ADAPTIVE LEADERSHIP

ACCELERATING ENTERPRISE AGILITY

JIM HIGHSMITH

Forewords by GARY GRUVER, CHRISTOPHER MURPHY, *and* JOHN CROSBY

Adaptive Leadership

Accelerating Enterprise Agility

Jim Highsmith

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

ISBN-13: 978-0-13-359844-5
ISBN-10: 0-13-359844-6

First published, November 2013

Contents

Foreword by

GARY GRUVER

Jim Highsmith, one of the key founders of the agile movement, has done us a great favor by packaging up some of his key blog posts over the last decade into an easily digestible electronic book. While a lot of the agile movement over time has focused on the details of Scrum, Jim reminds us all that agile, at its core, is about much more. It is about embracing uncertainty and developing an organization that responds quickly. It is about creating processes and approaches that are designed to teach and adjust.

The agile movement is about being able to quickly respond to the ever-evolving business environment and being able to deliver ongoing value. Jim reminds us of those facts through several different perspectives in his different blog posts. He highlights the magnitude of uncertainty facing current business leaders and provides frameworks for addressing these challenges. He shows the importance of delivering value early and often with technical approaches like continuous delivery. He also points out that the biggest challenges are not technical, but rather relate to leadership and the organization change-management process.

One of my favorite sections of this book is on organizational agility, which I found representative of what we found during my time at at HP. We started with a focus of improving the productivity of our firmware development. What we found was that the changes we implemented in the firmware development process had a much bigger impact across larger parts of the organization. It changed the value proposition we were able to provide our customers and how we managed the delivery of all the components required to deliver LaserJet printers.

Throughout these blogs, Jim challenges us all as leaders of change to remember that the agile movement is about so much more than Scrum, and we should all take a broader perspective. Please join me as we look back on his perspectives and think about how we can get better at leading our organizations through uncertainty.

—Gary Gruver
vice president of release,
quality engineering &
operations, Macys.com
San Francisco, CA

Foreword by CHRISTOPHER MURPHY

I had been a follower of Jim Highsmith's writings in the blogosphere for several years when I first had the opportunity to meet Jim in person during the Agile Australia conference held in Melbourne in the second half of 2010.

During Jim's keynote address on adaptive leadership at that conference, I recall feeling particularly inspired and challenged—in the most positive sense—by his observations on leadership in dynamic and turbulent environments. For while much of Jim's thinking had derived from his experience with agile software development, I realized as I listened to the presentation that the impact was far more wide-reaching—to every level of organizational leadership.

Indeed, Jim's observations got me thinking. Change is accelerating. Why is that the case, and what is the role that technology, and technologists, are playing in that change? What does it take to not merely survive, but to actually *thrive* in an environment where change is the norm, rather than the exception? As Jim vividly demonstrates in this book, this turbulence creates enormous opportunity for those who are willing to embrace it rather than fight it. But this opportunity doesn't come automatically; on the contrary, it often requires fundamental change in leadership assumptions, models, and structures that are counterintuitive to traditional hierarchical structures and management disciplines developed during, and inherited from, the last 300 years.

It is pertinent that as I write this, I am sitting in a cottage in Derbyshire, England, in the midst of a family vacation touring the historical remains of great industries that spawned the Industrial Revolution—the significant mills, factories, canals, and mines of Northeast England. The trip has reinforced in me something that many of us implicitly know: that just as the rapid advances in 18th-century industrialization required fundamental changes in management and leadership, so do the similarly rapid technological changes of the current era. In contrast to the past, which was focused on the marshaling at scale of a largely uneducated, manual workforce, our focus now is the motivation and unleashing of the talent of highly educated, creative knowledge workers. For that, we must shed the legacies of the past and equip ourselves with a new set of leadership assumptions and models that are appropriate to the *current* revolution, not simply borrowed from the past.

For it is indeed another revolution that we are in, not only with regard to the physical hardware technologies, but perhaps more importantly for the knowledge workers who unleash their potential. The rapid increases in computing power are widely recognized and understood, and are easily demonstrated by the power of the computers that we now hold in the palms of our hands. It is less widely recognized, however, that software development—the primary method by which the advances in hardware technology are released to industry and society—has progressed in similar seismic leaps and bounds. Modern software professionals are armed with sophisticated languages, frameworks, environments, and ecosystems with which to wield their craft. That increasing sophistication is leading to an increasingly powerful ability to craft impactful software for users, in increasingly short cycle times.

Moreover, software is no longer focused on automating manual tasks—the systems of record on which so much IT effort has been expended over the last twenty years—but on software that very often *is* the business, meaning the systems that individuals, businesses, and governments use to reach and engage their users and constituents. Nimble, disruptive, and technologically driven interlopers are challenging existing business models, and entire industries, through technologically driven disintermediation. Whether traditional media, travel, real estate, publishing, retail, financial services, photography, health care, or education, the chance is that some industry is being disrupted, and that, in some way, changes in technology are contributing to that change. To understand and respond to those changes, adaptive leadership will be required.

Adaptive leadership, at its heart, is an articulation of the adaptations necessary for modern organizations to flourish in this new, technological era, and to embrace the change that it enables. As any business leader knows and will attest, and as Jim ably reinforces in this book, at the heart of any successful organization or movement is its ability to understand and articulate its own unique, ambitious mission. Jim takes this a step further: not only must that organization or movement be able to clearly articulate its mission, but it must also clearly understand the difference between responsiveness and efficiency to achieve that mission and, where responsiveness is required, implement practices around delivering a continuous stream of value by discovering and managing the flow of market and technology opportunities. It must also create the adaptive, innovative culture necessary to accomplish this. Frequently, this endeavor will involve revisiting and, as necessary, adapting or removing the vestiges of 19th/20th-century management structures such as traditional planning cycles, "command and control" management styles, hierarchy and silos, and traditional portfolio management, among others. It may sound obvious, but to be a truly *adaptive organization*, the leaders of the organization need to practice and demonstrate *adaptive leadership*.

At the organization at which I am chief strategy officer, ThoughtWorks, a global software consultancy and products business, we work for clients who are at the heart of Jim's book: organizations with ambitious missions where technology is a major enabler of competitive advantage. Many of our clients are, not surprisingly, dealing with the implications of ever-increasing turbulence and speed of change. In some instances, they are the ones creating that turbulence; they are the archetypal "disruptors" using technology to introduce a new business model or, perhaps, to disintermediate a more traditional model. In other cases, they are the more established industry leaders who are being "disrupted" and are looking to respond.

Jim's work on adaptive leadership has been of significant assistance, both to ThoughtWorks' clients and to our own business. Articulating the difference between fundamental strategies of efficiency versus responsiveness and continuous value delivery has changed our conversations with business executives, which in turn has empowered those executives to transform their organizations. Adaptive leadership has provided the language, terminology, and ideas that have helped us forge stronger partnerships with our clients, helping them to bring exciting new products and services to market in a fraction of the time they would previously have required. It has also provided guidance and ideas that have helped our own internal management development efforts, culminating in the articulation of our new market positioning bringing together our offerings and proposition into a consistent, empowering theme.

Adaptive leadership, and its underling agile principles, is no longer a set of practices for the IT department; it is a powerful business differentiator. Jim's thinking on adaptive leadership should be compulsory reading for any business leader wanting to step up to, understand, guide, and support adaptive thinking across his or her organization in the current technological era.

—Christopher Murphy
co-president and chief strategy officer, ThoughtWorks
London

Foreword by JOHN CROSBY

As I see the pace of change continuing to increase, it's become apparent to me that organizations need to find new ways to adapt, survive, and prosper. Increasingly in our digital world, a structural competitive advantage is becoming harder and harder to maintain. Organizations that are able to embrace, experiment, and learn quickly within their environment and then take that acquired knowledge to market quickly will now succeed in the long term. As Jim highlights in this book, "discovering and executing on new opportunities is the critical capability." However, achieving this goal is not easy and requires a holistic approach within the organization. Over the last decade, I've seen the agile software delivery movement make great strides in improving the quality and reliability of software delivery—increasingly so with the advent of continuous delivery. Yet I have often seen the power of the agile approach lost when this process interacts with traditional approaches to business planning and execution. These nonagile approaches still place too much emphasis on trying to manage and inspect the inputs and deliverables as opposed to enabling great outcomes. The challenge is no longer "Can we build it?" but rather "Should we build it?"

Adaptive leadership provides a framework for individuals and organizations to address this challenge. Key to this is ensuring that at the core of the organization is the ability to instigate, encourage, and inspire disruptive thinking. At lastminute.com, we've found a culture of team empowerment critical to creating this ability. Give a team clear objectives in terms of business outcomes, be clear about constraints, and then allow the team the freedom to achieve the outcomes in the best way they identify. This may sound risky to many, and without the right approach it undoubtedly is! However, our experience suggests that this courageous leadership need not be based on blind faith. Continuous delivery has provided us with the solid bedrock of delivery quality and reliability. Paired with continuous design and the key build–measure–learn tenets of the Lean Start-up movement, it ensures we are building products that customers will both want and love.

Inevitably, there have been both challenges and learnings along the way that caused us to reevaluate our thinking. This is where adaptive leadership throughout the organization is crucial. It's not just about how the initial hypotheses of a business team may have been wrong, or how the velocity of a delivery team falls below expectations, or how either group must individually solve the problem. It's now about how the entire organization can quickly understand and rapidly act upon this situation. For without this ongoing focus on reducing cycle time and increasing learning, you can be certain you'll be left behind by faster, nimbler, more adaptive organizations.

—John Crosby
vice president of product and technology,
lastminute.com
London

Preface

Over the past couple of years, I've been thinking, writing, and speaking about issues of adaptive/agile leadership and organizational transformations. Working with companies, working with the Agile Leadership Network Executive council, and speaking at conferences and internally with company leaders have encouraged me to focus my energy on management and leadership issues around responding to the opportunities created by today's technological and marketplace change.

The agile movement has greatly impacted software development over the last decade since the Agile Manifesto was signed. The two underlying themes of the agile movement have been reasonably successful (there's always progress to be made)—namely, building better software and increasing satisfaction (and fun) at work. In a growing number of companies, agile/Lean values and practices have been infused throughout the organization, although there remain too few of these pioneers.

As the agile movement moves past its ten-year anniversary, we need to reflect on our successes, identify areas ripe for improvements, and create a vision for the future. I've labeled the last item "agile imagineering" (paraphrasing from Disney). Just imagine what we could accomplish if more companies focused on values and quality; if they focused on achieving their ambitious missions; if they focused on increasing responsiveness to customers; if they focused on self-organizing at every level; if they focused on collaboration, transparency, courage, and technical excellence; if they focused on disruptive thinking and inspiring their people; and if they used the success generated from this transformation to promote social and economic justice in the world.

My personal ambitious mission is to inspire executives and managers to build adaptive organizations and provide them with the framework and tools to do it. Just as the original Agile Manifesto vision hasn't been fully realized in the past decade, I doubt my mission will be realized in the next ten years. However, one of the primary practices of agility is envisioning and evolving rather than planning and doing—it's about having a BHAG (Big Hairy Audacious Goal, per Jim Collins [2001] in *Good to Great*) and adapting over time to move toward that goal, rather than having a prescriptive plan and executing tasks.

This book is a compilation of papers and blogs I've written over the last several years plus some new material. The core ideas were articulated in a ThoughtWorks white paper and subsequently expanded upon in my blog posts. One of the problems with a blog is that older entries get lost, so I wanted to pull them out into a more readily accessible format. I've rewritten material to bring a little cohesion to the disparate blogs, while other material remains as originally written.

—Jim Highsmith
Lafayette, Colorado
September 2013

About the Author

Jim Highsmith is an executive consultant at ThoughtWorks, Inc. He has more than thirty years' experience as an IT manager, product manager, project manager, consultant, and software developer.

Jim is the author of *Agile Project Management: Creating Innovative Products*, Second Edition (Addison-Wesley, 2010); *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems* (Dorset House, 2000; winner of the prestigious Jolt Award), and *Agile Software Development Ecosystems* (Addison-Wesley, 2002). Jim is the recipient of the 2005 international Stevens Award for outstanding contributions to systems development.

Jim is a coauthor of the Agile Manifesto, a founding member of The Agile Alliance, coauthor of the Declaration of Interdependence for project leaders, and cofounder and first president of the Agile Leadership Network. He has consulted with IT and product development organizations and software companies in the United States, Europe, Canada, South Africa, Australia, China, Japan, India, and New Zealand.

Chapter 3

Deliver a Continuous Flow of Value

In a turbulent and uncertain world, it is no longer adequate to deliver value; an organization must deliver a continuous flow of value, in ever-shortening cycles. Companies must figure out how to take advantage of the continuous flow of opportunities that pass by. Some opportunities linger for years, while others come and go quickly. Adaptive organizations possess the abilities to both sort good opportunities from bad and—this is a critical piece—get the timing right. Organizations need the processes, practices, culture, and leadership to sort through hundreds, if not thousands, of opportunities and turn those potential ventures into specific ambitious missions. These missions are achieved by executing on the right opportunities at the right time. Doing this over and over again, opportunity after opportunity, creates a continuous flow of value for organizations.

Every phase of the business must operate iteratively in short cycles—from the initial identification of new opportunities, to the discovery of product requirements and business models, to the actual product development, to deployment. Companies must hone their product delivery and technology infrastructure capabilities to deliver value time after time after time. No longer will a focus on just the short term or just the long term be adequate. Instead, a simultaneous focus on multiple time horizons, a balancing of short-term solutions with quality, will enable future solutions to be continuously delivered.

The IBM study mentioned in Chapter 1 pointed out that CEOs expect technology, and the changes brought about by technology, to be a critical issue for the next 3 to 5 years. It is somewhat vexing that today many of the opportunities arise from technology, as do many of the solutions to those opportunities. Technology both disrupts and enables. We need to be better at anticipating the former and encouraging the latter. Further, technology today—from mobile devices to implanted medical devices—is driven by software. While the iPhone is a brilliant piece of hardware and its manufacturing process is highly refined, without iOS (the operating system) and applications the iPhone would be just a pretty paperweight. Thus this section on delivering a continuous stream of value focuses on software delivery processes and practices, especially those processes and practices that managers need to understand. In today's world, executives and managers, no matter which department they are in, need to understand the critical software delivery levers.

While the activities of an adaptive leader might seem endless, four critical actions contribute directly to continuous value delivery: improving speed-to-value, having a passion for quality, doing less, and building the necessary capabilities. This chapter addresses these critical actions.

Speed-to-Value

For our purposes, speed and value both merit further explanation. However, the first order of business is to examine our long-held beliefs about what constitutes performance in both business and projects. If our business objectives are to be responsive and agile, then we should start by examining how we measure success.

Flickr deploys software changes multiple times per day—and advertises this fact on its website. A medical software company deploys versions of its application software more than 75 times per year. Salesforce.com has gained a competitive advantage through its highly automated continuous integration, testing, and deployment of new software features. All of these companies, and a rapidly growing cadre of others, are reaping the benefits of speeding value to their customers.

Achieving speed-to-value reaches far beyond the early agile focus on development speed. In the early 1990s, I worked on a project at a large New York City bank. We managed to build a new application in 6 weeks (using 1-week iterations). The customer was ecstatic, IT operations not so much. After much negotiation, ops agreed to deploy our new application—one the client really wanted quickly—in 6 months.

Speed-to-value embodies two key concepts: speed and value. "Value" means that we are constantly evaluating—at a portfolio, project, capability (epic), feature, and story level—the value we are delivering to customers. It incorporates everything from calculating the ROI of projects to determining the relative (or monetary) value of features. Then on a release, milestone, and iteration level, we constantly prioritize and adjust scope based on value and cost.

The agile mantra has always been to deliver value early and often, but we have not always pushed that point to the limits of actual deployment and customer solutions. The reasons are more organizational than technical (although there have been significant enabling technical advancements).

The organizational issues are both product life cycle and business customer oriented. Although delivery teams have become agile, marketing, product management, or internal business departments have sometimes been reluctant to change their traditional modes of operation. Some product organizations have completely changed their perspective, however—from demanding commitment to features a year in advance to accepting the flexibility that agile development allows—and have profited handsomely from that responsiveness to customers.

Similarly, at the back end of the life cycle, development and operations may work closely on smooth transitions from development complete to deployment complete. Value is realized only when features are actually deployed, not when they are ready for deployment. Speed-to-value should be measured across the entire life cycle, from placement on a portfolio backlog to actual deployment.

An even more difficult change may be getting business departments to think through how they can use frequent deployments to their advantage and then change their business processes to accommodate them. They will also need to decide how to articulate the benefits of these frequent deployments to their customers. For some business departments, daily deployments of new features may have significant consequences; for others, it may not. Finding the right schedule of deployments for different groups means business departments will need to become more agile themselves.

These organizational agile transformations are often much more difficult than implementation of agile practices and principles in the engineering department, but their benefits can be extraordinary. As enterprises learn to be more adaptive, agile, flexible, or dexterous, the potential for competitive advantage multiplies rapidly.

BEYOND SCOPE, SCHEDULE, AND COST: THE AGILE TRIANGLE

> It's better to have fuzzy numbers for things that are important rather than precise numbers for things that aren't.

Many agile teams are faced with the paradox of being asked by management or customers to be "adaptive, flexible, or agile," while at the same time being asked to "conform to plan," where the "plan" is a traditional Iron Triangle plan based on scope, schedule, and cost. We ask teams to be expansive, to work closely with customers and respond to them, and to seek value—but then we penalize them for being 10% over budget.

The Agile Triangle, shown in Figure 3.1 and introduced in *Agile Project Management* (Highsmith, 2010), addresses the real goals of projects—producing value that delights customers, building in quality that speeds the development process and creates a viable platform for future enhancements, and delivering within constraints (i.e., scope, schedule, and cost). The Agile Triangle alters how we view success.

First, let's look at value. A number of studies have shown that 50% or more of functionality delivered in software is rarely or never used. Even if some of that

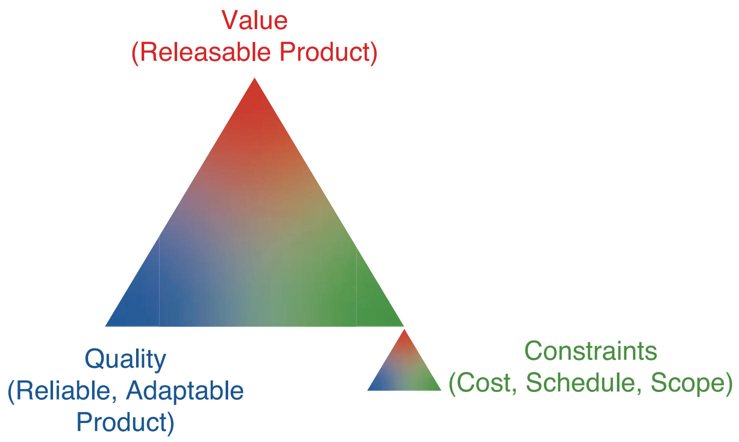

Figure 3.1 The Agile Triangle

functionality is necessary (e.g., the functionality for a year-end accounting close), there is still a huge percentage of unused functionality in most software systems. This leads the conclusion that scope is a very poor project control mechanism—we should be using value. Furthermore, rather than asking, "Did we implement all the requirements?" the question should be "Can we release this product now?" I've known projects that were deemed releasable with just 20–30% of the originally anticipated functionality—and the customers were delighted. They got their fundamental needs met—very fast!

The Agile Triangle also elevates the critical role of quality, a dimension we have given lip service to for far too long. If we are serious about quality, then it deserves a primary place in any measurement program. Quality comes in two flavors—today and tomorrow. "Today" quality addresses the current iteration or release of a product. It measures the reliability of the product: "Does it operate correctly?" If a product operates reliably, it delivers value to the customer in the form of implemented features. Products that are unreliable, ones that give incorrect answers or periodically fail completely, will fail to deliver current value.

The second quality dimension is future quality: "Can the product continue to deliver value in the future?" The ability to deliver in the future tests an application's ability to respond to business changes, both anticipated and unanticipated. While we can often use flexible designs for anticipated changes (e.g., allowing for tax table changes), the strategy to deal with unanticipated changes is different. Responding to the unanticipated future requires adaptability, and the key to adaptability is keeping technical debt low.

The final piece of the Agile Triangle is constraints—scope, schedule, and cost. It's not that these elements are unimportant, but simply they are not the goals of a project. Constraints are critical to the delivery process; they establish clear boundaries within which the team must operate. However, only one of the three elements can be paramount, and on agile projects this is normally schedule.

The Agile Triangle gives us a different way of looking at success, a way that helps resolve the paradox of adaptability versus conformance to plan.

CONSTRAINTS DRIVE INNOVATION

On a recent vacation, I visited the Mingei International Museum in San Diego. During a tour by the museum director, we were looking at a mid-1930s Santo Domingo Pueblo (New Mexico) necklace. "Interesting about this necklace," the director said, "are the 'nots'—not coral, not obsidian, but old melted phonograph records for the black element. During the Depression the Native American artists were constrained by the lack of materials. Everyone thinks that creativity and innovation are driven by freedom. In the art world, they are often driven instead by the constraints."

His comment got me thinking about the constraint point on the Agile Triangle (Figure 3.1). While the most frequent discussions are about value and quality, we can't forget the role of constraints and the way in which they often drive innovation and design. Teams at Ideo, the highly recognized design firm, are often driven by time constraints. While they have great freedom and a highly collaborative environment, they operate under very tight time constraints—and they usually deliver.

There are two key points here:

- Constraints are critical to innovation and creativity.
- Constraints should be carefully constructed.

The second point here is one that often eludes us. Constraints are often arbitrarily set, with little thought to their impact on product development. For example, as Exploration Factors get higher (greater risk and uncertainty), setting aggressive time constraints may be very useful. However, setting aggressive scope requirements at the same time is usually counterproductive.

In the Agile Triangle, schedule, cost, and scope are identified as the main constraints. How we set these constraints has a significant impact on the development effort. Do we set constraints for one of these elements or all of them? Do we set aggressive targets or looser ones? If we want the team to be innovative and creative, setting too many or too aggressive constraints can undermine our goals. One technique

I've found effective is to create a tradeoff matrix that places scope, schedule, and cost along one axis, and fixed, flexible, and accept characteristics along the other axis. Each constraint can then have one and only one value. For example, "fixed scope, flexible schedule, accept cost" would indicate that scope was the most important constraint with limited leeway, schedule would be somewhat flexible (wider tolerances), and cost would be the least important with greater tolerances (but still within acceptable limits). This tradeoff matrix is negotiated between development and product management, and it gives the entire team a good idea about relative importance of the constraints. It helps "steer" innovation and design.

Another type of scope constraint is one that often occurs when replacing an existing system. The requirements are often given as "duplicate the existing functionality." While this may not be the most useful constraint in some circumstances (much functionality may not be used, so it should really be deleted), it is frequently encountered in these projects. Another version of this scope constraint arises when building a product that has direct competition: the constraint may be to duplicate most of the competitor's functionality before releasing the product.

The bottom line is that constraints provide the delivery team with critical information that helps them be innovative and effective. As such, constraints should be carefully considered because they can guide the team to an effective solution—or when done wrong, to one that is awful. Don't forget to think carefully about this third point on the Agile Triangle.

Determining Business Value

The topic of business value is a complex one, and it's easy to get mired in the morass of calculating ROI or in trying to define which intangibles are relevant to your organization. What we need is a model for looking at business value focusing on the portfolio and project levels. Business value is important for a couple of reasons: it helps us focus on what we think is important to our organization, and it helps us make priority decisions among our myriad of opportunities, projects, and product features.

A definition that resonates with my concept of value comes from Wikipedia: "business value is an informal term that includes all forms of value that determine the health and well-being of the firm in the long run. Business value expands concept of value of the firm beyond economic to include other forms of value such as employee value, customer value, supplier value, channel partner value, alliance partner value, managerial value, and societal value. Many of these forms of value are not directly measured in monetary terms."

Ulrich and Smallwood would concur with this last sentence about value measurements. In *How Leaders Build Value* (Smallwood, 2006), this author states that 85% of a company's market capitalization can be attributed to intangible factors such as leadership, culture, and patents. Investors look at the stream of earnings volatility over time to determine price/earnings ratios (which determines market capitalization), and intangibles drive that stream—we just have to look at an intangible like Steve Jobs's leadership to prove the point.

Given the turbulence and uncertainty of today's business environment, picking the right intangible factors can be a daunting task. Nevertheless, one of the key capabilities required is the ability to discover and capitalize on the opportunities that this turbulence creates. A company's ability to take advantage of opportunities requires a number of intangible factors critical to sustaining a flow of earnings.

To summarize, then, business value has both tangible (financial) and intangible components, intangibles are critical to long-term success, and the ability to capitalize on the flow of opportunities is a critical intangible capability for most companies.

To implement this focus on intangibles and opportunities, I propose using a model with Business Value Points (BVP) and a Business Value Points Matrix (BVPM) (Figure 3.2) that would help prioritize projects (e.g., product development). Agilists have long used story points to measure cost. Story points are relative and nonfinancial (sort of). In the past, I've advocated calculating relative value points (Highsmith, 2010) for capabilities (epics) and features (stories are too low level) to indicate that value is important and to help teams prioritize features. But there is an even better reason to use Business Value Points rather than dollars (or euros or yen): it raises the visibility of intangibles and lowers the visibility of financial measures! This is not to

	Start-up	Scale	Mature
Financial	10	40	75
Opportunity	40	20	5
Social	25	20	10
Trait	25	20	10

Figure 3.2 The Business Value Points Matrix

say that financials are unimportant, but rather that other measures are just as important. If Ulrich and Smallwood are correct, focusing on intangibles in the long term has a major impact on financials. Thus we need a model to avoid over-focusing on short-term financials.

Because capitalizing on opportunities is critical to surviving and thriving in our volatile economy, the columns in the BVPM would be "Start-up," "Scale," "Mature," and "Decline." The rows in the matrix would be "Financial," "Opportunity Capture," "Customer Impact," "Employee Impact," "Social Impact," and "Traits," or some combination of these measures customized to your organization (the categories in Figure 3.2 are abbreviated). The numbers indicate the relative importance of the factors. For example, in a Start-up phase, financial results might be relatively unimportant whereas opportunity capture is very important. Conversely, in the Mature phase, financial results might be by far the most important consideration. Most of these factors are self-explanatory, except for traits. Traits are behaviors and capabilities of employees. For example, in a volatile business environment, it might be important for managers and staff to become more adaptable. A project to implement an agile approach to product development would further the objective of increasing the "adaptability" behavior.

This approach changes portfolio management. For example, Mature-phase projects can't be directly compared to Start-up projects—each needs a "bucket" of dollars that is determined by an executive group. Within each category, projects would be prioritized by assigning them a value from 1 to 100, using the table value in each category as a maximum. For example, in the Start-up category, project 1 might have a business value of 55 (financial = 5, opportunity = 30, social = 10, trait = 10), whereas project 2 has a value of 80 (financial = 10, opportunity = 35, social = 20, trait = 15). With a budget for Start-up projects (an entire book could be written on the problem of having a fixed budget for start-ups), a business value priority list might indicate that the "cut-off" for funding would be 12 projects.

While this type of analysis might not be complex enough for financial types, we have to remember the objectives of assigning relative business values:

- Aiding prioritization of projects
- Systematically utilizing both financial and nonfinancial criteria to demonstrate the importance of value

Looking at Figure 3.2, it becomes obvious that a complex, detailed financial analysis would not be useful in evaluating Start-ups, but it would be helpful in evaluating the Mature category. Also, at a portfolio level, each stage of opportunity capture needs a different portfolio management process. For example, whereas projects in the

Start-up phase need fast, frequent review, the Mature-phase review could progress at a more leisurely pace.

A similar model can be used at the capability or feature level, although the criteria may prove very difficult to assess. Traits, for example, while relevant on a project level, wouldn't help us prioritize features. Allocating BVP at a feature level would, therefore, require some adjustments to the criteria matrix.

This section has merely scratched the surface of determining business value. Even so, it offers a model that addresses key issues in today's world—speed, uncertainty, and new technologies, among others—from an opportunity management and intangible focus perspectives.

Beyond Project Plans

The agile community has long advocated self-organizing teams. However, the emphasis has been on how teams perform work, make technical decisions, and the like. Most teams continue to operate in the same traditional way when it comes to measuring project performance and the application of controls. If empowerment truly focuses on decentralized decisions and authority, maybe it's time to reevaluate how we empower teams from a financial and performance perspective. In too many cases, we are still binding them to fixed plans, as shown by a traditional Gantt chart in Figure 3.3.

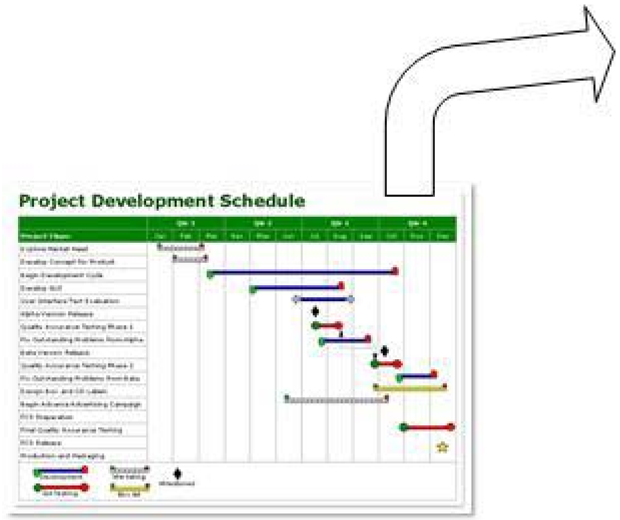

Figure 3.3 Beyond Project Plans

My inspiration for this section comes from meeting Bjarte Bogsnes, Vice President of Performance Management Development at Statoil and author of *Implementing Beyond Budgeting* (Bogsnes, 2008), in Australia at a conference. After talking with Bjarte, I reread his book, which discusses how he helped eliminate budgets in several large companies. One of the insights he gained was thinking about the question, "What do we really use budgets for?" The equivalent question in project terms could be, "What do we use project plans for?" The most obvious answer to that question has three components (you may think of others):

- Coordination with other activities
- Financial and value controls (value is more than money)
- Motivation

The insight that Bjarte and others had was that they were using a single number for multiple purposes and that the single number was causing significant problems. By eliminating budgets, monitoring costs and revenue in new ways, and creating a new set of relative performance guides, companies can break loose from the budgeting straitjacket and improve performance. Maybe there is a parallel for projects.

Traditionally, managers look at three project measures: schedule, cost, and scope. Furthermore, they insist on meeting all three planned measures exactly—a virtual impossibility in today's turbulent business environment. What if we look at three measures for each of these?

- Targets: Desired business outcomes (usually a stretch goal)
- Forecasts: Best current estimate of outcomes
- Constraints: The limits of the team's authority

Let's apply these measures to a project whose traditional cost "budget" is $250,000. We could have a target of $200,000 (might happen if everything goes right), a forecast of $240,000 (our current estimate of the total cost), and a constraint of $275,000 (the team is authorized to spend up to this amount). If the team delivers the project with the capabilities agreed to (value, not scope), with the appropriate quality of results, within the time "constraint," then any cost between $200,000 and $275,000 might be considered acceptable. I say "might" because only a holistic evaluation of the outcomes can determine performance. Give project teams more leeway with results (a lot more leeway), and performance usually improves.

One key determinant of project success is team motivation, and unfortunately most traditional project controls and measures try hard to "demotivate" teams. Consequently, just using the wider limits on the traditional measures cited previously won't be enough. Traditional measures tend to be of the stick kind ("Do this or else"). Time and again, studies, highlighted by Dan Pink's (2009) work, show that the best motivators are intrinsic—purpose, mastery, and autonomy—and focus on positives rather than negatives. Better motivators for projects are purpose-driven outcomes such as customer value delivery and quality. Better motivators are relative comparison measures, not absolute numbers. Better motivators are those that allow the team autonomy, and that includes leeway on things such as cost, scope, and schedule. Give teams vision, facilitate their self-organization, and provide constraints (loose boundaries), and you encourage them to be creative and innovative in their solutions.

Giving people stretch goals may motivate them, but when those goals are linked to potential punitive actions if not met, their motivational value is lost. Motivation needs to focus on vision and purpose, not penalizing people. When people think they will be judged against plans, they are likely to fudge the plans, or even sometimes the actual performance. Instead, break plans into two numbers, each with a specific purpose: a target based on business needs that represents a stretch goal, and forecasts (updated regularly) that are the best estimates of outcomes.

But, someone always says, what about cost control? (It always seems to be about cost control, not value delivery.) The answer for this is (1) to track actual costs carefully and watch trends, and (2) to establish an authority constraint (limit) for each project. This constraint should be generous, not onerous. Projects that are forecasted to exceed their constraints would need further review and possibly additional funding (or termination). The big difference here is the difference between a predicted cost and a cost constraint.

Project teams also need to coordinate with others (e.g., teams, projects, departments), usually about schedules. Forecasts are the numbers used for coordination. If targets are really stretch goals—with, say, a 50/50 chance of achieving them—then project teams shouldn't base their plans on these targets, but rather on forecasts, meaning our best estimates of outcomes. By using these two measures in tandem, both coordination and motivation are improved.

For this evaluation system to work, two things have to happen. First, everyone, including both managers and team members, has to understand the rationale behind each type of number. Second, everyone needs to evaluate the results holistically. Every number has a purpose, but performance is evaluated by taking all the numbers into account in a holistic manner. No single number, or even a series of numbers, is adequate to completely evaluate a complex undertaking such as a project. Performance evaluation should focus primarily on dimensions such as value and quality, and secondarily on constraints such as cost or scope.

Any metrics system can be gamed. The success or failure of any metrics system lies in the intent of the managers and leaders applying the system. As Rob Austin, dean of the business school at the University of New Brunswick, says, "If there is a single message that comes from this book, it is that trust, honesty, and good intentions are more efficient in many social contexts than verification, guile, and self-interest" (Austin, 1996).

FEATURE FOLLY

In my executive presentations, I spend a fair amount of time on the topic of "Do Less," talking about how we waste an incredible amount of time and money building features that are rarely or never used. Published studies put this number at far greater than 50%. I can see audience members agreeing with my message, but I can also see the little thought bubble floating slightly above their heads: "but not in MY organization." Organizations worry about improving development productivity by 10% when what they should be worried about is improving customer demand effectiveness by 25%. I've always said that one of the biggest potential productivity improvements from the agile approach lies in all the features that we don't do because of the constant attention to simplicity and delivering the highest value to the customer.

In most organizations, software development efficiencies are subjected to infinite analysis, while customer demand effectiveness isn't mentioned at all. We neither measure nor calculate feature value in the beginning, nor do we measure whether value was actually captured as the customer or product manager had predicted. In many large companies, project ROI might be calculated as part of the portfolio management process, but rarely does anyone follow up to see if that ROI was, in fact, attained. Consequently, development teams have multiple feedback mechanisms, whereas customer/product teams have relatively few, except possibly at a macro level (product sales, for example). Product managers are left with gut feel as the basis for most of their feature-level decisions. No wonder 50% or more of developed features are rarely or never used. Product management teams aren't the culprit here, but rather the lack of adequate feedback mechanisms.

If we look at a typical agile project team, we will likely see a development subteam and a product/customer subteam. Roles, responsibilities, and feedback mechanisms have been defined. For example, the product team identifies stories, writes stories, prioritizes them, and develops acceptance criteria (both automated acceptance tests and feature showcase evaluations). In turn, there are micro-level (feature and story) feedback mechanisms to steer the development effort. But what about feedback from product management to the business? These links are often much more tenuous, such as measures of overall sales that tell how a product is doing at a macro level, but say nothing about individual features. Internal IT products generate even less feedback in most cases. Lack of feedback leads to feature bloat, because it's always easy to succumb to customer requests and internal demands to improve the product.

At the Agile Brazil 2011 conference, I was listening to Josh Keriesvky's talk on his Lean Start-up experience, and gravitated to thinking about this problem. Here's a starter list of ideas about potential solutions:

- Develop customer demand effectiveness measures for every product management organization and team.
- Calculate relative or monetary value for every feature.
- Use relative benefits such as increasing customer happiness, reducing customer risk, and improving the internal collaboration culture to evaluate feature value.
- Build feature usage information into software to provide feedback to product management. Product manages could thereby gain insight into what was actually used and what wasn't.
- Develop feature evaluation experiments into the feature identification and prioritization process. For example, do A/B testing on features.
- Do false feature analysis. Give users access to a feature that, when selected, pops up a message such as "This feature is under development. When it is completed, are you 'very likely,' 'somewhat likely,' or 'not likely' to use it?" Josh talked about an expensive feature that his company decided not to implement because of the results from this type of test.
- Use short, focused surveys to measure customer happiness. For example, ask your customers, "What was your experience using capability Y (this capability being several features): Awesome, OK, Not So Hot."

From a Lean perspective, the biggest "waste" factor in software development is low- or no-value features. If we were able to eliminate 15%, 20%, or 30% of the scope of software projects, we would increase the chances of meeting planned delivery dates. Agile's biggest contribution to productivity, then, might be all the functionality we don't produce.

Of course, agility is not just about cutting out stuff, but about emphasizing the right thing—and that thing is value. Product managers should be calculating business value, either relative or monetary, to the feature level so that teams can produce the highest-value feature early. Then, if scope reductions need to happen later in a project, the eliminated features come from the lower-value list.

While the agile and Lean communities have often alluded to value-based development, the actual practices to support this objective are not yet sufficient. There aren't sufficient feedback mechanisms at the feature level to help mitigate the constant push toward feature bloat. Maybe taking a look at some of the ideas from Lean Start-up and other sources can help.

Features or Quality? Selling Software Excellence to Business Partners

As Figure 3.4 asks, should we deliver features or quality? It's always been difficult getting business partners (from executives to product owners) interested in quality—be that code quality, design quality, automated testing, or technical debt. Software technical excellence numbers (ah, if we just had good numbers) don't mean much to business partners.

Recently I've been adding to the Agile Triangle (value, quality, constraints) the idea that while business partners are only mildly interested in quality or software excellence (such concepts are too esoteric), they *are* interested in cycle time (getting stuff out faster). Furthermore, I hypothesize that cycle time is a function of quality (among other factors). Thus we need to "sell" software excellence on two key bases—valued delivered and cycle time. If cycle time is, in fact, a function of quality, then we should be touting cycle time (the *what*) improvements and leaving code quality, design quality, and technical debt discussions (the *how*) mostly to the engineers.

From a business partner's perspective, the question "Features or quality?" is easy to answer—more features. The partner is being asked to trade off a business outcome (features) for a technical outcome (quality)—something that is easy for the partner to understand versus something that is difficult. It's not a hard choice, as you might imagine. However, the question "Features or cycle time?" isn't so easy to address—because it requires trading off two business outcomes. If we can show the relationship between cycle time and quality and then start measuring and reporting cycle time, perhaps we can give our business partners a better and more realistic way of assessing software delivery performance.

When I was presenting this hypothesis to a group recently, one of my colleagues offered this challenge: "How can cycle time be a function of quality when everyone knows that quality can be traded off for additional functionality—people do it all the time." This challenge stuck with me for several months without a good answer until I

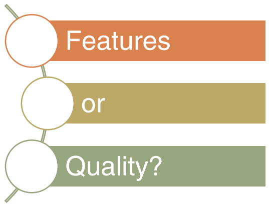

Figure 3.4 Features or Quality

heard Martin Fowler's talk on technical debt. I was mulling all of these issues over on a bike ride when the solution occurred to me.

When people trade off more features for less quality, it's usually for a single release occurrence, not for an aggregation of releases over time. This is an outgrowth of waterfall development, in which intervals between releases were long, often a year or more, and trading new features for lower quality (say, poor design or less testing) obscured the cost and pushed consequences far into the future. When we have a large batch size (hundreds of features) and a long time frame (a year or more), the next releases (small maintenance or enhancements) are so trivial in relation to the first release that the feedback or impact of low quality is very difficult to determine. In a waterfall project, it is easier to cut refactoring, for example, because the impact is felt in the future, and even then only the engineers feel the pain. Cycle time measures are irrelevant when release cycles are too long.

However, as agile teams reduce delivery cycles to months, weeks, and days, the impact of poor quality becomes much easier to determine. When a team is running 1-week deployment cycles, the effects of poor testing in one cycle may pop up quickly, in the next cycle or two. Poor design in one cycle begins to retard feature delivery in the next few cycles—the consequential feedback comes in a few weeks. If a team is measuring both feature throughput and cycle time, either or both can suffer quickly from software mediocrity. However, even in our agile era, not enough teams systematically measure cycle time (other than those who are doing Kanban), so the relationship between quality and cycle time remains murky for many.

If the delivery teams are keeping reasonable metrics, the quality/cycle time relationship becomes clear quickly. What also becomes clear is that the old assumption about features and quality are wrong—that technical excellence can increase throughput *and* reduce cycle time. Waterfall projects cover up this understanding.

Finally, just a brief note to admit that cycle time measures can be thorny. There are three types of cycle time, all important. The first is feature delivery time (from inception to release). The second is release frequency (how often we release and/or deploy the product). Finding the starting and ending points for feature cycle time can be tricky. But these difficulties can be overcome as we learn better ways of measuring. Once these cycles begin decreasing from months to weeks and days, the impact of technical excellence becomes much clearer. The third type of cycle time is product delivery time—from inception to final (or release) delivery. Studies have shown, for example, that business customers of IT have about a 6-month window that they consider "good" performance. Correspondingly, projects with greater than 6-month delivery cycles are usually in trouble from the start—regardless of other factors.

Let's help our business partners move from thinking about "business" features versus "technical" quality to a more productive view of "business" features versus "business" cycle time.

ALL PROJECTS ARE NOT THE SAME

One of the big problems with successfully executing projects is that while we know projects are very different from each other, we often manage them and measure their success in the same way. Think for a moment about the oft-quoted Standish reports in which project success is measured on the traditional Iron Triangle basis of meeting scope, schedule, and cost plans. In the Standish scheme, all projects, of any type, are successful or not based on the same criteria.

A friend of mine worked on a project recently where the client said, "I have a fuzzy vision of what we want. I don't have any idea what the detailed requirements should be. I need results fast." The client even refused to participate in a story identification process, leaving experimentation up to the development team. Managing this project against a backdrop consisting of traditional measures of scope, schedule, and cost would eliminate any chance of success; it is a different kind of project. The team evolved the product from a vision and evolutionary learning and the client was very pleased with the results, because he understood it was a different type of project.

In *Agile Project Management* (Highsmith, 2010), I wrote about assigning an Exploration Factor (EF) to each project (or release of a product). The EF attempts to identify a level of uncertainty and risk for a project by looking at both technology and requirements. Technology can run the gamut from "well known" to "bleeding edge," and requirements can range from "stable" to "erratic." Combining the two factors yields EFs (see Figure 3.5) ranging from 1 (very low uncertainty) to 10 (very high

	Product Technology Dimension			
Product Requirements Dimension	Bleeding Edge	Leading Edge	Familiar	Well Known
Erratic	10	8	7	7
Fluctuating	8	7	6	5
Routine	7	6	4	3
Stable	7	5	3	1

Figure 3.5 Exploration Factors

uncertainty and risk). EF 1 and EF 10 projects are very different, so they need to be managed differently and assessed differently. For example, if the requirements are uncertain, measuring progress against a scope plan is ridiculous. This doesn't mean the project is uncontrollable, just that we have to control it differently.

Before we think about how to measure success on projects with high EFs, we need to understand what makes these projects successful. The pressures on high-EF projects are usually uncertainty (about either requirements or technology, or both) and speed. Typically high-EF projects are also strategic, customer facing, Web or mobile, or some other flavor that tend to increase the uncertainty. Sponsors want them done quickly. How are projects like this managed successfully? By using a combination of a well-articulated vision, quick iterations, functionality evolved to a minimal viable product and beyond, adaptability as a result of learning from customers as the product evolves, focus on value delivery, and time boxing.

One important aspect of controlling high-EF projects is to view scope, schedule, and cost as constraints, not objectives. Because the project requirements are not known, it is often difficult to estimate time. Given the lack of specificity, however, the team needs constraints—for example, a time box of 3 months that limits exposure. At the end of this time frame, the project sponsor can evaluate the value delivered and the questions answered, and then decide whether to invest in the next increment. The question to be asked at every iteration and release is, "What is keeping us from deploying this product now?" This is very different from the normal progress question of "Have we developed all the planned scope yet?"

Conversely, a very low-EF project (say, a 1 or 2) could be managed with scope questions because the requirements are knowable in the beginning (or at least people think they are, even though they are often wrong). Note that software development projects are rarely low-EF projects.

The bottom line is this:

- All projects are not the same.
- Projects with different Exploration Factors should be managed differently.
- Performance measures should be different for high-EF projects than for low-EF ones.

SCOPE ISSUES IN AN AGILE PROJECT

I was talking with a colleague the other day about troubles with scope management in an agile project. She was lamenting problems that were arising with a particular client who was concerned about the progress of the delivery team. Because agile teams use time-boxed iterations and then let scope adjust to those iterations, this shouldn't be a problem—should it?

A number of issues surround scope management in an agile project, many of which are the same as the issues that arise when trying to manage scope in a traditional project:

- Perception versus reality
- Poor definition of how to measure scope
- Running an agile project in a nonagile organization
- Wish-based planning

Agile projects, unfortunately, don't eliminate the perception-versus-reality problem. Miscommunication and misunderstanding can impact an agile project even as teams try to expose reality—or at least their version of reality. Short iterations and working software can reduce the perception/reality gap, but as long as projects are delivered by people and evaluated by other people, a gap will often remain. Good project managers realize they have to manage both—reality and perception.

Another question that isn't asked or answered very often is, "What is scope?" Is it the number of requirements, stories, or features? Is it the total work hours or story points estimated for the project? Is it the documented requirements? At which point is scope determined, given that in an agile project features can change over the life of the project? These are all bottom-up measures of scope (and therefore progress). Maybe a better approach, especially in an agile project in which the detail stories and features are changing, is to ask a top-down question, "Can we deploy this product at the end of this iteration?" Obviously, the answer to this question involves some determination about feature completion, but the question really asks about value, about enough value to deploy, not about whether a set of detailed requirements has been met.

Scope issues often crop up when agile teams confront traditional organizational success measurements. The teams may view themselves as being successful on the project even as managers are wondering what's going on because they don't understand this "iterative" approach. This is somewhat different from the perception-versus-reality issue; it's more the clash of two different perceptions (or two realities).

Finally, too many organizations subscribe to what I've called "wish-based planning." They do a lousy job of capacity planning—that is, balancing the demands for work to be done with the actual capacity of the organization. These managers don't understand the difference between stretching limits and being completely unreasonable. All too often, then, stretched project plans become irrational wish-based plans. Agile teams will still experience scope problems in this type of dysfunctional situation.

Adopting an agile approach won't fix all your scope problems. Agile development can help teams and management look at scope from a different perspective, but the long-held perceptions of scope will be difficult to change in many organizations.

Speed

Both speed and value are important. Delivering value early and often (every iteration or set of iterations) can improve ROI substantially over delivering value at the end of a 12- to 18-month serial project. Nevertheless, speed needs to incorporate not just engineering, meaning creating features that are ready for deployment, but also the needs of the business organizations that actually deploy and use the features. Speed might measure the time between order input and order fulfillment, or the time between release of a feature into development and its deployment. The former is a measure of business results speed; the latter is a measure of software features that support this business process.

Speed measurements in Kanban projects—time from release off a backlog into development until the feature is complete (tested, accepted)—are being used in service-level agreements. Because of the strict work-in-process limits, agreements such as "We agree to deliver new features within 21 days with a 95% confidence limit" can be made.

Speed is also about perception, and the elapsed time of a project can be more about perception than reality. When people declare, "The project is late," they may actually mean the project is taking too long, irrespective of the planned schedule. The potential for such a negative perspective to emerge grows as projects lengthen. For example, even though a project is planned for 2 years and is on schedule, the perception of its progress is often negative just because of the overall length of time (of course, 2-year plans are almost never accurate). By comparison, a project that delivers results in 3 to 6 months will usually be well perceived—even if it is a month "over budget." To some extent, regardless of plans, results in a short period are considered successful while projects that roll out over longer periods are considered not successful. Reducing project time frames can, by itself, improve the perception of success—at least in terms of delivering greater speed.

Reducing Cycle Time

An increasing number of organizations are moving toward radical reductions in cycle time as they move toward rapid business responsiveness and continuous delivery.

One mantra that seems to help teams and organizations in this quest is, "If it's hard to do, do it more often!" Keep this mantra in mind. If something appears too hard, too costly, or too slow, figure out a way to do it more often. I once worked with a company whose product took 6 months of quality assurance (QA) prior to release. The QA manager couldn't imagine how to reduce the time to 2-week iterations, so I asked him if he could figure out how to do it every 2 months. After several subsequent iterations, his group was able to support 2-week iterations. From Lean manufacturing examples, we often see that 80% or more of the time taken to accomplish a process usually works out to be wait time, not work time, so pushing for significant reductions is often much easier than anticipated at first.

In trying to answer the question, "How can we do something frequently?" the answer may come from a combination of simplification, elimination of constraints, and automation. Every time we push to do something more often, whether it is a software build and integration or a design, we learn. Learning comes from repetition.

From Goldratt's (1984) theory of constraints, we have learned to look for process bottlenecks. The bottleneck could be the lack of a particular skill or the throughput of a machine or a computer, but the key aspect of a bottleneck is that its elimination can create significant improvements in overall throughput and cycle time reduction. Conversely, if we don't think about bottlenecks, we can add significant resources without impacting throughput at all.

Teams should always ask the question, "How could we simplify this process or activity?" In some cases, this question may be more like "What can we eliminate or streamline to reduce the time to do this from 6 weeks to 1 week?" Again, the time to accomplish some overall process can often be traced to delays between groups and excessive control processes or steps. For example, if a sequential process takes 8 weeks and involves four groups, each group may have a logging, prioritization, and reviewing process for work items. Reducing the process to 1 week by eliminating communication delays may also eliminate the need for these control processes.

Because we are in the business of IT, automation always comes to mind as an enabler to doing things more often. Despite its allure, we shouldn't jump to implement automation until some of these other ideas have been tried.

The mantra "If it's hard to do, do it more often!" espouses the agile value of responding to change. In today's high-change world, responsiveness to change is tied to the cost of change—reducing its cost increases our responsiveness. The high cost of some changes should not be viewed as a barrier to responsiveness, but rather as an opportunity to increase our responsiveness by overcoming that barrier.

Cycles, Cycles, Cycles

One of the problems in integrating agile delivery or continuous delivery into enterprises is the differences in cycles. In the past, companies have tended to run on annual budgeting cycles and even longer strategic planning cycles, with some (though not always close) coordination between the two. Product management tends to run on product cycles, which, depending on the product type, can last for months to years (for airplanes, for example). Projects tend to run in phases (traditional projects) or releases and iterations (agile). Project management offices, reflecting upper management's desires, operate on monthly, quarterly, and annual cycles.

There are several potential problems with all these cycles:

- The cycles don't coordinate well.
- Everyone wants everyone else to conform to their cycles.
- Finance considerations, particularly for public companies, seem to drive everyone else's cycles.
- Different kinds of projects and business initiatives need different cycles.

Project cycles have always clashed with financial cycles, as projects just don't naturally finish in December. The cost side of projects has traditionally been reported on a calendar basis, but the value side has often not started until the project has completed. Incompatible cycles have plagued developers and accountants alike—for example, biweekly payrolls don't match up nicely with monthly financial reporting.

Most cycle mismatches are time related, but other types of discrepancies can occur. For example, agile cycles deliver partial results in 2-week iterations. By "partial results," I mean working software, but only part of the application; architectural pieces, but not an entire architecture; or requirements, but only details for the stories done during the iteration. However, many company governance cycles are built around completed results—a complete requirements document, a complete test plan, or a complete database design. Governance systems based on this model make agile projects difficult to "govern" in these organizations because of this cycle mismatch.

In my prior life as a waterfall methodologist (yes, I admit it) in the 1980s, I used something called the Warnier-Orr methodology, which included a bracket-style diagram called the Warnier-Orr diagram. Part of the methodology for resolving a cycle conflict was to determine the "lowest common denominator." For example, in a payroll system that needed to generate biweekly paychecks and monthly financial statements, the lowest common denominator would be days. While this is a simple example, it's amazing how many systems kept data at the wrong level of detail and couldn't generate all the required information for different cycles.

Many agile organizations are using the three-tier model for development shown in Figure 3.6: Iteration (2 weeks), Release (3–6 months), and VisionMap (RoadMap) (6–18 months), with corresponding differences in the granularity of the deliverables

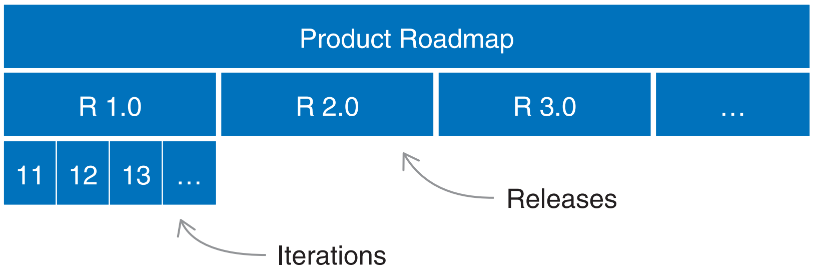

Figure 3.6 Cycles with Different Time Horizons

(stories, features, capabilities). What if the entire enterprise used this "short-horizon" model to actually run the business? What if everyone synchronized their efforts based on 2-week iterations? What if everyone focused on 2-week delivery of partially completed products, documents, business initiatives, plans, and other business artifacts? I was talking with a client recently who was having cycle meshing problems—the company's product management group wasn't syncing up very well with its agile delivery teams. In other companies, DevOps initiatives are attempting to match operations and release management more closely to delivery team releases.

When companies get serious about enterprise agility, one area that will require major change is moving from financial cycles being the driver to focus on a product-driven (deliverables) "short-horizon" model (which delivers the required financial information but is not driven by those financial cycles) that is more adaptable to changing conditions.

Shortening the Tail

In *Agile Project Management* (Highsmith, 2010), I wrote a short section on a performance metric called "shortening the tail." I liked using the metric of "tail length" because it is easy to calculate and tells a lot about an organization's agile implementation. It's not a vanity metric, like the number of developers who have attended a refactoring seminar, but a true learning metric because it focuses on a key tenet of agile development—running, tested software. It's also a metric that can help an organization move closer to continuous delivery.

The tail is the time period from "code slush" (true code freezes are rare) or "feature freeze" to actual deployment. This is the time period when companies do some or all of the following: beta testing, regression testing, product integration, integration testing, documentation, defect fixing. The worst "tail" I've encountered was 18 months—18 months from feature freeze to product release, and most of that time was spent in QA. I've routinely encountered software companies whose tail is 4 to 6 months of a 12-month release cycle. Other companies, including a growing number of software companies, have honed their processes to have a zero tail length—they are truly doing continuous delivery and continuous deployment. Using the tail length metric, particularly in products or applications that have large legacy code bases, can help organizations monitor their progress toward continuous delivery.

I worked with an organization several years ago that had a 6-month tail in a 12-month release cycle—and the tail was getting progressively worse with every release. The tail had expanded from 4 to 6 months over the past three release cycles. The company set a goal to achieve a 1-month tail and worked diligently to achieve that goal over time.

Shortening the tail is a simple, powerful metric for measuring progress toward agility. The goal of agile teams is to produce shippable software every iteration, but most are far from this goal—especially if they have large, old legacy code bases. Think of everything a company might have to do to reduce a tail from 6 to 3 months, then to 1 month, then to 1 day. The company would have to learn how to do continuous integration across its entire product. It would have to improve its level of automated testing to drive regression and integration testing back into every iteration. It would have to improve the level of automated unit testing done by developers to reduce testing time at the end of iterations and releases. It would have to bring customers into the development process much earlier, rather than waiting until the end for beta testing. It would have to integrate documentation specialists into the team and produce documentation continuously during iterations. It would have to invest in systematic refactoring to reduce the technical debt, and thereby reduce testing and defect fixing time.

You can probably think of more the company would have to do.

Each of these items would contribute in some way, large or small, to reducing the tail by days or weeks, as Figure 3.7 shows. For large products, the tail might never reach zero, but it could be small. Just think of the competitive disadvantage a company has when its delivery tail is 18 months, or even 6 months. Such a lengthy tail means that for 6 or 18 months prior to release, no changes in the competitive environment could be incorporated into the company's products.

If continuous delivery seems too big a step, start on that path by first reducing the tail length on your product releases. Before long, continuous delivery won't seem that big a stretch.

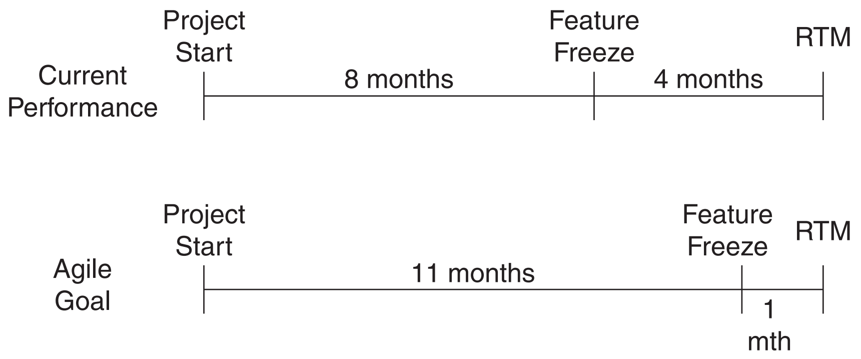

Figure 3.7 Shortening the Tail

QUALITY MANAGEMENT

The more I visit companies and see mangled agile implementations, the more I become convinced that quality, or lack thereof, remains the central issue in achieving effective agility. Organizations begin agile implementations with higher quality as the goal, but then all too frequently they do not carry through with the discipline to achieve the goals they have established. They may have goals and desires, but not enough engagement and commitment. Admittedly, a critical problem is often the relentless pressure on delivery, but managers must step up and begin to transition from the vicious cycle, depicted in Figure 3.8, of "incur technical debt, slow delivery, increase pressure, fail to repay debt, incur more technical debt" to a virtuous cycle of "build high quality, speed delivery, decrease pressure, and repay debt."

Some might ask if management really has an impact on quality. After all, the perception is that quality is up to developers and testers. This next story illustrates that management can, indeed, have a dramatic impact. A large project team that was regularly measuring code toxicity (a combination of several measures) noticed a spike in that toxicity. Tracing back to the time the increased toxicity started, the team members determined that the cause was a change in managers.

When managers talk about quality being a priority, but then fail to allocate money to acquire adequate testing expertise and tools, or fail to emphasize quality with their teams, or fail to allocate time to create and maintain test suites, their lack of real commitment to quality begins to show. When you are caught in the bowels of a vicious cycle, turning that situation around is a management issue. Of course, the technical teams must embrace the requisite practices and discipline, but without managers and executives who are engaged in seeing that quality is critical to the turnaround, teams will have a very difficult time delivering quality products. The strategy needs

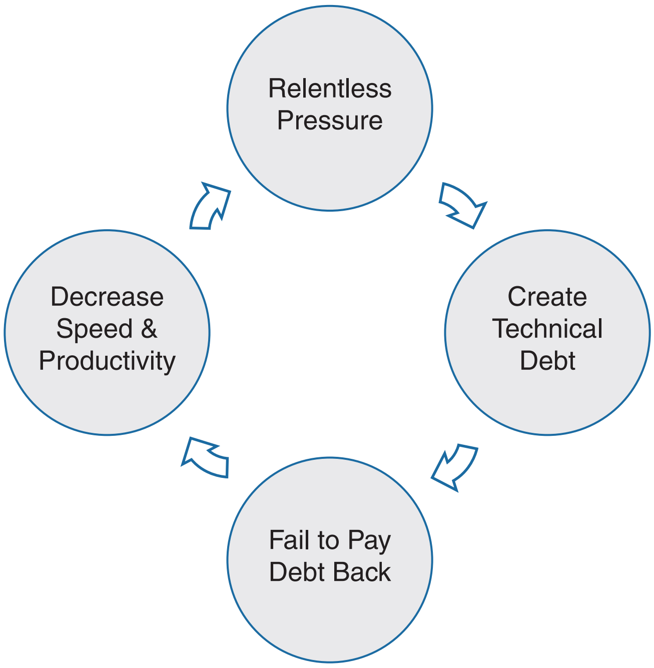

Figure 3.8 Technical Debt's Vicious Cycle

to move from "more features, more features" to "fewer features, higher quality, then more features."

But what does engagement in quality mean? Probably the most difficult task is for managers to really commit to the short-term pain required to deliver long-term gain. The gain may take only months to achieve, but there is always the pain of short-term performance loss (and investment) while people learn new practices. Unfortunately, this pain often leads to lack of full commitment, where managers fail to push their teams to full agile implementation—they get the pain, without the gain. This lack of commitment comes from lack of real understanding of how quality impacts speed, how technical practices fit together (e.g., refactoring, test first, and simple design), and which investment tradeoffs are appropriate. For example, many managers succumb to the pressure to deliver features over quality because they think quality shortcuts are detrimental in the long term, while feature delivery is a short-term issue (short-term gain for long-term pain). From our experience with effective agile teams, we now recognize that inattention to quality begins to degrade delivery velocity in only a few iterations. The road to fast, productive software development goes through quality—a lesson highlighted again and again by metrics gurus such as Capers Jones and Michael Mah, but one that is still not embraced by many practitioners.

Managers are often caught in a perceived dilemma between perfection and "nice to have." On one side, they are often skeptical of what they perceive as the technical team's desire for perfection. They can't discern the difference between perfection and excellence, so they fail to support adequate quality measures. They know whether a feature gets delivered to the customer, but they don't really know how to assess its quality. On the other side, they need to understand the difference between the two aspects of quality—reliability and adaptability—and how to achieve both.

Quality software requires engagement and execution. Execution is the realm of the technical team; engagement is the management side. Managers must do more than say, "Quality is job 1," every 2 months. They must understand what the right quality framework is; they must appreciate the consequences of poor software for customers; they must find the appropriate balance between features and adaptability; they must recognize the impact of technical debt; they must invest in training, tools, and time; and they must have the commitment and discipline to deliver quality products in the face of feature pressure.

THE FINANCIAL IMPLICATIONS OF TECHNICAL DEBT

Technical debt may be costing you more than you imagined!

There has been a great deal of discussion in the agile community about technical debt. The technical debt curve in Figure 3.9 shows how technical debt increases the cost of change over time, until software becomes almost unmaintainable. Two

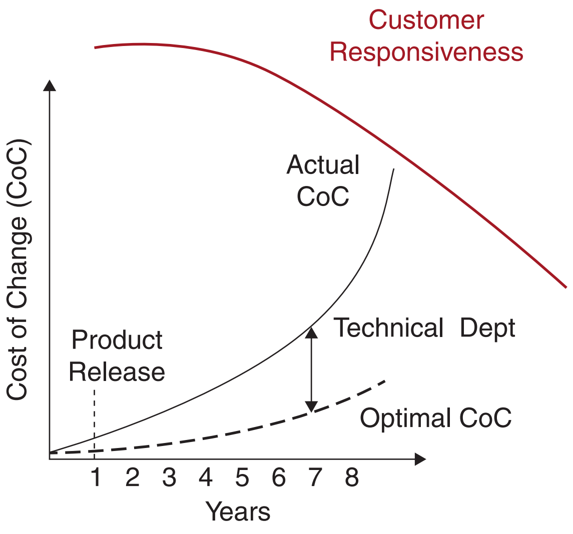

Figure 3.9 The Technical Debt Curve

time scales can be shown on this curve: a longer-term scale that shows how software degrades over time and severely impacts customer responsiveness, and a shorter-term (by iteration) scale that shows how technical debt can seriously impact development speed very quickly.

One problem with technical debt is that its impact can be slow growing and somewhat hidden. We know how the question "Fix the technical debt, or build new features?" is usually answered. As the debt gets worse, customers complain about slow delivery, increasing the pressure to take more shortcuts, which in turn increases the technical debt, which then slows the delivery process, which in turn increases customer dissatisfaction, and so on, in a rapidly downward spiraling vicious cycle. Unfortunately, by the time many organizations start paying attention, all the solutions are bad ones: (1) do nothing and the problem gets worse; (2) replace/rewrite the software (expensive, high risk, doesn't address the root cause problem); or (3) systematically invest in incremental improvement.

A number of people have been working to identify the financial cost of technical debt by examining existing software and calculating the cost of fixing bad code. Studies have indicated this overall "hidden" cost of technical debt is in the $1 trillion range in the United States. Unfortunately, this is only the tip of the iceberg when looking at the total financial impact.

Looking at the Agile Triangle (see Figure 3.1), we can envision that project value numbers (net present value [NPV] or some such measure) eventually work their way into corporate earnings. In software companies, this would be a direct correlation because the value is contained in products themselves; in internal IT projects, the correlation is a little fuzzier, but nonetheless remains valid. Quality has two components: reliability (i.e., the ability of the software to operate as billed) and adaptability (i.e., the ability to respond quickly to future enhancements). Using these definitions, technical debt can impact current value (earnings) if it doesn't perform as it should, and it impacts future earnings by slowing down the delivery process.

Unfortunately, many companies are driven by Wall Street's emphasis on current earnings as a measure of company value. When this attitude permeates the company, managers opt for delivering current features over reducing technical debt. The financial impact of technical debt then goes beyond the cost of fixing the debt, having a big impact on the ROI of any project because it delays benefits and costs more to implement.

If those are future impacts, how do we get financial managers, product managers, and others to focus on the present when it comes to reducing technical debt? The answer is seemingly simple: market capitalization.

In the fourth quarter of 2000, a well-known technology company missed its Wall Street earnings prediction by a few cents per share and proceeded to lose more than $20 billion in market capitalization in the next few days. The biggest problem with technical debt is not its impact on value or earnings, but rather its impact on predictability. Two companies with equal average earnings growth, but different earnings volatility, can have very different market capitalization driven by the volatility.

Everyone has horror stories of software applications that are virtually unmaintainable when changes cause erratic behavior. These applications are terribly difficult to test and are always buggy in production. Technical teams try to "plan" upgrades to these applications, but inevitably they miss projections, often by a lot. When a company is depending on new features to launch new product versions, delays affect earnings volatility. Getting back to the technical debt curve, as technical debt increases, development speed decreases, customer satisfaction decreases, and predictability of results decreases.

The bottom line for technical debt: it's expensive to fix, but much more expensive to ignore. Technical debt reduces future earnings, but even more critically, it destroys predictability which in turn impacts market capitalization in the near term, not in the future.

DO LESS

Many managers use the mantra, "Do more with less." At the Agile 2010 conference, Pat Reed, Senior Director from the Gap Inc., shortened this mantra to "Do less." Her theme of value optimization, and eliminating marginal value work, included creating a culture of value, determining value calculations at the portfolio level, allocating value to software features, and determining the highest-value chunks of functionality to implement next—whether those chunks were projects in a portfolio or stories in an iteration planning session. By developing in an agile fashion and deploying features frequently (continuously), value can be recognized by the business early and often.

> Everyone tries to do too much: solve too many problems, build products with too many features. We say 'no' to almost everything. If you include every decent idea that comes along, you'll just wind up with a half-assed version of your product. What you want to do is build half a product that kicks ass. (The founders of 37signals (Taylor, 2011))

In an agile project, the team always tries to work on the highest-value story. But what if the highest-value story is from the next project in the development queue? Toward the end of a project, or even earlier, the highest-value story or feature may be found in the next project, which means it may be time to stop the current project and move on.

Three studies conducted by The Standish Group (Jim Johnson, CEO, The Standish Group International, XP2002 conference) and the Department of Defense (*Crosstalk Journal*, 2002), and reported by IEEE (IEEE conference, 2002), in the early part of this decade indicate that far more than 50% of functionality in software is rarely or never used. These aren't just marginally valued features; many are no-value features. Think of the cost of these features. Think of the benefits from doing less, from eliminating these features. A CIO friend of mine once delivered a customer relationship management (CRM) application with 25% of the originally requested functionality—and the customer was delighted. In fact, the customer cut off development! The other 75% of features proved to be "nice to have" but not significant contributors to business value. Delighting customers has both a content aspect and a timing dimension. Fifty percent of the features delivered in 6 months may be far more "delighting" than 100% delivered in 18 months. Doing less should operate at all levels. The practice of allocating value to features seeks to "Do the highest-value chunk of work," but also to "Do less"—that is, to eliminate marginal valued features and cut functionality on those features with lower value.

"Do less" has other implications. Reducing work-in-process, for example, increases throughput by cutting down on time-wasting multitasking. Value stream mapping show us where to cut out non-value-adding activities. In looking at value capture, agile managers need to examine cumulative value delivered versus cumulative cost incurred on a project. Then questions can be posed, such as "Do we want 100% of the planned value for 100% of the planned cost, or would we prefer stopping at 90% of the value for 70% of the cost?" Because agile development delivers the highest-value features early, this type of management tradeoff becomes reasonable, even imperative, to think about. Furthermore, the reason this question becomes so important is the issue raised earlier—other projects with higher value need to start sooner. Developing the last 10–20% of marginal functionality on one project delays capturing the higher value on the next project. Clearly, it's not just development cost, but also opportunity cost that managers have to evaluate.

Do less: cut out or cut down projects, cut out overhead that doesn't deliver customer value, cut out or cut down features during release planning, cut out or cut down stories during iteration planning, cut down work-in-process to improve throughput. At the same time, focus on delighting the customer by frequent delivery of value. In an agile organization the mantra should be "Do less," and maybe use the time and money saved to reduce technical debt, launch new innovations, and undertake improvement initiatives.

The "To Do Less" List

"Doing less" is a deceptive term, one that gets you thinking. While you might be put off by the phrase because the mantra in most companies is to "Accomplish more," doing less actually means delivering more value, more throughput, more ROI, and more creativity, and being more focused—but doing less low-value activity.

This message is, in effect, saying, "Think more; do less." It's easy to think about what to put into a product. We often get suggestions from a wide variety of sources—engineers, customers, managers, executives, the janitor, and other teams. Product backlogs can grow exponentially. It's easier to say "yes" to various stakeholders than to say "no." One problem consistently experienced by engineering organizations is too much work-in-process (WIP). Projects are undertaken because the prioritization scheme breaks down (if there is one). It's easier to tell the vice president of manufacturing that "We're working on your project" than "We can start on your project in 2 months when we have adequate capacity." As a result, projects stack up in WIP and throughput drops, sometimes drastically, because of thrashing and multitasking.

Each of us has a handy "To Do" list, but maybe what we really need is a "To Do Less" list. There would, of course, be some process to create these lists. For example, one rule might be that for every entry on the "To Do" list, we have to make one entry on the "To Do Less" list. A product manager might create a backlog of features to do, and another backlog of features not to do. Having an explicit list of "not to do" features sends a message to everyone that this list is important. Deciding not to implement a feature isn't enough—such features have a tendency to creep back on the active pile—so documenting that a feature has been placed on the "Don't Do" list is important.

Similarly, you might keep a list of all the meetings you decided not to attend (this one isn't as difficult). As you start keeping "meetings to attend" and "meetings not to attend" lists, you begin to think about and refine the criteria for each. Actually, you will begin developing these criteria for each of your dueling lists.

You could even think about celebrating your "Not Done" lists at retrospective activities. Here are all the features we did; here are all the ones we didn't do. Here are all the meetings I didn't attend this iteration. Here are all the projects we didn't start this month. Here are all the overhead items we pushed aside last quarter. Obviously, we also celebrate what we accomplished, and we highlight that those accomplishments were sharply focused on value, on real customer solutions, and on creative ideas. By lining up both lists, we begin to see how our focus on effectiveness rather than raw productivity actually improves overall performance.

Build Less, Start Sooner

Jeff Patton reminded me of two simple strategies for software development that I've talked about from time to time—build less software and start sooner. In this section, I'll revisit these simple, but powerful strategies.

First, managers and executives complain a lot about not delivering software (or any other product, really) in a timely manner. In Preston Smith and Don Reinertsen's ground-breaking book *Developing Products in Half the Time* (Smith, 1997), these authors discuss manufactured products, not software products, but many of their ideas are relevant here. Their research pointed out that, on average, in the time frame from initial identification of a need to product ship, *more than half the project's time was taken up before the development project actually got under way!* "The front end is so fuzzy that people tend to forget that it even occurs," say the authors. They go on: "we have seen situations where as much as 90 percent of the development cycle elapsed before the team started work." How many projects with extremely aggressive schedules have you been on where everyone knows the project has been under "consideration" for months and months, if not years? Once the development team is appointed, the mantra becomes "Hurry, hurry." Where was all the hurry when management was "considering" the project?

Part of the delay in starting projects is concern about uncertainty. Because of managers' and teams' unwillingness to start before all questions are answered (and of course many of the answers will still be wrong), project start times slip—again and again. Then when the project does start—well, you know the drill. Starting early is a discipline that can greatly improve schedule performance, and at very little cost. Work on ways to get important projects off the ground early. I once worked with a medical software company in Canada that had been "investigating" a new product idea for a year. I finally convinced the company to try a few proof-of-concept iterations. The feedback was startling, but all too predictable: "We learned more about our product direction in a few 2-week iterations than we learned in the last 9 months of analysis."

One time I was asked by a senior manager in a software company, "How can you help us deliver this large product on schedule?" I replied, "Do less"—build fewer features. "Do less" is really the flip side of "Focus on what is important."

These two strategies—build less software and start sooner—sound simple on the surface, but in practice they can be very difficult to implement because of organizational inertia and politics. Even so, they can be very effective and are worth the effort to pursue.

Bibliography

Appelo, J. (2011). *Management 3.0: Leading Agile Developers, Developing Agile Leaders.* Upper Saddle River, NJ: Addison-Wesley.

Austin, R. D. (1996). *Measuring and Managing Performance in Organizations.* New York, NY: Dorset House.

Beck, K. (2000). *Extreme Programming Explained: Embrace Change.* Reading, MA: Addison-Wesley.

Bogsnes, B. (2008). *Implementing Beyond Budgeting: Unlocking the Performance Potential.* Hoboken, NJ: John Wiley.

Boyd, J. R. (1995). The Essence of Winning and Losing (a five-slide set by Boyd).

Christensen, C. M. (1997). *The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail.* Boston, MA: Harvard Business School Press.

Collins, J. (2001). *Good to Great: Why Some Companies Make the Leap and Others Don't.* New York, NY: Harper Business.

DeMarco, T. A. (1987). *Peopleware.* New York, NY: Dorset House.

Denning, S. (2010). *The Leader's Guide to Radical Management: Reinventing the Workplace for the 21st Century.* San Francisco, CA: Jossey-Bass.

Farson, R. (1997). *Management of the Absurd.* New York, NY: Free Press.

Fulghum, R. (1991). *Uh-Oh: Some Observations from Both Sides of the Refrigerator Door.* New York, NY: Ballantine Books.

Godin, S. (2010). *Linchpin: Are You Indispensable?* New York, NY: Portfolio.

Goldratt, E. M. (1984). *The Goal: Excellence in Manufacturing.* Croton-on-Hudson, NY: North River Press.

Hamel, G. (2009, February). Moonshots for Management. *Harvard Business Review, 87*(2), 91–98.

Hamel, G. (2012). *What Matters Now: How to Win in a World of Relentless Change, Ferocious Competition, and Unstoppable Innovation.* San Francisco: Jossey-Bass.

Hay Group. (2010). *Best Companies for Leadership Study.* Hay Group.

Highsmith, J. (2000). *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems.* New York, NY: Dorset House.

Highsmith, J. (2010). *Agile Project Management: Creative Innovative Products.* Upper Saddle River, NJ: Addison-Wesley.

Hock, D. (1999). *Birth of the Chaordic Age.* San Francisco, CA: Berrett-Koehler.

Horney, N. (2007). Executive Briefing on Agility. Retrieved from Agility Consulting, Inc.: www.agilityconsulting.com

Humble, J., & Farley, D. (2011). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.* Upper Saddle River, NJ: Addison-Wesley.

IBM. (2010). *Capitalizing on Complexity: Insights from the Global Chief Executive Officer Study.*

IBM. (2012). *Leading through Commections: Insights from the Global Chief Executive Officer Study 2012.*

Isaacson, W. (2011). *Steve Jobs.* New York, NY: Simon & Schuster.

Kanter, R. M. (1983). *The Change Masters.* New York, NY: Simon & Schuster.

Kauffman, S. (1995). *At Home in the Universe: The Search for the Laws of Self-Organization and Complexity.* New York, NY: Oxford University Press.

Keller, S. A. (2011). *Beyond Performance: How Great Organizations Build Ultimate Competitive Advantage.* Hoboken, NJ: John Wiley & Sons.

Kidder, T. (2009). *Mountains Beyond Mountains: The Quest of Dr. Paul Farmer, a Man Who Would Cure the World.* New York, NY: Random House.

Kotter, J. (2012, November). Accelerate! How the Most Innovative Companies Capitalize on Today's Rapid-Fire Strategic Challenges—and Still Make Their Numbers. *Harvard Business Review, 11,* 44–58.

McFarlan, W., & Benko, C. (2003). *Connecting the Dots: Aligning Projects with Objectives in Unpredictable Times.* Boston, MA: Harvard Business School Press.

Michalko, M. (2010, March 27). http://creativethinking.net/WP01_Home.htm.

Moon, Y. (2011). *Different: Escaping the Competitive Herd.* New York, NY: Crown Business.

Moore, G. A. (2011). *Escape Velocity: Free Your Company's Future from the Pull of the Past.* New York, NY: HarperCollins.

Perrin, T. (2007). Global WorkForce Study. Retrieved from Towers Perrin: http://www.towersperrin.com/tp/getwebcachedoc?webc=HRS/USA/2008/200802/GWS_handout_web.pdf

Petzinger, T. J. (1999). *The New Pioneers: The Men and Women Who Are Transforming the Workplace and Marketplace.* New York, NY: Simon & Schuster.

Pink, D. (2009). *Drive: The Surprising Truth about What Motivates Us.* New York, NY: Riverhead.

Pixton, P., Nickolaisen, N., Little, T., & McDonald, K. (2009). *Stand Back and Deliver: Accelerating Business Agility.* Upper Saddle River, NJ: Addison-Wesley.

Postrel, V. (1998). *The Future and Its Enemies.* New York, NY: Touchstone.

Reinertsen, D. G. (2009). *The Principles of Product Development Flow: Second Generation Lean Product Development.* Redondo Beach, CA: Celeritas.

Ries, E. (2011). *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses.* New York, NY: Crown Business.

Ross, J. C. (2010). The IT Unit of the Future. *Center for Information Systems Research*, *XI*(X), 1–3.

Rotman. (n.d.). Integrative Thinking. Retrieved from Rotman Business School: http://www.rotman.utoronto.ca/integrativethinking/definition.htm

Sikes, H. S. (2013, February). Competing in a Digital World: Four Lessons from the Software Industry. Retrieved from https://www.mckinseyquarterly.com/Competing_in_a_digital_world_Four_lessons_from_the_software_industry_3058

Smith, P. A. and Reinertsen, D. G. (1997). *Developing Products in Half the Time: New Rules, New Tools,* 2nd ed. Hoboken, NJ: John Wiley & Sons.

Sull, D. (2009). *The Upside of Turbulence: Seizing Opportunity in an Uncertain World.* New York, NY: Harper Business.

Taylor, W. C. (2011). *Practically Radical: Not-So-Crazy Ways to Transform Your Company, Shake up Your Industry, and Challenge Yourself.* New York City, NY: HarperCollins.

Thomke, S. (2003). *Experimentation Matters: Unlocking the Potential of New Technologies for Innovation.* Cambridge, MA: Harvard Business School Press.

Tushman, C. A. (April 2004). The Ambidextrous Organization. *Harvard Business Review, 82*(4), 74–81.

Ulrich, Dave and Smallwood, N. A. (2006). *How Leaders Build Value: Using People, Organization, and Other Intangibles to Get Bottom-Line Results.* Hoboken, NJ: John Wiley & Sons.

Weinberg, G. (1986). *The Secrets of Consulting: A Guide to Giving and Getting Advice Successfully.* New York, NY: Dorset House.

Weinberg, G., & Smith, S. (2000). *Amplifying Your Effectiveness: Collected Essays (The Satir Change Model).* New York, NY: Dorset House.

Williams, L. (2011). *Disrupt: Think the Unthinkable to Spark Transformation in Your Business.* Upper Saddle River, NJ: Free Press.