UNLOCKING AGILE'S MISSED POTENTIAL

ROBERT WEBBER

IEEE PRESS

IEEE computer society

WILEY

Unlocking Agile's Missed Potential

IEEE Press
445 Hoes Lane
Piscataway, NJ 08854

About IEEE Computer Society

IEEE Computer Society is the world's leading computing membership organization and the trusted information and career-development source for a global workforce of technology leaders including: professors, researchers, software engineers, IT professionals, employers, and students. The unmatched source for technology information, inspiration, and collaboration, the IEEE Computer Society is the source that computing professionals trust to provide high-quality, state-of-the-art information on an on-demand basis. The Computer Society provides a wide range of forums for top minds to come together, including technical conferences, publications, and a comprehensive digital library, unique training webinars, professional training, and the Tech Leader Training Partner Program to help organizations increase their staff's technical knowledge and expertise, as well as the personalized information tool my Computer. To find out more about the community for technology leaders, visit http://www.computer.org.

IEEE/Wiley Partnership

The IEEE Computer Society and Wiley partnership allows the CS Press authored book program to produce a number of exciting new titles in areas of computer science, computing, and networking with a special focus on software engineering. IEEE Computer Society members receive a 35% discount on Wiley titles by using their member discount code. Please contact IEEE Press for details.

To submit questions about the program or send proposals, please contact Mary Hatcher, Editor, Wiley-IEEE Press: Email: mhatcher@wiley.com, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030-5774.

Unlocking Agile's Missed Potential

Robert Webber

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

Library of Congress Cataloging-in-Publication Data Applied for:

Hardback ISBN: 9781119849087

Cover Design: Wiley
Cover Image: © Studio Romantic/Shutterstock.com

Set in 9.5/12.5pt STIXTwoText by Straive, Chennai, India

Contents

Author Biography

Robert Webber introduces a novel approach for Agile portfolio management and planning based on his extensive experience in leading software development organizations at GTE, AT&T, and three successful startups. His roles as VPs of engineering and product management, CEO, and consultant for Fortune 500 companies provided the technical and business perspectives to facilitate Agile planning while improving schedule and financial predictability.

Robert has an Electrical Engineering degree (honors) from the University of British Columbia. He holds five patents and received an Academy of Motion Pictures Arts and Sciences Technical Achievement Award for his innovative work on cable-mounted cameras.

Foreword

As the author of *Code Complete* I have been asked to write forewords for other people's books more times than I can count. I say "no" nearly every time. If readers see a foreword by the author of *Code Complete*, I believe they will expect the book to be as good as *Code Complete*. I rarely have enough confidence in other books to make that promise.

This book is a rare exception.

I have known Bob Webber for more than 20 years, and during that time we have had hundreds of discussions about the topics in this book. We've noted the unique ways that software executives see the world – often correctly, and sometimes flawed. We have shared observations about the relationship between the typical software development organization and product management, sales, and top executives. We've analyzed the strengths and weaknesses of Agile implementations – based to some degree on book knowledge, but based more on what we have seen with the numerous teams and companies we have helped first hand. We've puzzled about the reasons that Agile so often fails to live up to expectations. We have had many lengthy conversations about managing portfolios of software projects, releases, value streams, and features – and how all that relates to Agile development.

This book begins with Bob's identification of a major problem: The typical organization's commitment to Waterfall planning undermines its ability to be truly agile, and that in turn prevents it from realizing the benefits of Agile development. Don't mistake this for the common doctrinaire argument, "You can't realize the benefits of Agile unless every remote corner of your organization becomes Agile." Bob's analysis is more focused and far more nuanced – it has to do with the specific interactions between top executives, product management, and software development that give rise to a destructive mismatch between Waterfall planning at the business level that eviscerates the use of Agile practices in the software organization.

After his clear explanations of this fatal weakness in Agile implementations, Bob presents an innovative solution, which is to treat software initiatives as a portfolio of investments, both small and large, and to prioritize them in a specific way that maximizes economic return. The key is in how investments are sequenced – some sequences produce far higher return than others, even when the work in total is all the same. Common sense turns out to be a weak method for sequencing investments, and this book presents a sophisticated approach that works markedly better. There is some math in that section, but it's worth it. When this approach is used, prioritization of "features" and initiatives becomes more objective and easier. Historical friction between engineering and product management dissipates. The organization becomes more successful.

The investment prioritization algorithm in this book – the sequencing – is a core that supports improved organizational performance in other areas. The core provides guidance for roadmap planning, increased productivity, and increased innovation. These are not small benefits. Roadmap planning is perhaps the most common Achilles heel I have seen in Agile implementations. Many organizations treat increased productivity and increased innovation as primary goals. Bob's software investment portfolio approach supports them all.

Despite containing a significant amount of math and some economic discussion, this is not an academic book. It is based on real-world experience, and it provides a step-by-step guide that you can use to implement this software investment portfolio approach in your organization. Bob provides practical resources such as an investment planning template and an extended case study that illustrates how all the concepts and techniques work together in practice.

Most books can't measure up to the promise of *Code Complete*, but the approach in this book can. Most organizations perform their software work in inefficient ways. Simply by doing the same work in a different order – changing the sequence – organizations can increase the returns on their software investments dramatically.

From my conversations with Bob over more than two decades, I can say with certainty that there is no one in the world who has spent more time understanding how to construct a portfolio of software initiatives and how to optimize the returns on such portfolios. This book describes Bob's best thinking on the subject, and I recommend it whole heartedly.

Steve McConnell, author of *Code Complete*

Preface

This is a journey that began in 2011 when I started a second career as a product development consultant. I wanted to leave behind the pressure of delivering software products and transition to a role that would allow me more time flexibility to pursue my interests. I wouldn't change anything in my first career. It provided me with a perspective from the engineering and business side to solve a software industry problem – the missed expectations of Agile development.

I would never have become a consultant if it wasn't for Steve McConnell, an icon of software engineering. I had read his books and admired how he updated proven software engineering practices for the current generation of software developers. I became a client of Steve's training and consulting company, Construx Software.

When I decided to leave the corporate treadmill, Construx was the only company for which I wanted to consult. Steve let me determine how I could best contribute and gave me the flexibility to focus on my interests. It gave me the opportunity to apply my knowledge and experience to help software companies build higher value software and release it faster. It also provided me with insight into what was really going on in Agile, and it wasn't the Agile I was reading about in books.

This was a period when everyone was "going Agile." I saw larger companies stumble when they tried to apply a simplistic Scrum model designed for small organizations. Product development organizations had the biggest challenges. Where is the product owner who knows everything about everything in an international organization with a complex domain and diverse customers? And how can engineering do real Agile development when product management plans large releases with fixed schedules stuffed with features?

My consulting assessments always pointed out that organizations were not really doing Agile. A basic tenet of Scrum is that teams have time to incorporate feedback and correct defects during sprint development. Instead, teams were still driven by imposed schedules and barely had time to create basic functionality. Teams were demotivated from continually missing schedules. Sprints were not "potentially shippable." Defects mounted up during sprints which led to major rework

in a "hardening sprint." Engineers were feature implementors instead of providing innovative solutions to problems conveyed to them in user stories and epics.

The value of releasing faster seemed obvious. Why did these organizations continue to use traditional Waterfall release planning methods with fixed schedules and content? Waterfall planning hadn't ever achieved the predictability they sought. How did they think that Agile roadmap commitments could be made prior to requirements development given that requirements were allowed to change?

Companies continue to use Waterfall planning because they are forced to make multiyear financial commitments. The sales organization needs to know the products and features they can sell before they will commit to revenue. And projects need financial justification based on delivery schedule and content commitments. Internal operations groups must budget with the expectation of increased productivity dependent on new features. Agile didn't address these issues. The somewhat flippant response of the Agile community was, "The business needs to be more Agile." Even though Waterfall planning leaves a lot to be desired in terms of predictability, it is the only way product managers know how to plan.

Some companies implemented the SAFe Agile framework to integrate business planning and Agile development. However, SAFe just institutionalized feature-based Waterfall planning. Product managers liked it because they didn't have to change the way they planned large releases at the feature-level. SAFe included an option for Lean Canvas planning but is not widely used in larger organizations. It's difficult to get funding approved for market experimentation. The business wants to see the releases and features they are paying for before funding development.

I've experienced a few epiphanies in my career. One occurred at the annual Construx software leadership summit in 2011 that influenced this book. Donald Reinertsen, author of *The Principles of Product Development Flow* [1], was a keynote speaker. His book presents a queue-based model of software development analogous to manufacturing organizations. The book includes a concept that can potentially change the way software is planned. It's called "Cost of Delay."

Cost of Delay is a simple yet elegant perspective of the software development process: "How much more money would you make if you could release your software one month earlier?" Reinertsen went on to show that prioritizing projects based on something called, "Weighted Shortest Job First" (WSJF), would optimize income. WSJF is the Cost of Delay divided by the project cycle time. It essentially says that companies make more money when they limit functionality and release and earn income faster. I recognized that WSJF can facilitate the mythical "Minimum Marketable Feature Set" (MMFS) advocated by Agile. I say mythical because sales organizations refused to compromise on functionality. They wanted anything and everything that could possibly lead to a sale in any country.

Thus began the journey to this book. Cost of Delay and WSJF were great concepts, but they were not widely adopted. Companies found it difficult to calculate Cost of Delay for non-linear post-release income profiles. Reinertsen's formula works for linear income profiles with constant income per month. I recognized that WSJF had to be simplified and product managers need to have methods and tools to simplify the calculations for any income profile.

My conclusion was that most companies have not achieved the potential of Agile because they are not really doing Agile the way it was envisioned. They are still trying to force-fit Agile into a Waterfall planning framework. This book enables these organizations to reap the benefits of their investments in Agile development by incorporating Agile planning and deployment. It also lets larger organizations replicate the powerful dynamics of a startup instead of the divisiveness and friction I observed between product management and engineering. The book describes how larger organization can build collaborative Agile product teams.

There is one other major influence that helped shape my solution. Tom Gilb has a long history of contributing to the software industry in the areas of software metrics, software inspections, and evolutionary development processes. Tom gave a talk on stakeholder value analysis at Construx in 2011. His method provides a way to dig down to what users really want before getting into requirements detail. It was the perfect fit for my proposed approach to Agile planning.

I want to thank Steve McConnell for his patience and support for this journey. Steve is a deep thinker who kept me on track by challenging many of my assertions. His philosophy of hiring smart people and letting them figure out how they can best contribute gave me the freedom to pursue my ideas. I also had the benefit of stimulating collaborations with other Construx consultants. Steve's recent book, *More Effective Agile* [2], is a useful and practical guide for engineering leaders that complements my work.

Don Reinertsen's book and principles provided the foundation for this new approach for Agile planning. He is an inspirational speaker with the analytical mind of an electrical engineer. I had the pleasure of talking to Don about some of my ideas at the Construx leadership summit he attended two years after his first talk. I highly recommend reading, *The Principles of Product Development Flow*. WSJF was a way to illustrate the value of reducing cycle time. The book contains much more. He goes on to show how cycle times can be drastically reduced by managing queues that build up in software development.

I have one other person to thank – my wife, Susan. In addition to being a great partner and mother, her PhD in Psychology and experience in leading change at high-tech companies gave me insight into the "soft" side of software development that needs to be addressed to realize the full potential of Agile. We can put all the procedures and processes in place, but little happens without human motivation.

This book explains how Agile can create high-performance self-directed teams, yet how it has been squandered by the constraints and controls from traditional development, especially those driven by the perpetuation of Waterfall planning.

And, of course, I want to thank Susan for infinite patience and support for the years I've spent developing and refining the Investment approach.

References

1 Reinertsen, D.G. (2009). *The Principles of Product Development Flow: Second Generation Lean Product Development*. Celeritas Publishing.
2 McConnell, S. (2019). *More Effective Agile: A Roadmap for Software Leaders*. Construx Press.

Introduction

The Lost Potential of Agile Development

I wrote this book because I want to see Agile development fulfill its promises. My assertion is that organizations that retain Waterfall planning have not obtained the true benefits of Agile. I found that organizations often referred to their software development frameworks as a "hybrid" approach. After almost a decade of software development consulting, I learned to recognize that the term "hybrid" was a sure sign of a failed Agile implementation. It meant that engineering was still held to fixed schedule and release content commitments. This book will show that Waterfall planning is the antithesis of Agile. However, product managers can now adopt new planning methods that support Agile development while meeting the business need for predictability. Engineering can finally do Agile with content flexibility.

After a successful career in software development, I joined Construx Software as a consultant. I gave seminars on requirements, innovation, and project management. I also conducted software development organizational assessments, which included many Fortune 500 companies. I reviewed their practices and organizational structures to recommend how to improve software development productivity and predictability, and to reduce time-to-market. I had the opportunity to see what was really happening "under the hood" in Agile.

While at Construx, I also had the opportunity to moderate group sessions at the annual Construx Leadership Summit where software executives shared their challenges and solutions. I heard the same concerns over and over. Engineering was still being held accountable for schedule, scope and budget commitments made during the release planning stage. Many reported that they had not been able to deliver on the expectations of Agile because of the continued use of Waterfall planning.

In this book, we will consider "Waterfall planning" to be any development framework where release development cost, schedule, and scope are fixed during an initial planning period. Scope is typically defined in terms of features. Development may be split up into several increments of release functionality, but the overall release schedule and scope have already been determined.

Waterfall planning with fixed schedule, scope, and resources undermines the fundamental Agile principles of evolving requirements to incorporate feedback and additional knowledge during development. Agile teams were supposed to pull work from queues at their own pace to ensure they can incorporate feedback and new requirements during the development cycle. Variable content was supposed to give teams time to correct defects to maintain "potentially shippable" software instead of building up mounds of defects that had to be corrected near the end of Waterfall projects. Agile teams were supposed to deploy software faster, with the ultimate goal of continuous delivery. However, planning begins with the assumption of large releases that are then "filled up" with myriad features of questionable value.

This book mainly addresses issues observed in larger organizations with separate product management departments. I've seen many cases where Agile works great with small teams under the guidance of a knowledgeable product owner who truly understands customers and the market. However, something seems to go wrong when silos go up around formalized product management departments. There is a major disconnect between the Waterfall planning perpetuated by product managers and the variable content necessary for Agile development. Product managers have not been provided alternatives to Waterfall planning that meet the predictability needs of the business and the schedule and/or content flexibility required by engineering.

I've also observed that there are common characteristics of the Agile examples touted by the Agile community. They are mostly in companies with a Business to Consumer (B–C) business model where the engineers have a solid understanding of the domain. Consider Facebook or Apple versus a medical device company serving international patients. The second characteristic of these example companies is the ability to attract the top percentiles of engineering talent. The third is that engineers are not constrained by Waterfall planning mandated to try to meet quarterly and annual financial targets.

Engineers with deep domain knowledge, especially when they are potential consumers of the product, can work effectively with little guidance. They can fill in the blanks of high-level user stories without frequent interaction with product owners. The Business to Business (B–B) companies that I assessed have complex domains and were never able to convey the necessary requirements detail through user stories alone. As an example, I worked with a company where Indian engineers were implementing highly regulated casino management software from user

stories when they had never been in a casino in their lives! It wasn't Agile development – just code and fix.

Talent is another critical factor. I've worked with teams of talented engineers that can deliver software with little or no software development framework or methodology. These are the top 15% of self-motivated engineers who will overcome anything in their way. And they can write virtually error-free code. Who gets this demographic? The "cream of the crop" goes to startups or fashionable companies like Google, Microsoft, Facebook, and Spotify. The rest of the industry is left with average engineers who need a solid development framework and detailed requirements.

The third Agile success element is the main subject of this book. The example companies have abandoned Waterfall release planning and embraced Agile planning, often because they are dominant in their markets and can set their own schedules. They are probably not like your company with a demanding sales team committed to quarterly revenue based on what is available for them to sell. I have not seen an effective Agile implementation where Waterfall planning persists.

Putting these three Agile success elements together, one realizes that most of the "ideal Agile" examples and literary advice is based on a small fraction of the software industry comprised of startups and sexy B–C companies that have embraced Agile planning and are loaded with engineering talent. I always felt bad for the Construx summit attendees who listened with envy about amazing Agile implementations and wondered what was wrong with them. It is not them. They could be making the same presentations given the same ideal conditions discussed above.

This book is for the frustrated "silent majority" of software development organizations where Agile has not met expectations. It addresses the typical large company with complex domains and average talent. These companies can now adopt Agile planning and deployment and improve predictability for the business. This book describes how highly motivated Agile product teams can be built around business problems with quantified value. Engineers are provided with the problem to solve instead of trying to convey detailed implementation requirements through user stories with an uninvolved or uninformed product owner.

The word Agile in this book refers to methods other than just Scrum, like Kanban. I use Scrum terminology in this book, but the same investment planning principles and methods apply to any Agile method using iterative development with evolving requirements where self-directed teams pull their work from backlogs.

There is one other term used throughout this book that needs to be clarified for those of you with some knowledge of company financials. When I use the term "income" I am referring to "Net Income," which is the income received with product cost, expenses and taxes removed.

Missed Business Expectations

Agile has not addressed the real-world need for financial predictability in business. We do know that roadmap commitments were usually not met with Waterfall development. We learned there is a limit to the accuracy of software development effort and schedule estimation. With the advent of Agile, engineering was now expected to make commitments even before requirements were developed, exacerbating the predictability problem.

The Agile community response of, "The business needs to be more Agile" did nothing to help product and project managers who were still held accountable for long-term commitments of schedule, scope, and cost. This widened the division between product management and engineering. Engineering is frustrated by not being able to do Agile development the way they were taught, having to resort to "hybrid" methods. Product and project management are frustrated because of the reluctance of engineering to make release commitments.

Certainly, there are organizations that have been able to manage variable content with Waterfall planning by reserving software development contingency. But these organizations are rare because contingency doesn't typically survive the pressure to commit during release planning.

I don't blame product management for the continued use of Waterfall planning. They weren't provided with an alternative. Product managers were left out of Agile because, in the view of the Agile community, they could be replaced with product owners who sit with a team to provide verbal clarification for everything they need to know. And, according to the Agile community, there was no need to get requirements right because they could release small increments and gain the benefit of rapid customer feedback. We went from the Waterfall world where we tried to know everything about everything before development starts to the purist Agile view where we can't know anything about anything.

Agile did not account for the real-world challenges faced by software product managers in established organizations. In the real world, sales departments must commit to future revenue so that projected earnings and growth rates can be met. These commitments become part of multiyear business plans committed to a board of directors. Of course, sales people naturally want to know what they will have to sell and when they can sell it before they can agree to sales quotas. This results in expectations that product managers can predict delivery at the feature-level on a multiyear roadmap.

Agile grew quickly from a methodology for small development teams to major projects and products without defining methods for product managers to meet the predictability demanded by the business.

My experience as a VP of engineering, VP of product management and a CEO in the software industry has enabled me to view Agile development from all sides. It has given me a good perspective on how the needs of all stakeholders need to

be addressed for Agile to be truly successful. Agile development is trying to swim upstream in a current of established business processes based on financial reporting requirements. The business will always demand predictability as long as our capitalist system requires CEOs to make earnings projections. Agile needs to meet business predictability expectations to be deemed successful. "The business needs to be more Agile" is a naïve response from the Agile community. I doubt that any business leader would argue with this, but agility can only exist within the predictability constraints of the business.

A New Approach to Agile Planning

Let's examine what predictability actually means to the business world. Multiyear financial forecast commitments must be made to shareholders and other stakeholders. Your company's executives really don't care about software estimation uncertainty. They want financial predictability. Agile can deliver predictability for the business in two ways. The first is to learn to make commitments that have high probability of being met, which is addressed in this book. The second is to plan and focus on delivering predictable financials and R&D Return on Investment (ROI). The Investment method described in this book enable product management and engineering to work together to balance income projections and development costs to meet financial targets.

Today features are still the primary planning element in software development. Features are an increment of value with a common understanding of benefit by sales, product management, and engineering. However, it is generally not possible to estimate financial return on a feature basis at the planning stage. The best we could do was feature prioritization with subjective rating methods. Some of you may be familiar with T-shirt sizing, where both "business value" and development effort are weighted for a feature. Features with the highest "business value" and lowest cost have higher priority. T-shirt sizing has been a useful method for subjective prioritization of features, but it doesn't project income on a feature basis.

The need for financial predictability on the business side, combined with the inherent Agile development need of content delivery flexibility, led to the Investment planning approach introduced in this book. "Investment" in the Agile planning context is defined as the smallest increment of software functionality that has the potential to increase income and/or reduce operational expenses. To generate income a customer must perceive value. Willingness to pay for something affirms business value.

Here are some examples of Investments:

New Sales

- Interactive dashboards to reduce customer operational costs by 10% to increase annual revenue by 5% over the next three years

Defense

- EU competitive features to prevent market loss of 20% over three years

Expense Reductions

- Multi-tenant capability to reduce annual hosting costs by 15%

Refactoring

- Deployment simplification to accelerate $3M in income over the next three years

Investments become the Agile planning element with this new approach. Features can then be justified and prioritized within an Investment based on their relative contribution to the financial target instead of trying to prioritize hundreds of features. This book uses Cost of Delay principles to show that Investments that generate the highest income with the lowest development effort should be prioritized higher.

The condition "potentially deployable" is used to separate the release packaging problem from the prioritization of development work. There are often valid reasons why multiple Investments may be packaged in the same release. Regulatory validation requirements are an example. Or there may be deployment process constraints involving retraining of operations staff. However, even if we are planning to release multiple Investments in a release, they should be developed in the order of highest value.

We want to be "agile" by being able to replace lower value Investments with higher value Investments at any time during the development cycle. With the ability to assign income to increments of software development, we can finally quantify the millions of dollars that are delayed because of the inability to deploy individual Investments. This can be viewed as "the opportunity cost of not being Agile." Investments in software infrastructure and deployment processes to release faster can now be justified.

Product managers may view Investments as conventional "initiatives" they have traditionally used in planning. There is a key difference. Initiatives relate to marketing programs of which R&D costs are only one component. Marketing initiatives include all activities necessary to develop, market and profit from a product. Investments defined in this book are based only on R&D cost so we can prioritize work within constrained software development capacity to maximize ROI. In effect, an R&D "Investment" is one component of an "initiative."

Engineering may view "Investments" as the Minimum Marketable Feature Set (MMFS) long advocated by Agile development. The comparison is valid for the case where an MMFS is associated with a financial forecast. However, the MMFS term is also used to describe an increment of functionality that adds any perceived

"value" for the customer. An Investment can be an MMFS, but not every MMFS is an Investment. The book will show, however, how the Investment approach facilitates the MMFS because Investments with lower development cost are prioritized higher. A Minimum Viable Product (MVP) may be created to validate the perceived benefit of an increment of software without revenue expectations just to obtain customer feedback. MVPs are Investments only if they generate income.

You may be questioning at this point how we can assign income forecasts at the Investment level because it requires commitment from sales and/or operations during the planning stage. Obtaining revenue and cost reduction commitments prior to starting development has long been a goal of product managers. However, release price bundling and the inability to forecast income at the feature-level have made this impractical. Product management can now get income commitments on an Investment before it can be prioritized within the Investment backlog. Methods to collaboratively plan income impacts with sales and operations prior to development are provided in this book.

Addressing Traditional Software Development Challenges

Don Reinertsen introduced Cost of Delay and Weighted Shortest Job First (WSJF) in his book *Principles of Product Development Flow* [1], but these calculations have not been widely adopted. Reinertsen introduced "Weighted Shortest Job First" (WSJF) as a project prioritization method. He showed that prioritization by monthly income divided by cycle time always provides the optimal development order of projects. He refers to monthly income as the "Cost of Delay." Although widely acknowledged as the optimal prioritization method, WSJF has posed a challenge for the software industry. It's not possible to prioritize software development within a release because features typically don't have income projections. His WSJF based on Cost of Delay in terms of income can now be used to prioritize Investments.

Inability to reduce technical debt has long been a point of frustration for engineering. It's not realistic to expect product managers to prioritize work with nebulous financial value ahead of the feature demands of their stakeholders. Engineering can now propose Investments for technical work that can be prioritized against business investments. The catch is that engineering needs to justify how these technical Investments improve financial results to offset their development costs and the cost of delaying income from other Investments. This book teaches engineering how to do that.

Investments also facilitate collaboration between product management and engineering. Relationships between product management and engineering

departments have often been adversarial because they have conflicting objectives. Product management pressures engineering to incorporate features into releases under pressure from sales and operations. Engineering naturally avoids commitment because they are the ones that end up being held accountable, especially when they know that additional work will come in without schedule relief. Product managers and engineering can now share common goals with Investment planning. Their shared objective is to define Investments with the lowest development costs that maximize income. They also need to ensure that Investment scope does not increase without offsetting income to maintain WSJF priority. Product teams with common goals can now be formed around Investments.

Agile teams can be staffed on the basis of Investments. We want Agile teams to be focused on value rather than functionality. By definition, Investments provide quantifiable value. Consider the higher motivation of an Agile team working together to solve problems directly related to customer value, rather than implementing pre-determined functionality. This achieves the original objective of Agile development of delivering increments of true customer value.

Lastly, Investment planning addresses the role conflict between product managers and product owners. I have observed product managers who still dwell on the details of requirements definition, even down to User Interface (UI) layouts. Investment planning elevates the role of product managers to focus on value rather than functionality. Product managers can concentrate on identifying opportunities for customer value for which customers are willing to pay. Product owners can focus on the functionality necessary to achieve that value. Investment planning provides complementary roles for product management and product owners that facilitate collaboration and innovation.

Motivation and Innovation

I have included a chapter on each of these topics based on some amazing insight and training I received early in my career. I felt that this book would be incomplete without addressing the organizational cultural aspects necessary to fully realize the potential of the Investment approach and Agile development.

Motivation must be established in your organization to adopt and use the practices provided in this book. Most organizational change implementations fail because they don't provide the motivation to follow new practices. And can you really change the divisive and confrontational relationship between product management and engineering in your organization? Can product management ever get sales to accept the MMFS instead of continual pressure for features of little value? You'll see that there are established principles of organizational behavior that show how to align consequences with new practices to facilitate

sustained behavior change. These principles have been taken into account for all the practices defined in this book. The practices themselves drive culture change.

Innovation is at the end of the Agile rainbow. However, the potential of Agile to ignite innovation is rarely achieved. The Investment model described in this book facilitates innovation by allowing engineering to contribute at the planning stage. Investments convey the problem to solve instead of constraining engineers with features and prescriptive requirements. However, even with new practices to facilitate collaboration and innovation, it won't happen unless you can create the nebulous "culture of innovation."

Culture is defined in terms of the organizational behavior model of motivation. We will see that innovation is a product of behaviors that must be frequently reinforced in your organization. Presenting ideas or building a proof of concept on one's own time are examples of behaviors observed in companies with a "culture of innovation." Behaviors require motivation, hence the connection with the organizational behavior model. Chapter 14 tells you exactly what your organization needs to do to reach the end of the Agile rainbow where product managers and engineering collaborate to produce innovative products within short cycle times that outdistance your competition.

Your Organization

This book provides everything you need in terms of practices, templates, and tools for you to implement the Investment model in your organization. The final chapter in this book provides a step-by-step guide, from gaining internal support to creating your first Agile Investment roadmap. It also tells you how to align consequences in your organization to make lasting change. This book provides references to downloadable spreadsheets to support Investment planning, including Cost of Delay calculations. They can be accessed at: www.construx.com/product-flow-optimization-calculators.

Reference

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