcare costs rising?" A data analyst replies that it's because of an *increase in the average number of claims per individual*. You indicate this cause on the diagram.

Others point out that there's also been an *increase in the average cost per health event*. You add the cause to the diagram. You ask, "Why is there an increase in the average cost?" The only answers you receive have to do with pricing factors over which BL has no influence, so you end that line of questioning.

Pointing to the first cause, you ask, "Why has there been an increase in the average number of claims?" Stakeholders reply that this is because of the declining health of enrollees. You continue this process until all branches lead to dead ends—signaling that the root causes have all been found.

Deliverables

Deliverable 1: Cause–Effect Diagram

Figure 7.6 is the cause–effect diagram resulting from the analysis.

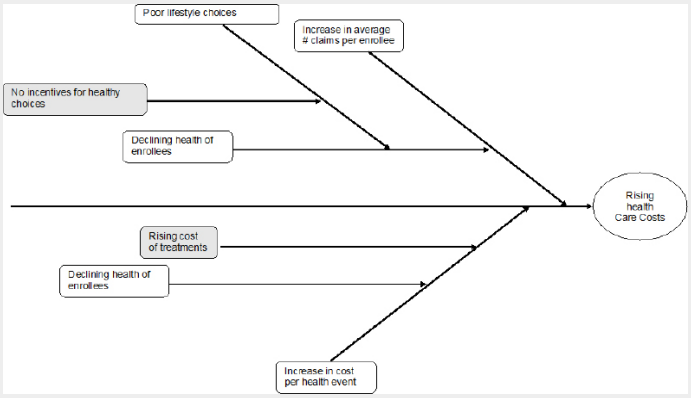

Figure 7.6 *Cause–effect diagram for BL case study*

Deliverable 2: List of Root Causes

- Lack of incentives for healthy behavior
- Rising costs of individual treatments (drugs and procedures)

Case Study Retrospective

Of the two root causes, lack of incentives for healthy behavior is determined to be the area in which BL can have the most significant impact. The company will investigate the opportunity to attract and retain customers by helping them lower their healthcare costs through products that incentivize healthy behaviors.

Case Study Epilogue

Following this investigation of the root causes of rising healthcare costs and the earlier Five Whys analysis of the opportunity to develop personalized products, BL has decided to proceed with the development of a new commercial line product, BLInK, **B**etter **L**iving through **In**surance **K**nowledge. The new product uses personalized data to encourage customers to adopt healthy behaviors. The advantages to the insured are lower healthcare costs as a result of improved health and rewards for healthy behaviors, such as reductions in gym memberships. The benefit to BL is improved loss-ratio forecasting due to data harvested from the product, resulting in higher profitability.

7.4.3 Cause–Effect Trees

You may have noticed that in Figure 7.6, *declining health of enrollees* appeared twice. This duplication occurred because it was a cause that had two different effects. Cause–effect diagrams don't have any explicit mechanism for indicating this kind of relationship or for showing other complex relationships—such as instances where *multiple* causes have to occur together before an effect can happen. **Cause–effect trees** address these shortcomings. Use them if you need to analyze several presenting problems holistically because their causes are intertwined. The trees also enable stakeholders to identify areas of improvement that have the potential to address multiple issues simultaneously.

7.4.3.1 Legend

Figure 7.7 shows a cause–effect tree legend that is adapted from the original form invented by Eliyahu M. Goldratt.[3]

[3] James F. Cox and John G. Schleier, *Theory of Constraints Handbook* (New York: McGraw Hill, 2010).

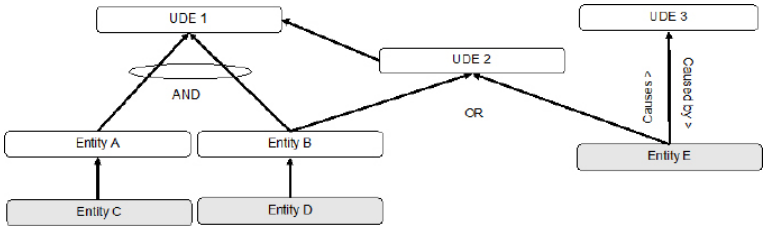

Figure 7.7 Cause–effect tree legend

Each rounded rectangle on the diagram represents an entity. An **entity** is an item that may be a cause, an effect, or both, based on its relationship to other entities. Each arrow points from a cause to an effect (e.g., in Figure 7.7, C causes A). If an effect has more than one cause, more than one arrow will point to it. When these arrows are unadorned, they represent on OR relationship, meaning either entity, on its own, causes the effect. For example, if either B or E occurs, then UDE 2 will happen. When two or more arrows are lassoed by an oval (a bar may also be used), an AND relationship is indicated, meaning *all* of the causes must occur to generate the effect.

The diagram includes special entities referred to as **undesirable effects (UDEs)**. These represent unacceptable symptoms.[4] Examples of UDEs are high defect levels, undesirable turnaround times, low market share, bugs, and any functionality that does not operate as expected. You may use parenthesis to indicate the stakeholder affected by the UDE—for example, *UDE 1: (Patient) Lack of access to medical reports.*

[4] Cox and Schleier, *Theory of Constraints*, 394–395.

7.4.3.2 Creating the Diagram

Begin the analysis by asking stakeholders to identify the symptoms that the business is experiencing. As these symptoms are detected, add them as UDEs near the top of the diagram. For each UDE, identify the stakeholder concerned about it (e.g., customer, business, or employer).

Tracing back from these UDEs, ask stakeholders to identify their causes—adding them to the diagram as they are discovered.

If there is more than one cause for an effect, ask if they both need to happen for the effect to occur. Use the

appropriate notation to document the response. Once no more causes are discovered, review the diagram with stakeholders and list the dead-end entities as root causes.

> Always choose the root-cause-analysis tool that can handle just as much complexity as you need for the situation—but no more. If there is only one problem under consideration, use the Five Whys method for a preliminary analysis. Use cause–effect diagrams if you need to do a deeper dive. If there are multiple, interrelated problems, use cause–effect trees.

BLInK Case Study Part 3: Cause–Effect Tree

Background

The prior cause–effect analysis has sparked a conversation within BL about other issues and problems the company is dealing with and whether it might be possible to address them, as well, during the upcoming initiative.

The Ask

You've been asked to bring stakeholders together to broaden the root-cause analysis.

Following are the deliverables for the event:

- Deliverable 1: Cause–effect tree mapping the undesirable effects brought up during the meeting to their root causes
- Deliverable 2: A list of UDEs and the stakeholders affected by them
- Deliverable 3: A list of root causes and the UDEs they contribute to

Preparation

To prepare for your meeting, transpose the causes and effects from the cause–effect diagram you created previously (see Figure 7.6) to a draft of the new deliverable, a cause–effect tree.

> **Tips**
>
> Use a cause–effect tree for this event because you'll be examining multiple problems that are likely to be interrelated.
>
> Begin the meeting by reviewing the cause–effect tree draft you created in preparation for the meeting. Revise the tree as needed on the basis of the review. Ask stakeholders about any other problems they are aware of, and add these to the model as UDEs. Ask why these UDEs occur, and map the resulting relationships. Continue as described in the preceding section until the root causes have been found.

What Transpires

You review the draft diagram. You ask attendees to cite any other problems they are aware of, from the standpoints of BL Inc. and its customers:

- A representative of BL's financial division reports that the business is concerned about decreasing profitability.
- Underwriters mention undesirable loss ratios—the amount paid out vs. paid in.
- A data analyst brings up another disturbing trend for the company—the increase in the average amount paid out per subscriber.

You represent these as UDEs on the cause–effect tree diagram.

You ask attendees about customer issues:

- A customer relations manager offers that employers have identified increased absenteeism as a primary concern.
- A marketing manager notes that subscribers are increasingly concerned about their declining health.

You add these UDEs to the diagram.

Working backwards, you ask "Why?" for each UDE now on the diagram, adding each cause and its relationships to other entities you discovered.

You ask what causes the first UDE, *decreased profitability*, and learn that it is due to three issues:

- Undesirable loss ratio. (You previously identified this item as a UDE.)
- Pricing is not competitive enough to attract and retain customers.
- Low brand loyalty

You ask stakeholders if any of these causes would be enough on its own to decrease profitability:

- Stakeholders tell you that the loss-ratio issue reduces profitability.
- However, they also offer that the last two issues—uncompetitive pricing and low brand loyalty—must both be present.

You "lasso" these causes, indicating an AND relationship.

You continue with this process, asking "Why?" about each entity until you reach dead ends (root causes).

Deliverables

Deliverable 1: Cause–Effect Tree

Figure 7.8 illustrates the diagram developed during this meeting.

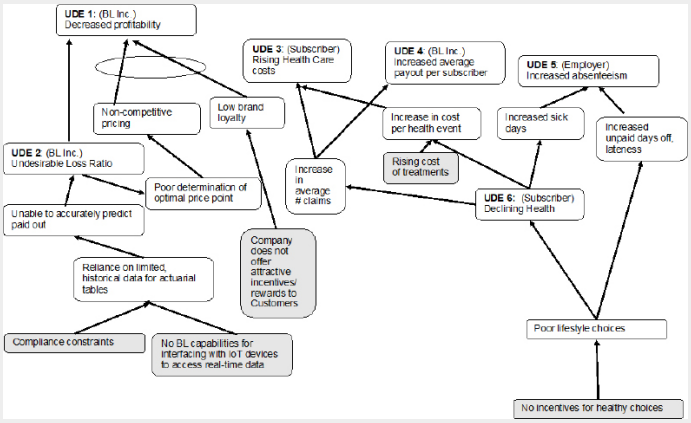

Figure 7.8 *BLInK cause–effect tree*

Deliverable 2: UDEs

The following UDEs are identified:

- UDE 1: (BL) Decreased profitability
- UDE 2: (BL) Undesirable loss ratio
- UDE 3: (Subscriber) Rising healthcare costs
- UDE 4: (BL) Increased average payout per subscriber
- UDE 5: (Employer) Increased absenteeism
- UDE 6: (Subscriber) Declining health

Deliverable 3: List of Root Causes

The following root causes are identified:

1. Compliance constraints—contributing to UDE 1 and UDE 2
2. No BL technical capabilities for interfacing with Internet of Things (IoT) devices to access data—contributing to UDE 1 and UDE 2

3. Company does not offer attractive incentives or rewards—contributing to UDE 1
4. No incentives for healthy choices—contributing to UDE 3, UDE 4, UDE 5, and UDE 6
5. Rising cost of treatments—contributing to UDE 3

Case Study Retrospective

The analysis strengthened the business case for BLInK—broadening the range of problems it would address. The analysis points product development in the direction of features that provide a rich source of data that can be used for actuarial tables, encourage healthy behaviors, and contribute to a more attractive rewards program.

Case Study Epilogue

After the meeting, stakeholders review their conclusions. They exclude root cause 1, *compliance*, because recent legislation that has removed compliance as an area of concern.

Reviewing the proposed BLInK program, you note that that the product will use a subscriber's health data to encourage healthy behaviors by offering benefits for healthy choices. You facilitate an examination of the program's impact on the remaining root causes.

Root causes 2, 3, and 4 are all found to be areas in which BLInK would have a positive impact. The product would provide the technical capabilities required to collect data from IoT devices. Moreover, it would deliver an attractive rewards program to customers, thereby increasing loyalty and providing incentives for healthy choices.

Based on the analysis, you conclude that BLInK promises to improve outcomes for the following UDEs:

- UDE 1: (BL) Decreased profitability
- UDE 2: (BL) Undesirable loss ratio
- UDE 3: (Subscriber) Rising healthcare costs
- UDE 4: (BL) Increased average payout per subscriber
- UDE 5: (Employer) Increased absenteeism
- UDE 6: (Subscriber) Declining health

7.5 SPECIFYING A PRODUCT OR EPIC

You can use the Connextra template to summarize the understanding you reached through root-cause analysis for a new epic or product. For example, in the BLInK case study, you understood the core benefit of the product: increased competitiveness and profitability due to better loss-ratio forecasting. In the Connextra format, you express this as follows: “As a group insurer, I want to gather lifestyle data so that I can improve loss-ratio forecasts.” In one sentence, this formulation identifies the primary beneficiary, the primary functionality, and the beneficial outcome. Another example of an epic written in this format is, “As a supply chain manager, I want to introduce dropship capability so that top-line sales can be increased without the inventory expense.”

Specify acceptance criteria for an epic that describe, at a high level, the minimum requirements for completion. The following example expresses the business objective that must be achieved, “Legacy system can be retired.”

> **Epic:** Modernize customer loyalty program
>
> **Acceptance Criteria:** Implementation of this epic means that the legacy system can be retired.

We examine acceptance criteria for epics and features at greater length in Chapter 10, “Quarterly and Feature Preparation,” section 10.9.

7.6 THE PROBLEM OR OPPORTUNITY STATEMENT

The problem or opportunity statement is a less concise but more informative summary of the business case for a product or change initiative, such as an epic.

The statement addresses the following **Five W** questions:

- *What* is the root problem (or opportunity)?

- *Whom* does it affect?
- *Where* does the problem occur?
- *When* does it occur?
- *Why* should we care?

You can articulate the problem statement in a paragraph, a section of a business case, or a sentence or two, using the following template:

> The problem/opportunity of [*what?*] affects [*who?*] [*where?*] [*when?*], the impact of which is [impact on the customer or business—the *why*].

In the template, the **impact** refers to the undesirable effect or symptom; the **problem** is the root cause of those effects.

> You can add a sentence that turns toward a solution as follows: "A successful solution would [benefits]."

In Scaled Agile Framework (SAFe), a similar format is used as part of the epic hypothesis as follows:

> "For [customers (the *who*)] who [do something (the context, or *when*)], the [solution] is a [something (the *how*)] that [provides this value]; unlike [competitor, current state], our solution [does something better (the *why*)]."[5]

[5] Scaled Agile Framework (SAFe), "Epics," Aug. 3, 2020, https://www.scaledagileframework.com/epic

The SAFe epic hypotheses template contains the following items:[6]

[6] SAFe, "Epics."

- Date of entry
- Epic name

- Epic owner
- Description—elevator pitch for the epic
- "For . . . who" template, as described earlier
- Business outcomes (objectives)
- Leading indicators that will be used to validate hypotheses
- Nonfunctional requirements

BLInK Case Study Part 4: Problem Statement

Background

Following up on the previous root-cause analysis workshops, you invite stakeholders to a meeting to craft the problem statement.

The Ask

The objective is to craft the following deliverable: problem statement.

> **Tips**
>
> Use the Five W questions *(who, what, where, when, why)* to structure your conversation with stakeholders. Discuss the deliverables of the prior case study. Write up the root causes identified in the analysis as *problems* in the problem statement. Identify the UDEs as *impacts*.

What Transpires

You review the results of the previous analysis with stakeholders, using the problem statement template to guide your questioning. You begin by asking stakeholders, "*What* are the root causes that BLInK is expected to address?" Stakeholders concur on the following causes:

- Rising healthcare costs
- Increased average payout per subscriber
- Increased absenteeism
- Declining health

Next, you ask, "*Who* is affected by these symptoms?" Attendees concur that, as indicated in the prior analysis, the affected stakeholders for these root causes are employers, subscribers, and the business (BL). You learn that all of these stakeholders are affected nationwide.

You ask *when* the problems impact stakeholders and learn that they do so on a daily basis. You ask *why* stakeholders are concerned, as you review the UDEs discovered in the prior analysis.

Deliverables

The following deliverable was created:

Deliverable 1: Problem Statement

The problems of rising healthcare costs, increased average payout per subscriber, increased absenteeism, and declining health affect employers, subscribers, and the business (BL) nationally every day, the impact of which is declining health, double-digit increases in healthcare costs for the average subscriber, and lost productivity due to absenteeism for the employer. A successful solution would incentivize healthy behaviors, resulting in reduced healthcare costs and absenteeism due to illness while providing BL Inc. with more accurate data with which to price premiums and a competitive edge in the marketplace.

Case Study Retrospective

The *problem statement* communicates a shared understanding of the problems BLInK is expected to solve and the stakeholders who will benefit.

7.7 THE PRODUCT PORTRAIT

We're about to develop the understanding we've been developing into a product vision statement, customer objectives, stakeholder analysis, and many other informative BA information artifacts. These artifacts should be transparent to business stakeholders and members of the development organization in order to promote a shared understanding of the product. The **product portrait** is a publicly posted chart—or information radiator—that provides that transparency. It also reduces time lost to interruptions by answering critical questions about the product.

Table 7.2 provides an overview of the tool.

Table 7.2 The Product Portrait at a Glance

What?	Product portrait: An overview, displayed in a public space, that provides a snapshot of the product. Based on Roman Pichler's product canvas.
When?	Create the portrait during initial preparation for a new product or significant change. Update it as needed.
Why?	▪ Promotes transparency. ▪ Reduces interruptions to the team.
Tips	Use the portrait to facilitate visioning workshops, working through the template from left to right to support a customer-driven analysis. Post it in a public place and update it often.
Deliverables	Product portrait includes vision statement, stakeholders, objectives, assumptions, metrics, design sketches, and features.

7.7.1 The Product Portrait Template

Figure 7.9 is a product portrait template. The template is based on Roman Pichler's product canvas,[7] with elements from lean startup and other sources. Adapt it according to your needs.

[7] Roman Pichler, "The Product Canvas" [blog post], July 16, 2012, https://www.romanpichler.com/blog/the-product-canvas

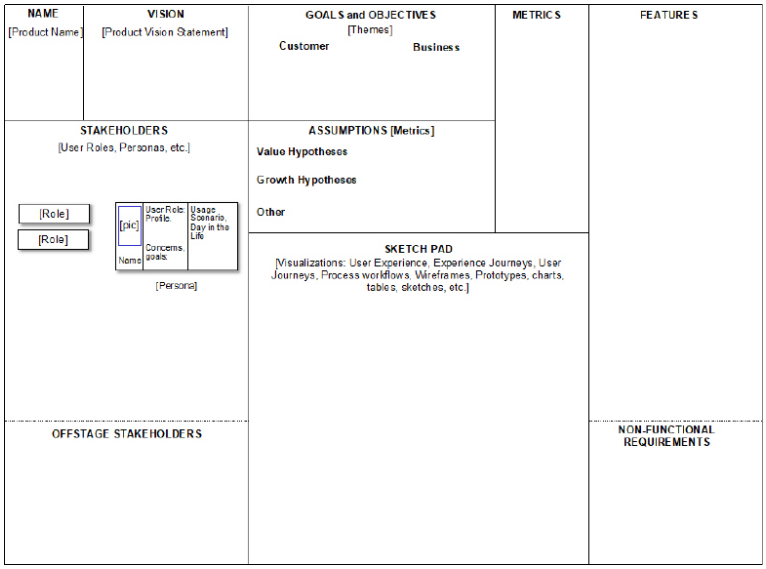

Figure 7.9 *Product Portrait Template (adapted from Pichler's Product Canvas)*

Use the product portrait as a *facilitation tool* during visioning workshops, advancing from the top to the bottom and left to right. Follow this sequence: vision, stakeholders, goals, objectives, assumptions, metrics, and so on. In this way, the product portrait supports a customer-focused process.

Use the sketch pad area of the product portrait to communicate visual artifacts developed during the product's analysis and design. Pictures have significant benefits over words: they take advantage of our natural ability to use "parallel processing" when handling visual input. The sketch pad may include artifacts such as the following:

- User journeys
- Process models
- Wireframes and other user interface design items
- Charts and tables

7.8 CRAFTING THE PRODUCT AND EPIC VISION STATEMENTS

Root-cause analysis looks backward to understand the underlying causes of problems. Vision statements look forward.

7.8.1 The Product Vision Statement

The **product vision statement** looks forward, describing the change users will experience when the product is deployed to the market and what the company hopes to accomplish by developing it. The responsibility to articulate the product vision statement and supporting objectives belongs to product-level POs and business analysts. A team-level analyst is responsible for understanding the product vision and objectives, communicating them to the team, and ensuring that the requirements support them.

7.8.2 The Epic Vision Statement

Similarly, an **epic vision statement** describes the future state after an epic—a significant change to an existing product. Within the problem statement format, the epic vision statement can serve in place of the final sentence, "A successful solution would. . . ." Use the epic vision statement for the elevator pitch within the SAFe epic hypotheses template. The articulation of the epic vision statement is the responsibility of the area PO or product-level PO, depending on the scope of the change. A team-level analyst is responsible for communicating the epic vision and its objectives to the team and ensuring that the requirements support them.

7.8.3 Properties of Well-Crafted Product and Epic Vision Statements

The product vision statement should express the product's most inspiring purpose rather than its more mundane objectives.[8] In line with full-potential planning (see Chapter 9, "Long-Term Agile Planning," section 9.4), it should be bold and innovative (e.g., to be

first in customer service or to dramatically increase market share within three to five years). Similarly, an epic vision statement expresses an inspiring reason for a change initiative rather than describing an incremental improvement.

[8] Roman Pichler, "8 Tips for Creating a Compelling Product Vision," October 8, 2014, https://www.romanpichler.com/blog/tips-for-writing-compelling-product-vision

A vision statement should be short so that it can be easily remembered and propagated throughout the organization and out into the market. John Kotter identifies the following properties of a well-crafted vision statement:[9]

[9] John Kotter, *8 Steps to Accelerate Change in Your Organization*, Kotter Inc., 2020, https://www.kotterinc.com/wp-content/uploads/2020/06/2020-8-Steps-to-Acceperate-Change-eBook-Kotter.pdf

- Communicable
- Desirable
- Creates a verbal picture
- Flexible
- Feasible
- Imaginable
- Simple

An *agile* vision statement articulates a view that leaves ample room for **emergence**, whereby new features are discovered by observing how customers use the product rather than predetermining what will be useful. To allow for emergence, don't specify features in the vision statement; focus, instead, on the raison d'être for the product or change. For example, Instagram's product vision statement is about "capturing and sharing the world's moments." Note how this statement describes the *value* of the product but does not specify whether those moments are captured as photos, video, or virtual

reality. This lack of specificity is deliberate because it does not place unnecessary constraints on how the product will evolve. Had Instagram's vision statement specified photos, it would have limited the potential scope of the product in the minds of users, the business, and product developers.

The product vision statement should be designed for longevity. However, it should be revised if the company wishes to reorient the product in a fundamental way in response to changes in the market or public perception of the product. For example, Facebook changed its product vision statement in 2017 from "making the world more open and connected"[10] to the statement, "to give people the power to build community and bring the world closer together."[11] As this example also illustrates, it's not enough to change the vision statement. The change has to percolate down to changes in the product and the culture of the development organization.

[10] Nick Statt, "Mark Zuckerberg Just Unveiled Facebook's New Mission Statement," *The Verge*, June 22, 2017, https://www.theverge.com/2017/6/22/15855202/facebook-ceo-mark-zuckerberg-new-mission-statement-groups

[11] Facebook, "About," https://www.facebook.com/pg/facebook/about

7.8.4 Vision versus Mission Statements

Some organizations have vision *and* mission statements. The product vision statement describes an envisioned future, while the mission statement focuses on what the company has to do *today and every day* to get there. In other words, the vision statement is *aspirational*, whereas the mission statement is *operational*. As former International Institute for Business Analysis (IIBA) director Alain Arsenault puts it, it's the difference between having a *vision* of a healthy environment and a *mission* to clean and sanitize every room frequently.

For example, Zappos's vision (an online store service) is the aspirational view that "One day, 30% of all retail transactions in the US will be online. People will buy from the company with the best service and the best selection. Zappos.com will be that online store."[12] Its day-to-day mission is expressed in its tagline, "POWERED by SERVICE,"[13] and through one of its core values, "Deliver WOW through service."[14] The mission statements communicate the strong service orientation of the company in its daily operations.

[12] Zappos.com, "About Zappos," https://www.zappos.com/marty/c/about-zappos

[13] Zappos.com, "What We Live By: Our Core Values," https://www.zappos.com/about/what-we-live-by

[14] "How to Write Mission and Vision Statements for B2B: And Why It Matters," Blender, https://www.themarketingblender.com/vision-mission-statements. See especially "Vision Statement Examples (A to Z)" and "Mission Statement Examples (A to Z)."

In practice, these nuances are not always observed. For example, Uber's stated *mission* is "to bring transportation—for everyone, everywhere"[15]—a statement that is more an aspirational *vision* of the future than an operational directive for the present (although that's quickly changing).

[15] Uber Newsroom, "Our Mission," Uber Technologies Inc., 2018, https://www.uber.com/en-CA/newsroom/company-info

BLInK Case Study Part 5: Product Vision Statement

Background

You've completed a root-cause analysis for the BLInK product identifying the problems the product should address.

The Ask

As a senior-level business analyst, you have been asked by senior management to facilitate a visioning workshop to craft the product vision statement for BLInK.

Preparation

You prepare the following deliverables and distribute them to participants:

- List of UDEs and the stakeholders affected by them (Case Study Part 3, Deliverable 1)
- A list of root causes and the UDEs they contribute to (Case Study Part 3, Deliverable 2)
- Problem statement (Case Study Part 4, Deliverable 1)

What Transpires

You open the meeting by explaining how to create a well-crafted product vision statement, provide examples, and review input artifacts. You ask stakeholders to envision a future where BLInK has been rolled out successfully. How will things change for the better?

One stakeholder notes that, by incentivizing healthy behaviors, BLInK reduces absenteeism (a problem experienced by employers), improves health and decreases healthcare costs (addressing concerns of employees), and at the same time reduces payouts and increases profitability (concerns of BL Inc.). You challenge stakeholders to focus on the most vital concern.

Deliverable

During the meeting, you develop the following deliverable.

BLInK product vision statement: "To incentivize customers to make healthy choices and reward them when they do."

Case Study Retrospective

Product owners and business analysts at various levels will communicate the product vision statement throughout the development organization, to business stakeholders, and to customers, where it will guide product development and marketing efforts.

7.9 STAKEHOLDER ANALYSIS AND ENGAGEMENT

A **stakeholder** is any person or group that has an impact on an initiative or is impacted by it. Stakeholder **analysis** is a process used to identify stakeholders and analyze their needs, attitudes, influence, and the initiative's impact on them. Stakeholder **engagement** includes developing a strategy and timelines for collaborating and communicating with stakeholders and the ongoing activities to carry them out.

The business analyst has primary ownership of the stakeholder analysis and engagement process. These duties include developing, conducting, and managing the process; determining the participants and timelines, and establishing procedures for maintaining stakeholder analysis information.[16] Perform a comprehensive stakeholder analysis to ensure all stakeholders are considered and as a first step in determining the requirements for a product or initiative.

[16] Alain Arsenault, in an editorial note to the author, July 2020.

Stakeholder analysis and engagement includes the following activities:[17]

[17] International Institute of Business Analysis, "Plan Stakeholder Engagement," in *BABOK v3: A Guide to the Business Analysis Body of Knowledge*, 3rd ed., (Toronto, Canada: IIBA, 2015), 31–35. The steps in the book are derived from activities in *BABOK v3*, section 3.2.

1. Identify and analyze stakeholders.
2. Plan stakeholder collaboration.
3. Plan stakeholder communication.
4. Facilitate and conduct ongoing engagement and analysis.

7.9.1 Identify and Analyze Stakeholders

The first step is to identify the stakeholders. At this point in visioning, you've begun to do that in the problem statement. If you're an outside consultant like me, you might start with one stakeholder—the person who first contacted you, such as the sponsor or the champion for the product. That person then leads you to the next interviewees, who lead you to others.

7.9.1.1 Stakeholder Checklist

If you rely solely on this process, however, you can miss essential stakeholders. For example, early in my BA career, I was so focused on users of the product, I neglected to interview the steering committee that would be reviewing my recommendations. As a result, I was unprepared for the questions they asked me during my final presentation about the project's financial justification. Since the existing software was no longer supported, it *seemed* evident that financial justification was one issue I didn't need to address. Had I thought to interview the committee members beforehand, I wouldn't have been blindsided during my presentation. Fortunately, I was saved by the product's champion, who explained why there was no choice but to move ahead. Ever since, though, I've made a habit of using a checklist to make sure I *consider* every kind of kind of stakeholder—even though I don't expect to include them

all. See Appendix A5 for a checklist of stakeholder types and their contribution to the analysis. Other sources of information for identifying stakeholders include company websites, internal corporate groups, organization charts, process maps, and documentation from previous initiatives.[18]

[18] Arsenault, personal communication, 2020.

7.9.1.2 Stakeholders List, Roles, and Responsibilities Table

Refine the stakeholder analysis by investigating the involvement of each stakeholder in the initiative and with the product. Table 7.3 is an example of **stakeholder lists, roles, and responsibilities**, an artifact used to document the results of stakeholder analysis.

Table 7.3 Stakeholders List, Roles, and Responsibilities

Stakeholder Group (User Group, Business Function, Title, etc.)	Contact	# in Role	Location	Impact on Stakeholder		Involvement of Stakeholder with Initiative		Attitude toward Change (e.g., Supports, Resists)	Expectations of Stakeholder	Expectations of Team toward Stakeholder
				Level	Primary Impact	Level	Primary Function with Respect to Initiative (e.g., Investor, Shareholder, Vendor, Proxy User)			
Director of new product development	R. Hamdi	1	Dallas	H	Improved analytics for data-informed decision-making	H	Sponsor			
Chief financial officer	H. Groot	3	Los Angeles	L		H	Steering committee member (approve budget, monitor progress)			
Marketer	J. Gupta	10	New York, Dallas	H	Ability to launch campaigns across multiple platforms	M	Business SME, user			

Use techniques like *user role modeling workshops* and *user personas* to refine the stakeholder analysis further.

> To read about personas, see Chapter 10, section 10.2. To learn about user role modeling, see Chapter 10, section 10.18.

7.9.2 Plan Stakeholder Collaboration

Next, determine how each stakeholder or stakeholder type will collaborate with the development team.

Gain agreement on the following points:

- Forum for the collaboration: the method of engagement, such as requirements workshops, focus groups, wikis, scheduled meetings (real and virtual), Web conference, conference calls
- Location
- The expected type of involvement (e.g., as an active collaborator, tester)
- Time expectations (e.g., full-time team member, one hour per week)
- Responsiveness (e.g., same-day response to urgent queries)

> **Tip**
>
> Don't wait until the initiative is underway to address expectations; do it as soon as possible. I learned this lesson the hard way from a former client, who continually interrupted me throughout a project for progress reports for upcoming meetings. All too often, the work was pointless, since the meetings didn't occur. Worse—it was time-consuming work I hadn't planned for when setting timelines. Prevent conflict by managing expectations up front.

7.9.3 Plan Stakeholder Communication

Use the preceding analysis to create a stakeholder communication plan that specifies the following:

- Informational requirements—information needed by the stakeholder from the team (e.g., status updates) and by the team from the stakeholder
- Forum or method of communication (e.g., summary reports, status update meetings, email, wiki, website)
- Timing, frequency of communication
- Level of granularity required
- Audience level (expert, general)
- Preferred communication style (formal, casual)

For guidelines on choosing the best means of communication, see Chapter 15, “Rolling Analysis and Preparation—Day-to-Day Activities,” section 5.15.5.

7.9.3.1 Stakeholder Impact/Influence Matrix

Use a *stakeholder impact and influence matrix* to determine the strategy used to communicate with stakeholders. Table 7.4 illustrates the four quadrants in the matrix.

Table 7.4 Stakeholder Impact and Influence Matrix[19]

Influence of Stakeholder			
	High	Ensure stakeholder concerns are addressed. Keep stakeholders abreast of issues and overall progress.	Consult stakeholders regularly about features, business objectives and targets. Keep stakeholder abreast of issues and overall progress.
	Low	Communication can be summary (e.g., public announcements).	Consult stakeholders on targeted areas where new features or changes will impact them.
		Low	**High**
		Impact on Stakeholder	

[19] IIBA, *BABOK v3*, 345.

Place each stakeholder or role in the appropriate quadrant based on the nature of their involvement, then plan a communication strategy for each stakeholder using the guidelines in the corresponding section, as shown in Table 7.4.

7.9.4 Facilitate and Conduct Ongoing Engagement and Analysis

Execute the plan. Facilitate collaboration between stakeholders and the team and communicate with stakeholders as described in the plan. Stakeholder analysis is iterative and ongoing. As you learn more about stakeholders and discover new ones, update information and plans as required.

BLInK Case Study Part 6: Identify Stakeholders

Background

Now that you've identified the main stakeholders, the next step is to analyze their relationships to the initiative and devise a strategy for communicating with them.

The Ask

You have been asked to produce the following deliverable for the BLInK venture:

- List of stakeholders, roles, and responsibilities (Draft)

> **Tips**
>
> Review the cause–effect tree you created in Case Study Part 3 (Deliverable 1, Figure 7.8). Look for references to stakeholders and the UDEs that affect them. Focus on UDEs that the product is intended to address. These will suggest stakeholder interests.
>
> Then review the stakeholder checklist in Appendix A5 to see if there are any stakeholders you missed. Document what you learn in a draft of the Stakeholders List, Roles, and Responsibilities table (Table 7.3). Do not be concerned about completing the table or including all the columns at this time.

What Transpires

You review the results of the root-cause analysis and the stakeholder checklist with attendees to derive a list of stakeholders. These include the policyholder and employer. You learn that the policyholder of group insurance plans usually is the employer. You gain consensus from SMEs to treat them as one role type, with *Policyholder* as the primary name.

You investigate other impacted stakeholders, documenting your results in a Stakeholders List, Roles, and Responsibilities table. Next, you review your list with attendees—using the Stakeholder Influence and Impact Matrix (Table 7.4) to categorize stakeholders. Finally, you discuss each quadrant and confirm the high-level communications strategy to be used for the stakeholders within.

Deliverables

Deliverable 1: BLInK Stakeholders List, Roles, and Responsibilities

You produced Table 7.5 as a result of this preliminary analysis.

Table 7.5 BLInK Stakeholders List, Roles, and Responsibilities

Name(s), Main Contact		Stakeholder Role (Type)	Impact on Stakeholder	Involvement of Stakeholder with Initiative (Responsibilities, Authority Level)
M. Grande		Policyholder (employer): Pays for the policy	Lower premiums; decreased absenteeism	Consulted
R. Hinkle		Primary subscriber (employee): Main person named under the policy	Reduced healthcare costs; improved health	Consulted
H. Groot, F. Hill		Subscriber (also called enrollee, member): Any person covered under the policy	Reduced healthcare costs; improved health	Consulted
R. Norman, Z. Branitsky		Broker: Sells policies, represents buyer	Higher revenue from new business	Consulted
X. Krieg		Agent: Sells insurance policies, represents one or more insurance companies	Higher revenue from new business	Consulted
A. G. Houseman		Underwriter: Adjudicates insurance applications, determines coverage and premiums	Better targeted pricing and coverages	Consulted
M. Lautrec		Legal and compliance auditors		Approvals
Y. Hendles		Actuary: Analyzes financial risk	More accurate forecasting and risk assessment due to a richer, more personalized dataset	Consulted, informed
Z. Nguyen		Vendor	Responsible for devices and data services	Consulted, informed
R. Yang		Sponsor	Improved profitability, loss ratio; decreased average payout per subscriber	Formal approvals
V. Chretien		Product owner		Responsible for prioritizing product backlog

Case Study Retrospective

As a result of the stakeholder analysis, you are now able to construct a communication plan that includes the voices of all key customers and stakeholders.

7.10 ANALYZING GOALS AND OBJECTIVES

The next step is to translate the product or epic vision statement into goals and objectives. Consider the perspectives of the business and the end customer: What does the company gain by developing the product or carrying out the change initiative? What are the benefits to the purchaser and the user?

Since we're about to focus on goals and objectives, let's review what we know about them from Chapter 3, "Fundamentals of Agile Analysis and Planning." A **business goal** is something to which the enterprise aspires to, such as

> Establish an enterprise-wide assortment planning capability by the end of the year.

A **business objective** is an outcome below the enterprise level (e.g., Provide the ability to promote services and benefits by the end of the third quarter). The objective can be a **learning objective**, for example, to test the hypothesis that, if users are warned before they share an article or video they haven't viewed, they can be incentivized to view the content and form their own opinion before sharing it. Companies should maintain a learning mindset throughout their lifespans.

Goals and objectives should be measurable and time-bound (e.g., the business goal to increase sales *by 10 percent* by the *end of the year*).

> See Chapter 3, sections 3.4.2.2 and 3.4.2.3 for more on goals and objectives.

7.10.1 Use Circumstance-Based Market Segmentation as a Basis for Goals and Objectives

Use circumstance-based market segmentation to identify the **jobs** that customers hire the product to do—the high-level **usages** of the product (e.g., launch a marketing campaign). If a new product is being developed, represent these usages as customer goals and objectives. For example, one job or customer goal for a gaming app is to babysit toddlers so that their parents can have some personal time.

Once the product is established, continue using circumstance-based market segmentation to reveal opportunities for new usages of the product and to improve the way they are currently implemented. For example, after the gaming app in the preceding scenario is released, further market analysis reveals a new "job" customers want to carry out through the product: parents want to protect their children from viewing harmful content while interacting online. This insight is the basis for an epic—a change initiative whose objective is provide protection across the platform by a given date.

> See Chapter 8, "Seeding the Backlog—Discovering and Grading Features," section 8.4, for more on circumstance-based market segmentation.

As noted earlier, goals and objectives must be measurable. Define the metrics that will be used to measure progress. In line with full-potential planning guidelines (discussed Chapter 9, section 9.4), specify metrics that represent *real-world outcomes*, such as increased market share and compound annual growth rate (CAGR), rather than process metrics such as total lines of written code or velocity. A success metric for the gaming app in the previous example might be a 50 percent reduction in the average time between episodes of whining.

7.10.2 Representing Goals and Objectives within the Story Paradigm

The following are guidelines for representing goals and objectives within the story approach. The benefit of doing so is that you can use a single paradigm to trace the entire evolution of a goal to detailed requirements items. However, if your organization already has an effective way to manage goals and objectives, there is no need to change it.

7.10.2.1 Represent User Capabilities as Epics and Features

If the goal or objective is to implement a high-level usage (a *job* a user can do with the product), create an epic or feature to deliver it, depending on the estimate for the item. An epic may require multiple teams over multiple quarters. A feature must be implementable within a single quarter; it may include multiple teams.

7.10.2.2 Represent Other Goals and Objectives as Themes

Represent any other goal or objective as a theme. A **theme** is a mechanism for clustering epics, features, and stories by any topic across organizational lines, product areas, and products. A single epic, feature, or story may belong to multiple themes.

Consider, for example, the objective of highlighting a new offering in all communications with customers. This objective is not in itself something users do with the product, but it affects many of the capabilities they use, such as generating customer invoices and renewals. Consequently, you treat the objective as a theme and the work to change the capabilities as epics, features, and stories.

BLInK Case Study Part 7: Craft Goals and Objectives

Background

Now that you've identified BLInK stakeholders, you invite them to an exploratory meeting to discuss the goals and objectives for developing the product.

The Ask

You've been asked to facilitate an investigation into two perspectives on goals and objectives for the product—the viewpoint of customers and the viewpoint BL Inc.

Following are the deliverable of this step:

- Deliverable 1: Customer goals and objectives (with metrics)
- Deliverable 2: Business goals and objectives (with metrics)
- Deliverable 3: List of metrics

What Transpires

You convene a meeting of sponsors, program managers, product managers, customer relationship managers, and marketing executives. You begin by summarizing the undesirable effects (UDEs) uncovered by the prior root-cause analysis. You note the issues of *poor health outcomes* and *rising healthcare costs* and suggest a subscriber objective to *improve health outcomes* and *lower costs*.

You invite those with a knowledge of the market to consider these and other customer objectives. You ask business stakeholders to describe business-side objectives. Once you've identified the goals and objectives for the product, you facilitate a discussion about the metrics that will be used as success indicators.

Deliverables

Deliverable 1: Customer Goals and Objectives (with Metrics)

- Reduce insurance premium contribution (policyholder). Metric: Percentage premium change (M7)
- Empower customers to control premium (subscriber). Metric: Satisfaction survey (M14)
- Reduced healthcare costs (subscriber). Metric: Subscriber annual payout (M15)
- Improved health (subscriber). Metrics: Number of claims per person (M4); Average annual amount claimed per subscriber (M5)

Deliverable 2: Business Goals and Objectives

- Reduce churn. Metric: Voluntary churn rate (M10)
- Increase market share. Metric: Market share (M11)
- Reduce loss ratio. Metric: Loss ratio (M3)
- Reduce number of claims. Metric: Claims per policy (M12)
- Increase the predictive capability for risk. Metric: Loss ratio (M3)
- Attract customers. Metric: Growth rate (M13)

Deliverable 3: List of Metrics

- M3: Loss ratio
- M4: Number of claims per person
- M5: Average annual amount claimed per subscriber
- M7: Percentage premium change
- M10: Voluntary churn rate
- M11: Market share
- M12: Claims per policy
- M13: Growth rate
- M14: Satisfaction survey
- M15: Subscriber annual payout

Case Study Retrospective

The goals and objectives for the product will be used to guide the prioritization of features. The metrics will enable the customer to gauge the success of the venture.

7.11 ANALYZE LEAP OF FAITH HYPOTHESES

The next step is to consider how the vision and value proposition for the product will be validated. In traditional product development, this can be achieved by using historical data to make projections. If the product is novel, you can't use this approach, because there is no history of products in its class. Instead, you validate the business case using an experimental, evidence-based approach—running low-cost MVP experiments to test the hypotheses that underlie the vision.

7.11.1 What Is a Lean Startup?

Leap of faith hypotheses and MVPs are critical tools in the lean startup approach devised by Eric Reis. Before you say, "That's not for my organization; we're not a startup," let me clarify the term. A **lean startup** is "an organization designed to create new products and services under conditions of extreme uncertainty."[20] The critical factor is not the age of the business—it's the level of uncertainty regarding the product. In fact, many of the lean startup organizations I work with are not startups in the popular sense of the term. They're in mainstream business sectors such as insurance, telecom, finance, and government. But they're involved in lean startup ventures because the products they're developing are novel. The following are some real examples of lean startups:

[20] Eric Ries, *The Lean Startup*, (New York: Random House, 2011), 34.

- A telecom developing a self-service site for customers to customize their own plans, where there is extreme uncertainty regarding whether customers will want to use the site.

- An insurance company developing products that use data from IoT devices to personalize rates and benefits (as in our BLInK case study). In this instance, there is extreme uncertainty regarding whether customers will be willing to set aside privacy concerns.
- A government agency considering offering a loan risk-assessment service. This is a lean startup for the organization because it has traditionally provided this service for free as part of the application process for its core products. Though there was data to indicate that customers would want the service, there was extreme uncertainty about whether they would pay for it.

7.11.2 What Are Leap of Faith Hypotheses?

The MVP process begins with identifying **leap of faith hypotheses**—critical assumptions about the product that must be true for the venture to be successful. In Chapter 9 and Chapter 12, "MVPs and Story Maps," you'll learn to use these hypotheses to drive MVP planning, with the objective of testing critical assumptions as quickly and inexpensively as possible.

7.11.2.1 Hypotheses for Change Initiatives

Hypothesis testing is not exclusive to the start of new product development. Continue running experiments to test hypotheses throughout the product's life. For example, Twitter is currently experimenting with a feature for shared items: the user will see one post with a similar point of view, one with a slightly different point of view, and one that is completely different.[21] The learning objective is to test the hypothesis that if users see these different takes, they'll want to dig deeper into the article or watch the full video. The benefit of running controlled experiments first is that there may be other

assumptions and effects the company hasn't anticipated. Experimentation brings those effects to light so that they can be addressed before the changes are widely released.

[21] From an interview with Jack Dorsey. Michael Barbaro, "Jack Dorsey on Twitter's Mistakes," *The Daily* [podcast], August 7, 2020, https://podcasts.apple.com/ca/podcast/the-daily/id1200361736?i=1000487397342

7.11.3 Value Hypotheses

The lean startup approach identifies two broad categories of leap of faith hypotheses: value and growth hypotheses. **Value hypotheses** are critical, unproven assumptions about the product's value.

Following are examples of value hypotheses:

- Customers will want to use this product once they see it; they just don't know it yet, because it's the first of its class.
- Customers will be willing to put up with a lower-quality product than is currently available in exchange for a low price or high convenience.
- Customers will engage heavily with the product.

7.11.4 Growth Hypotheses

You can have a product that delivers value to customers yet still does not achieve a fast-enough **growth rate** to succeed. Fast growth is especially important for new entrants because they need to capture the market quickly before incumbents have a chance to respond. The growth assumptions that must be true for the venture to succeed are referred to, in lean startup, as **leap of faith growth hypotheses**.

Following are examples of growth hypotheses:

- Each user will refer at least *x* users (viral company model).

- The right proportion of buyers to sellers will be achieved (marketplace company model)

7.11.5 Specifying Metrics

Often, teams consider metrics only after the feature has been implemented. That's too late to inform development decisions because, by then, critical development decisions have already been made. It's the responsibility of the PO with the support of the business analyst to persuade stakeholders and developers to specify metrics early in the process so that the data can be used to inform investment. The metrics should be refined over time (e.g., as MVPs are planned and tested).

Determine the metrics that will be used to validate hypotheses and measure progress toward goals and objectives. Choose metrics that measure outcomes over ones that measure process steps. For example, measure quality and cost or hospital readmission rates rather than time to complete a task.

Lean startup provides guidance for defining **actionable metrics** that isolate the impact of change over **vanity metrics**—measurements that seem to suggest improvement but are, in fact, inconclusive. Let's examine those guidelines.

7.11.5.1 Vanity Metrics

Suppose a newspaper business decides to launch a project to boost advertising revenues. To assess the program's success, it measures the publication's monthly advertising revenue before, during, and after the campaign. Sure enough, revenue rises. They conclude that the program was a success. Were they right?

Not necessarily. There could have been other reasons for the rise, such as the closing of a competing publication. The metric they chose made people *feel good* about the

project but didn't prove anything. In lean startup, such metrics are referred to as **vanity metrics**.

7.11.5.2 Actionable Metrics and Split Testing

In contrast to vanity metrics, **actionable metrics** provide the business with measurements it can act on to make decisions. The problem with the previous metric example was that it didn't mask out the background noise of everything else happening in the environment. To do that, you need to add a control group subject to the same context as the test group, except that it doesn't experience the intervention you're measuring. This approach is referred to as **split testing** or **A/B testing**.

Returning to our newspaper example, we could expose the new program to a small group (cohort) of customers while leaving another group unexposed—and measure the *difference* between ad revenue gains for each group. If there is a net improvement in revenue for the test group but not for the control group, the increase must be due to the program, since environmental changes would have affected both groups equally.

You can use this approach with more than two groups as well. For example, you can measure the conversion rates for two groups—each exposed to a different change in a user interface—and compare those measurements against a control group that doesn't experience any change.

Metrics to test value hypotheses include the following:

- Number of daily active users or number of monthly active users (a measure of stickiness)
- Conversion rate from free trial to subscription
- Frequency of opens
- Average session length

- Total time in the application (daily, weekly, monthly)
- Engagement retention (percentage of initial users who return to the application within a given period)

To block out environmental factors, perform split testing and take care to measure the *differences* in values between test groups and control groups.

Metrics for growth include the following:

- Referral rate: Average number of referrals made by each user.
- Net promoters (NP): = (% promoters) – (% detractors),[22] where

[22] Frederick F. Reichheld, "The One Number You Need to Grow," *Harvard Business Review*, December 2003, https://hbr.org/2003/12/the-one-number-you-need-to-grow.

  - % promoters = percentage of customers who respond with a 9 or 10 to the question, "On a scale of 0 to 10, how likely is it that you would recommend the product to a friend or colleague?"
  - % detractors = percentage of customers who respond with a 0 to 6.

Research shows a correlation[23] between high NP scores and growth. High NP also means a low cost to acquire a customer (CAC)—since existing customers do much of the marketing at no cost.

[23] See Reichheld, "The One Number You Need."

- Growth rate of monthly active users (MAU), where MAU = # unique users who have visited a site or used an application at least once during the month)
- Bounce Rate: % of customers who visit and leave immediately

7.11.5.3 Actionable Metrics for Enhancements

The preceding guidelines about metrics also apply once the product is established and undergoing continual improvement. Specify actionable metrics to measure the value of proposed features or revisions. As the feature is developed, test out solutions with users and collect actionable metrics. Use the resulting data to evaluate the effectiveness of alternative solutions for the feature. During beta testing, collect and analyze actionable metrics to validate the feature and determine whether to include and support it.

7.11.6 Hypotheses in Discovery-Driven Planning

Financial planners face the same challenge as do product developers when dealing with an innovative product: how to create a plan when there is extreme uncertainty surrounding the product and the market for it. **Discovery-driven financial planning** extends the MVP approach into the realm of financial planning. The financial planner works backward from the outcomes deemed necessary for a venture to be successful in order to determine the financial hypotheses or assumptions that must be made. Then the planner devises a strategy to test those assumptions in the market as quickly as possible. The steps in this process are as follows:[24]

[24] Based on the process described by Christensen and Raynor, with some changes made to the number of steps and their activities. Clayton M. Christensen and Michael E. Raynor, *The Innovator's Solution* (Boston: Harvard Business Press, 2003), 227–229.

- Identify the required future outcomes (targets).
- Identify the assumptions that must be made for those targets to be met by preparing a reverse income statement.
- Create a plan to test those assumptions in order of importance.
- Implement the plan: Test the assumptions.

- Learn: Make strategic investment decisions based on test results.

Examples of financial assumptions discovered and tested through this process include the following:

- Cost of production
- Cost to acquire a customer
- Price that customers will be willing to pay for the product
- Packaging and shipping costs
- Product lifespan

For a more detailed description of discovery-driven planning, see Chapter 18, "Achieving Enterprise Agility," section 18.10.2, and Appendix B.

7.11.7 Assumption Checklist

Provide more information about hypotheses or assumptions in an assumptions checklist. Table 7.6 illustrates an example.

Table 7.6 Assumptions Checklist

ID	Assumption	Measurement (Target)
1	Return on sales (ROS) will be high	ROS (≥16%)
2	Fewer customers will decide to leave	Voluntary churn (10% reduction)

7.11.8 Using a Milestone Planning Chart to Plan Assumption Testing

Use a *milestone planning chart* to plan the timing of assumption testing. Table 7.7 illustrates some examples of milestones.

For a complete example, see Appendix B, section B.8.

Table 7.7 Milestone Planning Chart

Milestone Event Completed	Assumptions to Be Tested
Prototypes created	(A1) A price advantage of 10 percent will be enough to lure customers away from incumbent suppliers.
Pilot production	(A2) Production costs can be kept below $100/unit.

BLInK Case Study Part 8: Analyze Assumptions

Background

Having analyzed goals and objectives, you prepare to investigate the assumptions (leap of faith hypotheses) that must be true for those objectives to be attained. You invite stakeholders to a brainstorming session to identify those hypotheses and the metrics used to test them.

The Ask

Following are the deliverables of the analysis:

- Deliverable 1: Assumptions checklist (indicating metrics)
- Deliverable 2: List of metrics

What Transpires

You review the goals and objectives of the product (see deliverables of Case Study Part 7). You explain that you will be asking stakeholders to consider what assumptions must be valid for these goals and objectives to be achieved.

For the objective *increase predictive capability*, you discover the value hypothesis, "Customers will want to continue with the program after a six-month trial." For the objective to improve health, you discover the value hypothesis, "Behaviors will improve as a result of the program." Attendees continue discussing objectives and their related assumptions until no new assumptions come up.

Next, you facilitate a review of the assumptions. You ask stakeholders to sequence them in order of criticality—with the intention that those listed first will be tested earliest. Next, you discuss the metrics that will be used to test the assumptions that have been identified.

Deliverables

Deliverable 1: Assumptions Checklist

Value Hypotheses

- (A1) Customers will want to continue after a six-month trial. (M1)
- (A2) Behaviors will improve. (M2)
- (A3) Accuracy of risk, payout predictions will improve. (M3)
- (A4) Health will improve due to participation. (M4, M5)
- (A5) Customers will like the product when they see it. (M6, M16)
- (A6) Premium will go down. (M7)
- (A7) Reluctance to share data can be overcome when a benefit is shown immediately. (M8)

Growth Hypotheses

- (A8) Subscribers will refer others. (M9)
- (A9) Acceptance will grow once use becomes widespread—at which time large immediate discounts will no longer be required. (M8, M13, M16)
- (A10) BLInK will lead to improved customer retention. (M10)

Deliverable 2: List of Metrics

Metrics

- M1: BLInK renewal rate
- M2: Lifestyle score
- M3: Loss ratio
- M4: Number of claims per person

- M5: Average claimed amount
- M6: Response rate
- M7: Percentage premium change
- M8: Penetration rate
- M9: Referral rate
- M10: Voluntary churn rate
- M13: Growth rate
- M16: Post-trial conversion rate

7.12 CHAPTER SUMMARY

Here are the key points covered in this chapter:

- Root-cause analysis is a set of techniques for tracing effects back to their initial causes. The approach includes the Five Whys technique and cause–effect graphing.
- The product vision statement is a short description of a future when the product is used by its target market.
- Perform stakeholder analysis as early as possible and continue it incrementally throughout the development lifecycle.
- Represent business goals and objectives as themes.
- Represent customer objectives as epics and features.
- Leap of faith hypotheses are assumptions that must be true for the product to be viable. Identify them at the start of a venture so you can plan to validate them in the market as soon as possible.

7.13 WHAT'S NEXT?

In Chapter 8, we'll look at activities associated with *seeding* the backlog—the determination and specification of the features it will include.