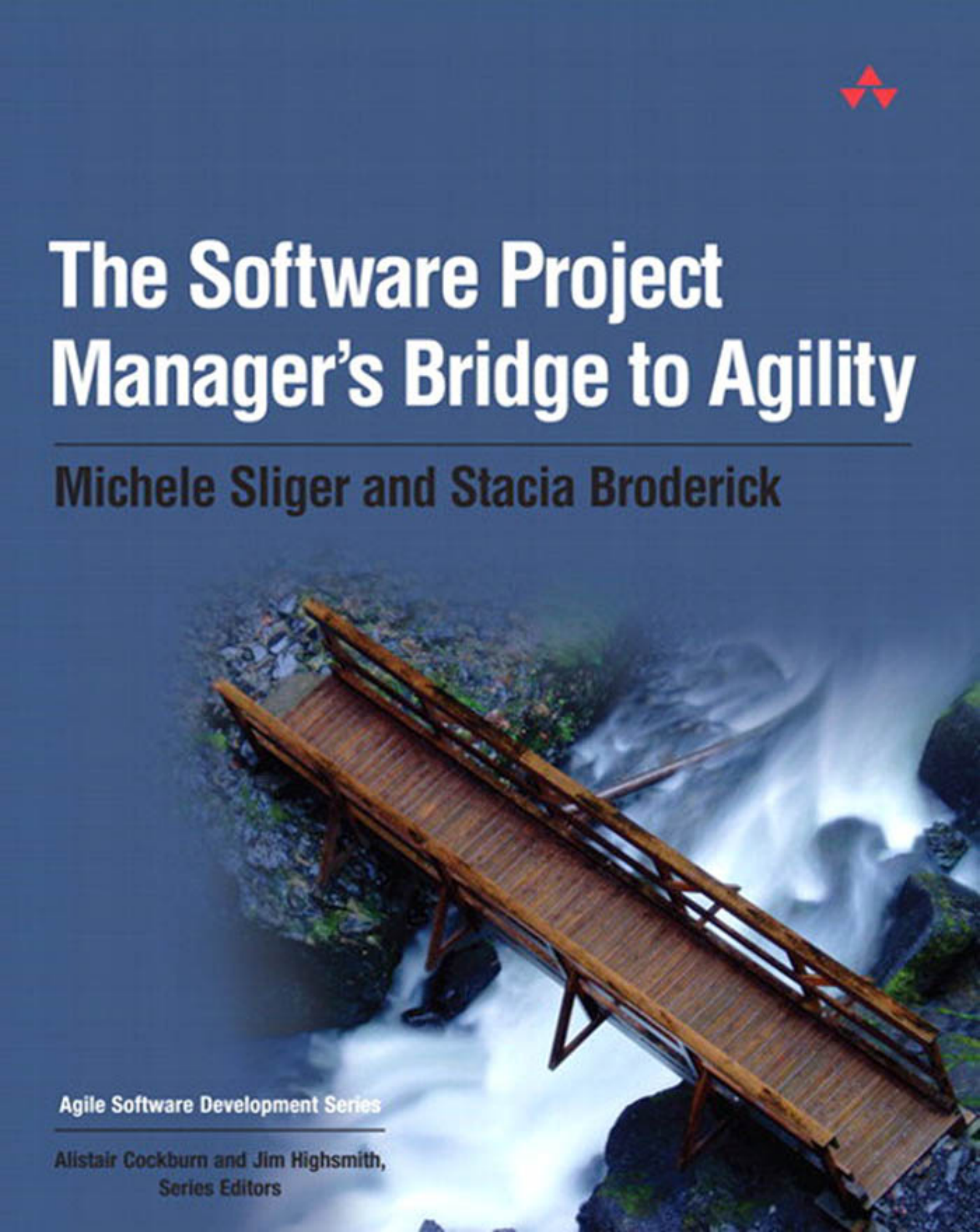


The Software Project Manager's Bridge to Agility

Michele Sliger and Stacia Broderick

Agile Software Development Series

Alistair Cockburn and Jim Highsmith,
Series Editors

Library of Congress Cataloging-in-Publication Data

Sliger, Michele, 1964-
The software project manager's bridge to agility / Michele Sliger, Stacia Broderick.
p. cm.
Includes bibliographical references and index.
ISBN 0-321-50275-2 (pbk. : alk. paper) 1. Computer software—Development—Management.
2. Agile software development . I. Broderick, Stacia, 1974- II. Title.

QA76.76.D47S563 2008
005.1068—dc22

2008008524

ISBN-13: 978-0-321-50275-9
ISBN-10: 0-321-50275-2

Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
First printing May 2008

This Book Is Safari Enabled

The Safari® Enabled icon on the cover of your favorite technology book means the book is available through Safari Bookshelf. When you buy this book, you get free access to the online edition for 45 days.

Safari Bookshelf is an electronic reference library that lets you easily search thousands of technical books, find code samples, download chapters, and access technical information whenever and wherever you need it.

To gain 45-day Safari Enabled access to this book:

- Go to http://www.informit.com/onlineedition
- Complete the brief registration form
- Enter the coupon code I1GL-5UBF-62GU-7QF5-HBBE

If you have difficulty registering on Safari Bookshelf or accessing the online edition, please e-mail customer-service@safaribooksonline.com.

Editor-in-Chief
Karen Gettman

Executive Editor
Chris Guzikowski

Senior Development Editor
Chris Zahn

Managing Editor
Kristy Hart

Project Editor
Betsy Harris

Copy Editor
Bart Reed

Indexer
Erika Millen

Proofreader
Paula Lowell

Publishing Coordinator
Raina Chrobak

Cover Designer
Alan Clements

Compositor
Nonie Ratcliff

Preface

We are dedicated to the use of agile practices in software development (a.k.a. agilists), but we didn't start out that way. We began as Project Management Professionals (PMP®)[1] who used more traditional methods in the development of software.

Why We Wrote This Book

We followed the approaches outlined in the Project Management Institute's *A Guide to the Project Management Body of Knowledge—Third Edition* (*PMBOK® Guide*) for much of our careers, and in moving to agile approaches we became more aware of the misconceptions out there surrounding the subject matter of this book—incorrect ideas that we once believed as well. Now as agile consultants, we continue to hear our clients say that they believe (incorrectly) that if they are to keep their PMP certification and follow the practices outlined in the *PMBOK® Guide* that they must use a waterfall-like methodology. We also hear the mistaken belief that agile approaches lack discipline and rigor. And we see the fear and dismay of those who believe that their investment in the Project Management Institute (PMI) may be for naught if they follow the path to agility.

It is our goal to dispel these myths in our book and show that the Third Edition of the *PMBOK® Guide* does in fact support agile software development methods and that the investment that project managers have made in the PMI and in the practices outlined in the *PMBOK® Guide* are still solid and appropriate to pursue. It is clear to us that the *PMBOK® Guide* is methodology-neutral and supports good project management practices regardless of the approach chosen. Although many are already aware of this fact, we find that there are still many who are not. As PMPs who are now agile enthusiasts, we feel it is important to also dispel the mistaken notion in the agile community that PMPs cannot be good agile project managers. We would like to build a bridge between the two—thus the need for this book.

Structure and Content of the Book

Accordingly, we've put much of the detail concerning this bridging in Part II, where we map the *PMBOK® Guide's* practices to agile practices. It is our intent to show project managers that in moving to an agile methodology, they do not move away from implementing PMI-recommended practices—they simply implement the practices in a different way, making sure that the intent behind these practices remains true. In some chapters you'll find a clear mapping, whereas in others the mapping is more imprecise. This book is intended to be a guide, a way to take the lexicon you are already familiar with and relate it to a new way of developing software. This book will not replace any of the more specific agile practice books in the market today, and we encourage you to supplement this reading with other books on particular agile methods (Scrum, XP, Lean, Crystal, and so on).

The next several sections provide a quick preview of the book.

Part I: An Agile Overview

Part I introduces you to the basic terms and concepts of agile software development. We begin in the first chapter ("What Is Agile?") with a look back at the emergence of agile ideas in the history of software development. You may be surprised to learn that even Winston Royce's paper on the waterfall approach recommended an iterative cycle and the involvement of the end user in the whole of the project! From this history we move forward and review the concepts behind the Agile Manifesto and its associated principles, which are the basis of all agile software development frameworks.

In Chapter 2, "Mapping from the *PMBOK® Guide* to Agile," we look at the history of the PMI and its most famous contribution to the practice of project management, the *PMBOK® Guide.* We'll examine how the *PMBOK® Guide* project lifecycle phases and project management process groups can be related to the Agile Fractal. And we'll reiterate again that you can be agile and be in keeping with the recommendations outlined in the *PMBOK® Guide.*

Chapter 3, "The Agile Project Lifecycle in Detail," describes the agile project lifecycle—from release planning to iteration planning to daily planning—and how demos, reviews, and retrospectives at the end of each

iteration allow the team to continually improve. This chapter begins the use of terminology and concepts that we expand on throughout the rest of the book.

Part II: The Bridge: Relating *PMBOK® Guide* Practices to Agile Practices

This is the part of the book where we review each of the *PMBOK® Guide* knowledge areas and discuss what you used to do as a traditional project manager, and what you should consider doing instead as an agile project manager. As the title implies, we are trying to build an explicit bridge between the traditional and the agile, and provide you with guidance on what tasks and activities you should substitute—or keep.

As it is in the *PMBOK® Guide,* the knowledge areas are not in any type of chronological order. In both traditional and agile project management settings, you will find yourself doing most of these activities in parallel.

Because there is some overlap in the knowledge areas, you may find some ideas and concepts repeated. We did this intentionally, because we expect many of you to use this part of the book as a reference guide, and may therefore start with any of these chapters in any order. However, to keep the repetition to a minimum, we do use references to other chapters rather than rewrite large sections.

The chapters in Part II include the following:

- Chapter 4: "Integration Management"
- Chapter 5: "Scope Management"
- Chapter 6: "Time Management"
- Chapter 7: "Cost Management"
- Chapter 8: "Quality Management"
- Chapter 9: "Human Resources Management"
- Chapter 10: "Communications Management"
- Chapter 11: "Risk Management"
- Chapter 12: "Procurement Management"

Part III: Crossing the Bridge to Agile

Whereas Part II covers the specific practical activity changes, Part III covers the softer skills of being an agent of change and what this change means for you personally and professionally. Having answered much of the "what" you need to do in Part II, we turn our focus to "how" to make these changes in Part III. From how your role changes, to how you'll work with others who aren't agile, to what to watch out for, we respond to the commonly asked questions of those who are about to cross the bridge. The chapters in Part III complete the main body of the book:

- Chapter 13: "How Will My Responsibilities Change?"
- Chapter 14: "How Will I Work with Other Teams Who Aren't Agile?"
- Chapter 15: "How Can a Project Management Office Support Agile?"
- Chapter 16: "Selling the Benefits of Agile"
- Chapter 17: "Common Mistakes"

Appendixes

We've included two appendixes we hope you will find useful. Appendix A, "Agile Methodologies," runs down a number of the software development methodologies that fall under the agile umbrella. Appendix B, "Agile Artifacts," includes a look at the typical agile project "artifacts."

Who This Book Is For

Although this book is targeted at software project managers who are members of the PMI, anyone who is doing traditional software project management will benefit from seeing agility presented in terminology to which they are accustomed. We will refer to these long-established methodologies as "waterfall," "plan-driven," or "traditional," all of which refer to sequential, phased, noniterative approaches to software development.

Final Thoughts

We should also make it clear that we are not sanctioned by PMI or any of its representatives. This book is the result of our research, interpretation, and experience. Although we used the Third Edition of the *PMBOK® Guide* in our studies, we expect that as the *PMBOK® Guide* goes through further revisions, you will still find the concepts presented here to be relevant.

Endnote

1. "PMP," "PMI," and "PMBOK Guide" are registered marks of Project Management Institute, Inc.

Introduction

How One Project Manager Crossed the Bridge

I'm Stacia Broderick, and I want to convey a deeply personal story of change in hopes of helping you recognize the importance of listening to yourself and learning how to grow, even when it is quite uncomfortable and scary.

I have been a project manager since 1993, agile since 2003. I am also a PMP, formally trained in the lexicon of the thousands of certified Project Management Professionals who went before me. When I started managing projects, I took certain pride in my abilities to plan a project, learned how to enter data into a project management tool, held status meetings, negotiated with contractors and third-party sourcing for resources and materials, mitigated risks in the project and, of course, controlled scope. I could perform forward- and backward-pass calculations in my sleep.

Project management was a perfect fit for me, who, as a third-grader, resource-loaded my two sisters and I into weekly rotating chore schedules. I even designed a process for reducing the number of dishwashing loads by only emptying the dishwasher based on a pull-and-batch system (pull a dish only when needed, and no more frequently; gather all dirty dishes in the sink until time to reload dishwasher; reload all at once), but my father did not support this new approach. For me—a self-admitted control freak—project management was a perfect fit.

My conflict with Scrum, one of the agile approaches to software development, began in 2003. I was vehemently opposed to this new, lightweight, not-sponsored-by-any-formal-governing-body methodology (or so I had thought). My life was turned upside down when Ken Schwaber came to train and mentor our team of managers and software developers. As a devout PMP, or perhaps as a result of still being relatively new to software development, I was a bit leery of Ken's initial teachings about self-managed teams and iterative development. As I drifted in and out of the two days of ScrumMaster training, the line that caught most of my attention was, "You have no power." Ken meant it in the sense that the product owner and delivery team roles would be collaborative in nature, and that a project manager wasn't the decision-maker in Scrum. Like a mantra, I repeated this line to see if I could get used to it. I kept thinking, "How could you possibly manage a project or people without power? Wasn't it a prerequisite that you had to muscle your way through a project and demand that people work overtime and weekends (but promise to feed them free pizza)? As the project team grew fatter and physically slower, didn't this mean you could more easily beat them into submission?" (I kid, I kid.)

When my boss failed to show up to ScrumMaster training, I was automatically thrown to the lions as my (now ex)-boss's replacement. Congratulations to me: I was the newly minted ScrumMaster of three project teams.

Wow. So now I had to lead people. I had never *lead* people before. I had certainly managed them, and collected the status of their tasks, and quizzed them on how much time was remaining on those tasks. And, of course, I questioned their estimates. (Everyone knows that developers are horrible estimators!) I sometimes even gave my helpful opinion on whether certain technical tasks were easy or difficult, much to the developers' delight, I am sure.

Of course, what I didn't realize at the time was that I really had no power to begin with. You see, I had always managed a group of knowledge workers—folks who grew up crunching numbers, writing complex code, creatively banging out products that at their roots consisted of only 1s and 0s. I truly believe that up until learning to lead, these knowledge workers merely tolerated me. I had never really managed them. They managed me by deciding to make me happy by filling out their timesheets. They humored me when I asked to be walked through the testing phase of the project plan, *again.* They certainly knew way more about how stuff really worked than I did. My life was ruled by impossible project plans (see Figure I-1). For a few

months straight I made great overtime by staying late at the office to perfect the Gantt chart, knowing in my heart that it would be out of date the very next day, if not the very next minute. Often, I was asked to "create a dashboard" for the executives: a report that I knew reflected a false, positive reality. Now that I look back, I wonder how I survived the "manager" title.

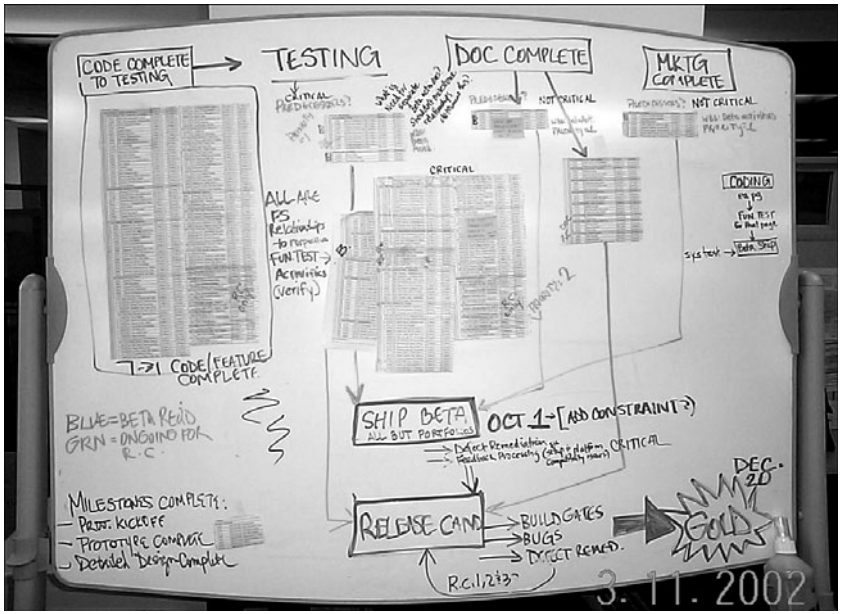


Figure I-1
Author's rendering of the "impossible project plan"

My first thoughts turned to tracking the status of projects. How will we know "where we are"? How will we know how much value we've earned? (My CEO at the time had written a book on earned value management.) How will we manage scope? (I had produced a scope change management process for the department and had spent weeks perfecting the diagram.) Most frightening of all, I wondered how insane our customers would think we are since we'd no longer be able to tell them when they could have everything they wanted. And what's with the paltry Scrum project tracking mechanisms? A burndown chart? What does that possibly tell us? That can't possibly tell us if we're on track! I want my percent-complete status reports! And let me say that the first few meetings with executives were disasters. I know that I left red-faced on many occasions.

All of these questions were fueled by the personal struggle I was going through: "Wow, if teams are self-managing, they'll no longer need me.

I don't have a place in this organization now that we're using Scrum." I had no idea how to act within this new realm. I had a very real struggle with getting past the "me" and focusing on the team. I was also troubled with ownership issues. I routinely struggled with not owning the administrative task of updating the product backlog; for me, this represented scope, and not having it within my charge was very frightening. I felt powerless and as if I had no role.

Somewhere around the third sprint, I started to get it. Once the teams started delivering real value that could be seen and touched, the light bulb went on. What were once yelling product owners were now engaged, energized product owners, who actually worked *with* the teams to talk about the user experience, helping developers deliver *valuable* product increments. Observing collocated team members who were often heard laughing, working closely together, and enjoying their personal lives again touched me in a way that no perfectly calculated project Gantt chart or nested work breakdown structure ever could. I began to realize what it meant for teams to work at a sustainable pace and to focus their energy on what really mattered: creating software for the company that they work for, while being able to enjoy their personal lives the rest of the time (after all, isn't this the foundation that keeps us all sane?). Coupled with a VP who "got it" and banned overtime for the department, the agile principle of sustainable pace really lifted morale and improved the quality of work life. I even had time one evening to visit the home of one of our developers, meet his wife, and learn more about real Indian food. It was a wonderful, personal experience (and I now love soan papdi, a wonderful Indian dessert).

After my personal light bulb went off about the value of agile development, I began to realize how I could provide value as an agile project manager. First of all, I let go of the backlog, and it relinquished its grip on me. By doing so, I gave control to someone else, namely the product owner, and let him prioritize the list. This gave me more time to focus on building teams. I moved into the collocated space with one of my teams, and I worked on justifying budget for other teams to collocate (and succeeded!). I created a newsletter for all of the project teams, called the *Daily Collaborator,* that included photos, stories, and interesting facts about the project. I learned how to report to executives, which was no small feat, by understanding their needs and by asking the team to help me determine how to show the project data. I made sure that stakeholders were involved in product reviews; sometimes it was difficult to get their time. I involved people from training and

support in our iteration reviews and garnered their support in the testing lab when we were manually testing part of the system. I helped set up product backlog meetings that replaced our traditional change control meetings. I worked with customers as they implemented early releases of our products to gather feedback and understand how we could improve their experiences. And when I was in a period of quietness, I observed, observed, and observed some more—in the team rooms, daily standup meetings, reviews, and general team interactions. These observations helped me determine which obstacles to tackle next; I kept detailed notes and added tasks to my own impediment backlog when I saw a change or an organizational impediment that needed attention. I was a chameleon and a peacock at the same time, retaining the ability to blend in with the environment, while standing out and displaying my feathers when the environment needed to change.

We celebrated a very successful release nine months after instituting Scrum. It was a proud moment for us all; we had each traveled a personal journey and transformation unlike any other. Our release t-shirts said "Develop with Heart; Deliver with Pride." That department of 85 people always will remain my fondest memory of a truly performing Scrum development organization.

My first three Scrum teams—the ones that truly scared the bejeezus out of me—will forever remain in my heart as the kind people who taught me the tough lessons of letting go.

The best day of my professional life was the day that I walked into one of my Scrum team's daily meetings and the team looked at me, smiling, and said, "We don't need you here, Stacia. Maybe you can use this time to work on other things or to help another team. We've got it under control." And you know, they did have it under control. I walked away on the verge of tears, but the tears weren't for me and my "loss"; they were from the happiness I felt at being able to let go and know that all would be just fine, and from the satisfaction I felt from helping individuals become empowered.

For me to cross the agility bridge, I had to understand what it meant to put others before me. This wasn't something that came naturally to me; because of a tough upbringing and lack of sense of self, I created a strong identity in my project manager title. I had to learn how to facilitate and listen for problems underneath the surface. Most importantly, I had to learn that the people doing the work know the work the best and will figure out the best way to get from point A to Z. All they really needed me for was to clear the path. They knew this already; Scrum helped me see it.

Whereas Michele got it right away, it took me awhile. We each came from very different places when embarking on our own personal bridges to agility. Michele's bridge was short and level; mine was a swaying suspension bridge, on a 45-degree angle, fraught with high winds and torrential downpours. What we both agree on is that since we've been helping teams—hundreds of teams—move to agile methods, we have never been happier in our professional careers. In the following chapters, we are pleased to present some ideas for translating what you already know about managing projects into your own agile paradigm. We'll dig deeper into what you should expect, how to successfully make the transition, and what steps you'll need to take in order to cross the bridge to agility.

Chapter 5
Scope Management

Project Scope Management includes the processes required to ensure that the project includes all the work required, and only the work required, to complete the project successfully.

—PMBOK® Guide

It is not the strongest of the species that survive, nor the most intelligent, but the ones most responsive to change.

—*Charles Darwin,* The Origin of Species

Next week there can't be any crisis. My schedule is already full.

—*Henry Kissinger*

"Scope creep" has always been the bane of traditional project managers, as requirements continue to change in response to customer business needs, changes in the industry, changes in technology, and things that were learned during the development process. Scope planning, scope definition, scope verification, and scope control are all processes that are defined in the *PMBOK® Guide* to prevent scope creep, and these areas earn great attention from project managers. Those who use agile methods believe these deserve great attention as well, but their philosophy on managing scope is completely different. Plan-driven approaches work hard to prevent changes in scope, whereas agile approaches expect and embrace scope change. The agile strategy is to fix resources and schedule, and then work to implement the highest value features as defined by the customer. Thus, the scope

remains flexible. This is in contrast to a typical waterfall approach, as shown in Figure 5-1, where features (scope) are first defined in detail, driving the cost and schedule estimates. Agile has simply flipped the triangle.

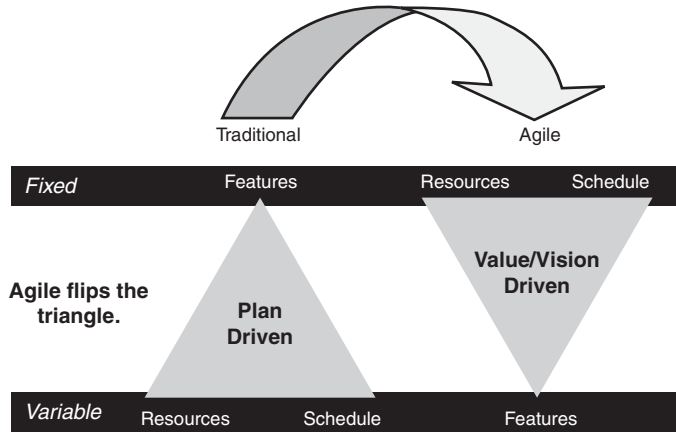


Figure 5-1
Waterfall vs. Agile: The paradigm shift (original concept courtesy of the DSDM Consortium)

Scope Planning

The *PMBOK® Guide* defines the Project Scope Management Plan as the output of the scope planning process.[1] This document defines the processes that will be followed in defining scope, documenting scope, verifying and accepting scope and completed deliverables, and controlling and managing requests for changes to the scope. In agile, the iterative and incremental process itself is what manages scope. Unless documentation is required for auditing purposes, no additional document outlining procedures for scope management is needed. Scope is defined and redefined constantly in agile, as part of the planning meetings—in particular, release planning and iteration planning—and by the management of the product backlog. Remember, resources and time are typically fixed in agile approaches, and it's the scope that is allowed to change. However, when fixed-scope projects are required, it is the number of iterations that will change, in order to accommodate the need for a full feature set prior to release. Additionally, one of the success criteria in traditional projects is the extent to which we can "stick to the scope"; in agile, it is more important to be able to efficiently and effectively respond to change. The success criteria in agile thus changes to "Are we providing value to our customer?" The primary measure of progress is working code.

Table 5-1 provides a summary comparison of scope planning from the traditional and agile perspectives. In agile projects, scope planning is referred to as "managing the product backlog."

Table 5-1
Scope Planning

Traditional	Agile
Prepare a Project Scope Management Plan document.	Commit to following the framework as outlined in the chosen agile process.

Scope Definition

The *PMBOK® Guide* practices of scope definition, work breakdown structure (WBS) creation, and scope verification occur iteratively in agile. A traditional WBS for software projects is usually divided at its highest level into phases of analysis, design, coding, testing, and deployment activities. Each of these phases is then decomposed into tasks or groups of tasks, referred to as work packages in the *PMBOK® Guide*. Traditional project planning begins top-down and relies on the elaboration of detailed tasks with estimates and dependencies to drive the project schedule via use of critical path analysis. Even though the *PMBOK® Guide* goes into great detail about scope decomposition by way of WBS (work breakdown structure), it also warns that "excessive decomposition can lead to nonproductive management effort, inefficient use of resources, and decreased efficiency in performing the work."[2]

In agile, we approach these practices differently in that we define features at a high level in the product backlog and then place features into iterations during release planning. One can think of the iteration—or even the feature itself—as the agile equivalent of work packages. The features are estimated at a gross level in the product backlog—no detailed tasks or resources are defined at this point in time. Once the iteration begins, the features slated for that iteration—and only that iteration—are then elaborated into tasks that represent a development plan for the feature. Think of it as just-in-time elaboration, preventing a wasteful buildup of requirements inventory that may never be processed. The *PMBOK® Guide* supports this idea of "rolling wave planning":[3] As the work is decomposed to lower levels

of detail, the ability to plan, manage, and control the work is enhanced because the short timeframe of the iteration reduces the amount of detail and the complexity of estimating. The agile approach assumes that because things change so often, you shouldn't spend the time doing "excessive decomposition" until you're ready to do the work.

Let's look at how scope is defined throughout an agile project by examining five levels of planning common to most agile projects: the product vision, the product roadmap, the release plan, the iteration plan, and the daily plan.[4]

Product Vision

At the outset of a project, it is typical to hold a kickoff meeting. Agile is no different; however, the way the agile vision meeting is conducted is unlike what a traditional project manager might be accustomed to. Although the vision is defined and presented by the customer or business representative, it is the team that clarifies the vision during the discussions and subsequent exercises. Therefore, the team is heavily involved, and group exercises are a big part of determining the final outcomes. See Chapter 4, "Integration Management," for more detail on vision meetings.

The vision meeting is designed to present the big picture, get all team members on the same page, and ensure a clear understanding of what it is that they've been brought together to do. The vision defines the mission of the project team and the boundaries within which they will work to achieve the desired results. The project's goal should be directly traceable to a corporate strategic objective.

Here the scope is defined at a very high level. It is not uncommon to leave the vision meeting with only a dozen or so features identified, such as "provide online order capabilities," "enable international ordering and delivery," "create data warehouse of customer orders to use for marketing purposes," and "integrate with our current brick-and-mortar inventory system." Clearly these are all very large pieces of functionality with little-to-no detail—and this is what is appropriate at this stage of the project. The farther away the delivery date, the broader the stroke given to feature details.

Product Roadmap

A product roadmap shows how the product will evolve over the next three to four releases or some period of calendar time, typically quarters. The

product roadmap is a high-level representation of what features or themes are to be delivered in each release, the customer targeted, the architecture needed to support the features, and the business value the release is expected to meet. The customer or product manager, agile project manager, architect, and executive management should meet on average two to three times a year to collaborate on the development and revision of the product roadmap. Figure 5-2 shows a sample roadmap template made popular by Luke Hohmann in his book *Beyond Software Architecture.*[5]

> **Note**
>
> In agile, the word "release" does not solely mean a product release to the end customer—it can also mean an internal release to fulfill integration milestones and continue to confirm that the product is "potentially shippable."

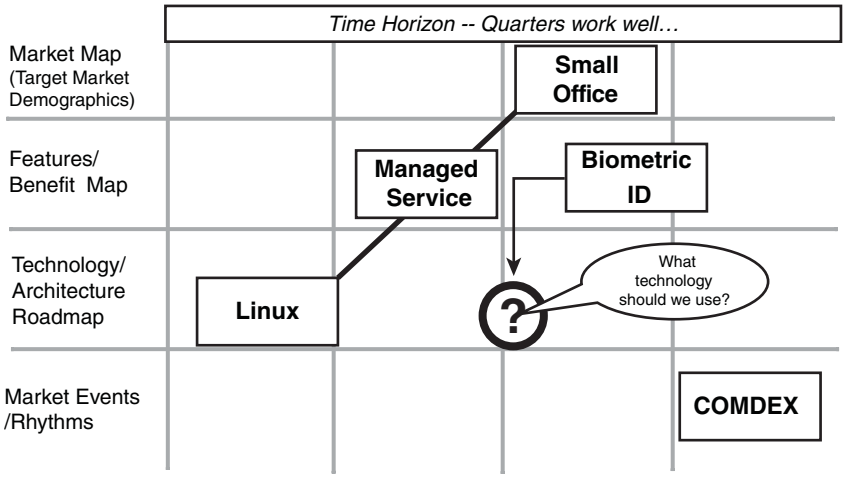


Figure 5-2
Product roadmap template, courtesy of Enthiosys and Luke Hohmann, from his book *Beyond Software Architecture*

Because the customer is responsible for maintaining and prioritizing the backlog of work, the customer also owns the product roadmap. In large corporations or on projects with multiple customers or product owners, the customer assigned to the project will often first work with others in his business unit to create a roadmap straw man as part of working out the priorities of deliverables with the business. Then this straw man is presented to key project team members (agile project manager, architect, and so on) for further revision. Finally, the roadmap is presented to the entire team and interested stakeholders, usually as part of the vision meeting and/or release

planning meeting. Feedback is encouraged at all sessions because it helps to better define a reasonable approach to product deliverables.

In addition to the vision plan and product roadmap, the end result of the product vision and product roadmap discussions should be the prioritized product backlog. These are all inputs into the next level of planning: release (or quarterly) planning.

Release (or Quarterly) Planning

In a release planning meeting, the team reviews the strategies and vision shared by the customer and determines how to map the work from the prioritized backlog into the iterations that make up a release or that make up a period of time such as a quarter. Figure 5-3 shows a typical release plan agenda, and Figure 5-4 shows the release plan done using a whiteboard and sticky notes, as is common in agile meetings when the team is co-located. The release plan is divided up into iterations (usually one flipchart page per iteration), with associated high-level features. The release plan also includes any assumptions, dependencies, constraints, decisions made, concerns, risks, or other issues that may affect the release. Again, documentation of these additional items can be as simple as posting the flipchart that they were originally recorded on or taking a picture of it and posting it on a shared website.

Last Responsible Moment Decision Points

Note that one of the items on the release planning meeting agenda is the identification of "Last Responsible Moment (LRM) decision points." LRM decision points identify points in the release where a decision must be made on an issue so as not to allow a default decision to occur. In other words, they identify "the moment at which failing to make a decision eliminates an important alternative".[6] Up until this point, the team can continue its momentum and gather additional information that will help in the decision-making. For example, one team knew it would have to make a decision between going with a Sybase database and an Oracle database. But the team did not have to decide this before they could start on the project—indeed, the team realized that it could develop code that was database-independent until the third iteration, when integration and reporting were required. Therefore, the team set the end of the second iteration as its LRM on the database decision, giving the architect and the DBA time to experiment with the work being developed concurrently.

Release Planning Meeting Agenda

- *Introductions, ground rules, review of purpose and agenda (Project manager)*
- *Do we need to review our current situation and/or existing product roadmap? (Project manager, architect, customer/product owner)*
- *Do we remember the product vision? Has it changed? (Customer/product owner)*
- *What is the release date? How many iterations make up this release? (Project Manager)*
- *What is the theme for this release? (Customer/product owner)*
- *What are the features we need for this release? (Customer/product owner)*
- *What assumptions are we making? What constraints are we dealing with? (Team)*
- *What are the milestones/deliverables expected? Do we have any LRM decision points? (Team)*
- *What is the capacity of the team (iteration velocity)? (Team)*
- *Can we move the features into the iterations? Do we need to break them into smaller features so that they can be completed in a single iteration? (Team)*
- *What issues/concerns do we have? (Team)*
- *Can we commit to this release as a team, given what we know today? (Team)*
- *Close: empty parking lot, action items, next steps (Project manager)*

Figure 5-3
Release planning meeting agenda

Figure 5-4
Release plan

Coordinated Release Planning

A colleague of ours once ran a release planning meeting with teams located in the U.S. and in London. Because of the size of the team and the budget constraints, not everyone could attend the day-long event. So the meeting was broken out into three days. Day 1 was focused on the U.S. team's release plan and all its assumptions about and dependencies on the London team. Due to time zone issues, the London team listened in on the phone for the first part of the meeting as the vision and the high-level detail and expectations around the features were discussed, then dropped off the call once the U.S. team started on the work of moving the features into the iterations. On Day 2, the London team did its work of moving the features into the iterations after reviewing the results of the U.S. team's release plan (photos and notes were made available on their shared wiki). At the end of Day 2, the London team posted its release plan. Day 3 was devoted to the coordination of the two plans, making sure all assumptions had been addressed and understood, all dependencies accounted for, and proper prioritizations had been made reflecting the teams' constraints. Both groups committed to the release plan on the third day after some final tweaking.

Teams that are not co-located should make every effort to bring everyone together for this meeting. Agile emphasizes face-to-face communication because of its benefits. However, balancing this with the realities of geographically dispersed teams means that budget constraints force teams to be selective about when they can gather together as a group. The vision and release planning meetings should receive high priority, because the information shared and decisions made in these meetings guide the team throughout the remainder of the release.

Iteration Planning

Traditional scope definition and many of the practices defined in the *PMBOK® Guide* knowledge area of Project Time Management are done as part of iteration planning. Here, features are elaborated (creating the equivalent of *PMBOK® Guide* work packages), tasks are identified, and the time needed to accomplish the tasks is estimated (see Figures 5-5 and 5-8). At the beginning of each iteration, the team should hold an iteration planning meeting to conduct this work. The team reviews the release plan and the prioritized items in the backlog, reviews the features requested for the current

iteration, and tasks out and estimates those features. See Figure 5-6 for a typical iteration planning meeting agenda. In keeping with the agile practice of just-in-time design, it is here that the details of the features are discussed and negotiated.

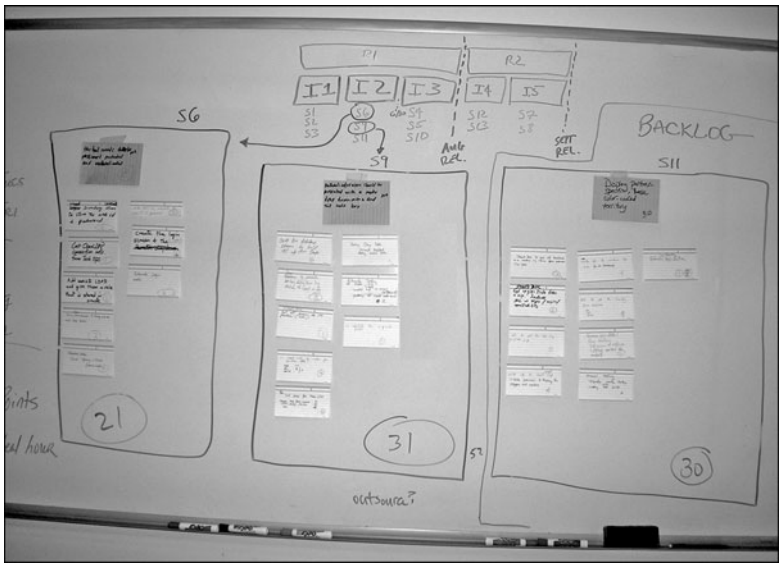


Figure 5-5
Iteration plan

Iteration Planning Meeting Agenda

- *Introductions, ground rules, review of purpose and agenda (Project manager)*
- *Do we know our iteration start and end dates? (Project manager)*
- *Do we know the team's velocity? (Team)*
- *Do we know what "done" means? (Team)*
- *What are the features we need for this iteration? What is the acceptance criteria for each feature? (Customer/product owner)*
- *Do we have enough information about the features so that we can task them out? (Team)*
- *Can we estimate the time it takes to complete the tasks? (Team)*
- *What assumptions are we making? What constraints are we dealing with? Are there dependencies that affect our prioritization? (Team)*
- *Are we within our velocity limits? (Team)*
- *What issues/concerns do we have? (Team)*
- *Can we commit to this iteration as a team, given what we know today? (Team)*
- *Close: empty parking lot, action items, next steps (Project manager)*

Figure 5-6
Iteration planning meeting agenda

Again, planning and design work is done only for the pieces that are being readied to code in that iteration, not for the entire system. It's often discovered during iteration planning that the sum of the task efforts exceeds the size of the iteration timebox. When this occurs, some of the work needs to be shifted either into the next iteration or back into the backlog. Similarly, if a team discovers that it has chosen too little work for the iteration, it will consult with the customer, who can then give the team an additional feature or two to make up the difference. This allows the team to make a realistic commitment to the scope of the work being defined.

Daily Stand-Up

One of the key heartbeats of agile development involves the practice of daily stand-up meetings. It is just what it sounds like: a daily meeting, where all team members attend, and while remaining standing, they each relate their status to the other team members and their plan for the day based on the progress that they've made. Standing helps keep the meetings short—stand-ups should run only 5 to 15 minutes. Its primary purpose is for the team members to inspect and adapt its work plan (iteration backlog) by quickly sharing information about the progress (or lack of) being made by each individual regarding the tasks that were committed to during the iteration planning meeting. These stand-ups help the team to remain focused on the agreed-to scope and goals of the iteration.

Summary Comparison

Table 5-2 provides a summary comparison of traditional and agile approaches to scope definition. In agile projects this is called "multilevel planning."

Table 5-2
Scope Definition

Traditional	Agile
Prepare a Project Scope Statement document that includes items such as the following: Project boundaries and objectives, product scope description…	Conduct a vision meeting to share the product vision; confirm and clarify the boundaries, objectives, and product scope description using exercises such as the elevator statement and design the box.

Traditional	Agile
And major milestones and project deliverables…	Conduct a planning meeting to prepare the product roadmap, as well as release or quarterly planning meetings that also include milestones and deliverables at an iteration level.
And product specifications and acceptance criteria…	Conduct an iteration planning meeting that results in the detail around each feature, and the tasks needed to complete the feature according to the team's definition of "done" and the acceptance criteria defined by the customer.
And assumptions and constraints.	All planning meetings identify and/or review assumptions and constraints.

Create a WBS

Agile teams do not tend to create formal WBSs (work breakdown structures). Instead, flipcharts and whiteboards are used to capture the breakdown of work. You've seen examples of these in Figures 5-4 and 5-5. So at the end of release planning, the agile equivalent of a WBS—a feature breakdown structure—would look like the sample release plan feature breakdown structure in Figure 5-7. If having iterations as work packages is not sufficient for your organization/billing needs, then breaking the work down further into smaller work packages would look like the results of an iteration planning meeting, as illustrated in Figure 5-8.

Table 5-3 compares the traditional and agile approaches to work breakdown. In agile projects, the work breakdown structure is captured in the release plan and the iteration plan.

Table 5-3
WBS Creation

Traditional	Agile
Create a work breakdown structure diagram.	Conduct planning meetings and give the team the responsibility for breaking down the work into smaller work packages (features and tasks), displayed as the release plan at the high level, and the iteration plan at the more detailed level.

Figure 5-7
Release plan feature breakdown structure

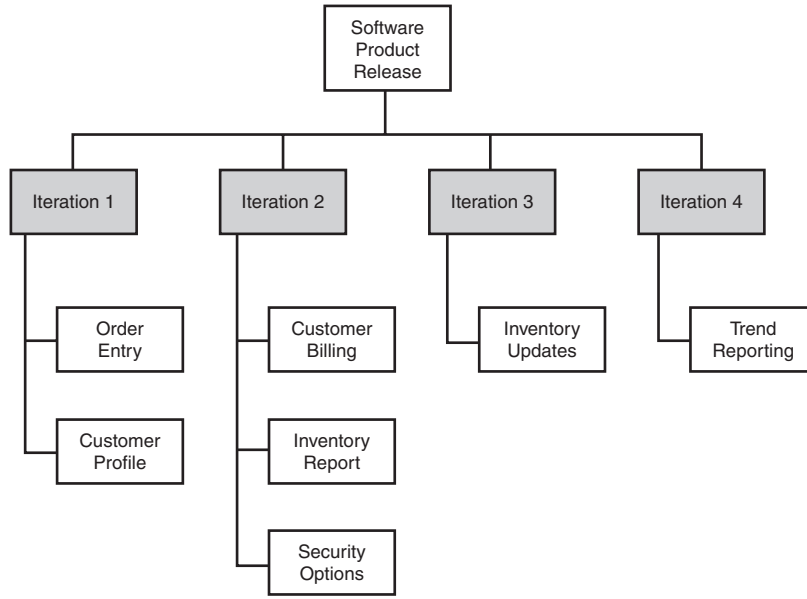
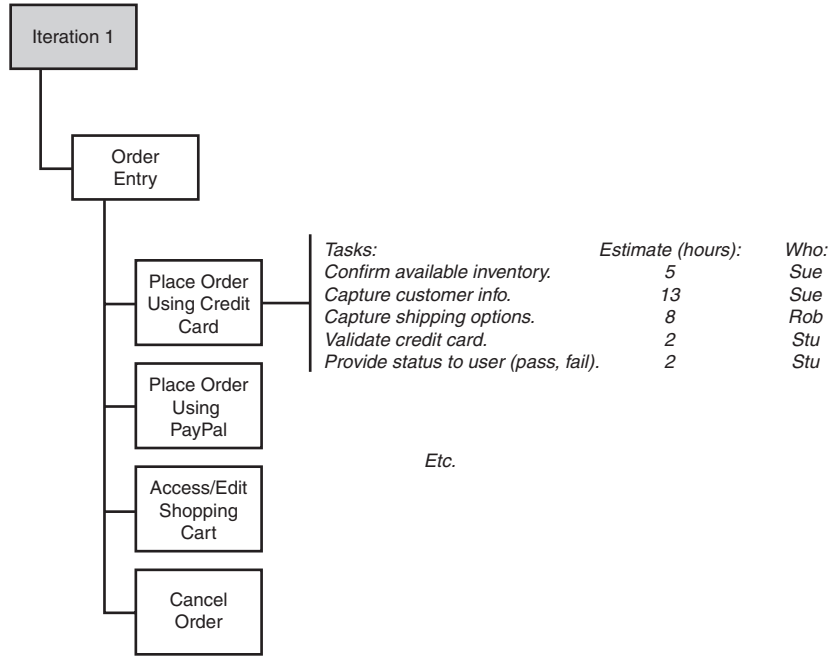


Figure 5-8
Iteration plan (partial)



Scope Verification

Scope verification is accomplished within the iteration, as the customer gets to review, test, and accept the implemented features. Ideally this happens throughout the iteration, but it can also happen at the end of the iteration, during the demo of the working code. Those features that were not accepted (either because they weren't ready or weren't right) move back into the backlog or into the next iteration at the discretion of the customer. Scope change control is handled by the management of this backlog, as discussed in the previous chapter on integration.

Table 5-4 makes the comparison between the traditional and agile approaches to scope verification. Scope verification is captured by the agile practices of acceptance testing and customer acceptance.

Table 5-4
Scope Verification

Traditional	Agile
Document those completed deliverables that have been accepted and those that have not been accepted, along with the reason.	Documentation of accepted features may be done informally (by moving the sticky notes to the "done" pile) or formally.
Document change requests.	Customer updates the backlog.

Scope Control

Controlling scope in agile projects consists of two things: managing the product backlog and protecting the iteration. Whereas the customer maintains the backlog, it is the agile project manager who protects the team and helps prevent scope changes from occurring during the iteration.

When a team commits to the iteration at the end of the iteration planning meeting, the delivery team is effectively saying, "Given what we know today, we believe we can deliver this work using our definition of 'done' within this iteration," and the customer is effectively saying, "Given what I

know today, this is the work that I am expecting by the end of the iteration, and during that time I will not mess with the iteration backlog" (that is, scope). The iteration backlog is thus locked in.

It is important to set the length of your iteration accordingly, because the customer must wait until the next iteration to make changes. If there happens to be lots of "requirements churn" (that is, requests for changes are coming in very frequently), you may want to discuss shorter iteration cycles with the team in order to enable more frequent changes. Maintenance teams may have iteration lengths of only one week, whereas larger system developments with known requirements may have an iteration length of four to six weeks. If the customer keeps trying to interrupt the team with changes, the iteration length may be too long.

There will always be exceptions, and in those cases a discussion between the customer and the agile project manager should help identify potential resolutions. Iterations can be aborted and restarted, but this should be the rare exception.

Given the short duration of iterations, it is easy to protect the iteration backlog from change. However, changes in the product roadmap and the release plan are expected and therefore should be reviewed regularly.

Table 5-5 lists out the differences between the traditional and agile approaches to scope control. Agile users refer to scope control as "managing the product backlog."

Table 5-5
Scope Control

Traditional	Agile
Use a change control system to manage change.	The customer manages the product backlog; once the team commits to the work to be done in an iteration, the scope is protected for that duration.
Update all documents as appropriate with the approved changes.	The team revisits release plans and product roadmaps regularly, making changes as needed to better reflect the team's progress and changes requested by the customer.

Summary

The main points of this chapter can be summarized as follows:

- "Scope creep" doesn't exist in agile projects, because scope is expected to change.
- Scope management in agile is primarily a function of "rolling wave" planning and the management of the product backlog.
- Scope is defined and redefined using five different levels of planning that take the team from the broad vision down to what team members plan to complete today.
- WBSs are not created per se; instead, release/quarterly plans and iteration plans serve to break down the work into smaller work packages, referred to as "features and tasks."
- Scope is verified by the customer, who is responsible for accepting or rejecting the features completed each iteration.
- Scope is controlled through the use of the backlog, rolling wave planning, and the protection of the iteration.

Table 5-6 presents the differences in project management behavior regarding scope management in traditional and agile projects.

Table 5-6
Agile Project Manager's Change List for Scope Management

I used to do this:	Now I do this:
Prepare a formal Project Scope Management plan.	Make sure the team understands the framework and process structure of the chosen agile approach.
Prepare a formal Project Scope Statement document.	Facilitate planning meetings—vision, release, iteration, daily stand-up—and arrange for the informally documented plans to be highly visible to all stakeholders.
Create the WBS.	Facilitate the release planning meeting so that the team can create the plan showing the breakdown of work across several iterations.

(continued)

Table 5-6
Agile Project Manager's Change List for Scope Management *(continued)*

I used to do this:	Now I do this:
Manage the change control system and try to prevent scope creep.	Step away from the backlog; it is owned by the customer. If needed, remind the customer that during the iteration, the team is protected from scope changes.
Manage the delivery of tasks to prevent or correct scope creep at the task level.	Allow team members to manage their daily tasks and facilitate conversations with the customer to avoid unnecessary work or "gold plating."

Endnotes

1. *PMBOK® Guide*, 107.
2. Ibid, 114.
3. Ibid.
4. Mike Cohn. *Agile Estimating and Planning* (Upper Saddle River, NJ: Pearson Education, Inc., 2006), 28.
5. Luke Hohmann. *Beyond Software Architecture* (Boston: Addison-Wesley, 2003), 287.
6. Poppendieck. *Lean Software Development*, 57.

Index

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U-V

W-Z-Y-Z