The Addison-Wesley Signature Series

A Mike Cohn Signature Book

Succeeding with Agile

Software Development Using Scrum

Mike Cohn

Foreword by Tim Lister

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States, please contact

International Sales
international@pearson.com

Visit us on the Web: www.informit.com/aw

The Library of Congress Cataloging-in-Publication data is on file.

Copyright © 2010 Mike Cohn

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc.
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax (617) 671-3447

ISBN-13: 978-0-321-57936-2
ISBN-10: 0-321-57936-4
Text printed in the United States on recycled paper at Edwards Brothers in
Ann Arbor, Michigan.
Second printing January 2010

Editor-in-Chief
Karen Gettman

Executive Editor
Chris Guzikowski

Senior Development Editor
Chris Zahn

Managing Editor
Kristy Hart

Project Editor
Jovana San Nicolas-Shirley

Copy Editor
San Dee Phillips

Indexer
Lisa Stumpf

Proofreader
Karen Gill

Publishing Coordinator
Raina Chrobak

Cover Designer
Alan Clements

Compositors
Jake McFarland
Bumpy Design

Foreword

All the time I hear people talking about software projects as journeys, and I think they are implying that software projects are not just journeys, but they are journeys into the unknown. We start with funding from a sponsor, muster together a stout-hearted crew, head out in what we guess might be a useful direction, and the rest is The Odyssey. We live the tales of the brave Odysseus: tales of Lotus Eaters, the Cyclops, Circe, the Sirens, Scylla, and Calypso. We succeed or fail only with the help or rage of the gods. How wonderfully romantic, and how perfectly silly.

I think that the more appropriate analogy along this line is the project as an expedition. We have a goal or a short list of goals. We have some well-proven maps; we have some vaguer ones, too. We have the advice and journals from those who have been out there and made it back to tell their stories.

We don't walk out the door and face the unknown; but on the other hand, there are some big question marks, and these bring us into a high-risk position. We accept these risks, because if the expedition can succeed there are surely significant rewards. We have skills, but there are uncertainties.

How do we deal with this? I recommend that we look back, oh, about 300 years, to the York Factory on Hudson Bay in Canada. At that time this was the headquarters of the Hudson Bay Company. The Hudson Bay Company's main line of business was to be the supplier of all necessary provisions for fur traders going out on, you guessed it, expeditions, from Hudson Bay. The fur traders developed a great way to start an expedition, and it was called "The Hudson Bay Start." Having done their one-stop shopping at The Company, the fur traders would go out of Hudson Bay only a mile or two and set up camp. Why? Certainly not to set up traps; they wanted to discover what they forgot to bring while they were less than an hour's hike back into town! Being the excellent project person that you are, you know that for the vast majority of time the leather-faced expert fur trader would reappear for another shopping trip.

What the heck does all this have to do with the book in your hands right now? With *Succeeding with Agile,* Mike Cohn has delivered The Hudson Bay Start for agile development. This is it. This is a weather-beaten experienced fur trapper giving you *the* checklist to work through before you begin your expedition. By reading this book, you will find that Mike brings up issues that you never thought of, offers advice on how you might handle situations, and helps you define new roles on your team.

Don't be the only person on your team to read this book; with self-organizing teams anyone can be expedition leader at any given time. This book is going to lead to many very interesting discussions; I guarantee it.

I worry a bit that I am saying that Mike has handed you a book without choices for you. He points out early and often that you must make your choices on individual, team, and organizational issues.

Succeeding with Agile is not about having a single successful project; it is about how agility can transform an organization. I guess in Hudson Bay terms, it's about how to have a great career as Voyageurs.

If you have any lingering doubts about Mike as an experienced expedition leader, notice that his company is Mountain Goat Software.

Tim Lister
Principal, The Atlantic Systems Guild, Inc.
New York City

Introduction

This is not a book for those who are completely new to Scrum or agile. There are other books, classes, and even websites for that. If you are completely new to Scrum, start with one of those.[1] Nor is this a book for purists. They can find many blogs that will argue the one, true way of agile or Scrum. This is a book for pragmatists. It is for those who have started with Scrum and then encountered problems or for those who have not yet started with Scrum but who know they want to. They don't need to read again about how to draw a burndown chart or what three answers each person gives at the daily scrum. They need advice on the harder stuff—how to introduce and spread Scrum, how to get people to let go of doing a big design at the start of the project, how to deliver software that works by the end of each sprint, what managers do, and more. If these concerns sound familiar, this is a book for you.

To answer these questions, this book draws on my experience with Scrum over the past 15 years, but especially over the last 4. For the last 4 years, every evening after I spent the day with one of my clients, I would go back to my hotel room and make notes about the problems they were facing, the questions they asked, and the advice I gave. I then followed up, either with return visits or e-mails. I wanted to know for sure what advice was working to solve which problems.

As I collected the questions, problems, and advice, I was able to look for common themes. Some obstacles were completely unique to one client or one team. Others were more prevalent and repeated across many teams and organizations. It is these more universal problems—and my advice on overcoming them—that form the basis of this book. This advice is particularly evident in two ways: First, most chapters include boxes labeled *Things to Try Now.* These re-create the advice I found myself giving most often or that was most helpful in particular situations. Second, most chapters also include boxes labeled *Objection*. I have tried in these boxes to reproduce a typical conversation in which someone disagreed with the point I was making at the time. As you read these objections, try to hear the voice of some of your coworkers. I suspect you have heard many of the same objections. In these boxes, you will see how I've sought to overcome them.

---

[1] A good starting point is www.mountaingoatsoftware.com/scrum.

What Else I've Assumed About You

Beyond assuming that you understand the basics of Scrum and now want to either introduce it into your organization or get good at it, I assume that you have some influence within the organization. That doesn't mean I have aimed this book at directors, vice presidents, and the CEO. The type of influence I am assuming is just as likely to come from your personality and individual credibility with your coworkers as it is to come from whatever job title is on your business card. Sure, having a fancy title can help. But as we'll see, the type of influence needed to succeed with Scrum more often comes from opinion leaders.

How This Book Is Organized

When I began this book four years ago, my working subtitle was *Getting Started and Getting Good,* as those were the two things I really wanted to help with. In collecting anecdotes and giving advice, I realized that getting started and getting good at Scrum are the same thing. There are not separate techniques we apply to start and then different techniques we use to get good at it.

Part I is about getting started—it includes advice on whether to start small or convert everyone at once, how to help people move from being aware that a new process is needed to desiring change to having the ability to do it, and how to select initial projects and teams. You will use the basic mechanisms introduced in this section not only to get started but also to get good. Among these are the improvement communities and improvement backlogs of Chapter 4, "Iterating Toward Agility."

In Part II, I focus on individuals and the changes each needs to make as part of the process of adopting Scrum. Chapter 6, "Overcoming Resistance," describes the type of resistance some individuals may exhibit. In it, I offer advice for thinking about why someone is resistant and then provide guidance on how to help the person get past the resistance. Chapters 7 and 8 describe the new roles that exist on a Scrum project and the changes necessary in the traditional roles, such as programmer, tester, project manager, and so on. Chapter 9, "Technical Practices," describes some of the technical practices (continuous integration, pair programming, test-driven development, and so on) that should be used or at least experimented with and that can change much of how individuals approach their day-to-day work.

In Part III, we expand outward from individuals to teams. We look first at how to structure teams to best achieve the benefits of Scrum. Next, in Chapter 11, "Teamwork," I cover the nature of teamwork on a Scrum project. In Chapter 12, "Leading a Self-Organizing Team," we look at what it means to lead a self-organizing Scrum team. In that chapter, I provide specific advice for what ScrumMasters, functional managers, and other leaders can do to help a team self-organize for success. Chapters 13–15 round out Part Three with a discussion of sprints, planning, and quality.

Part IV expands our focus outward once more, this time to the organization. In Chapter 17, "Scaling Scrum," we take an extended look at what is necessary to scale Scrum up to work on large, multi-team projects. In Chapter 18, "Distributed Teams," we consider the additional complexities of distributed teams. Then, in Chapter 19, "Coexisting with Other Approaches," we add yet more complexity by discussing how to make Scrum work when part of the project uses a sequential process or when there are compliance or governance requirements. Part IV concludes with Chapter 20, "Human Resources, Facilities, and the PMO," focusing on special considerations of the impact of Scrum on an organization's human resources, facilities, and project management office groups.

Part V contains two chapters. Chapter 21, "Seeing How Far You've Come," summarizes various approaches to measuring how far an organization has progressed in becoming agile. Chapter 22, "You're Not Done Yet," concludes the book with the reminder that being agile requires continuous improvement. It doesn't matter how good you are today; to be agile you must be better next month.

A Note on Some Terms

As with most things, writing about Scrum is harder than talking about it. It is too easy to misinterpret a sentence or take one sentence out of context. To avoid these problems, I have tried to be careful and precise in my use of certain terms. I use the word *developer*, for example, to refer to anyone on the development side of the project. This includes programmers, testers, analysts, user experience designers, database administrators, and so on.

The word *team* poses its own challenges. It, of course, includes the developers, but does *team* include the ScrumMaster and product owner? Naturally, this depends on the context. When I have wanted to be especially clear, I use *whole team* to refer to everyone: developers, product owner, and ScrumMaster. However, slavish use of *whole team* would have reduced the readability of the book. So you will encounter *team* as well, but usually in places where the context makes it sufficiently clear which group I'm referring to.

In referring to Scrum and agile teams, I have also needed a term to refer to those teams that are neither. In various places, I have used *sequential, traditional,* and even *non-agile*. Each conveys a slightly different meaning and is used appropriately.

How to Use This Book

Many books have a heading like the one above this sentence. But those headings usually say How to *Read* This Book. The best way to read this book is to use it. Don't just read it. When you encounter a *Things to Try Now* section, try some of them. Or note them and try them at your next retrospective or planning meeting, if that is what I recommended.

It is not necessary to read the book in order. In fact, there could well be entire chapters you do not need to read. If in your organization's quest to become good at Scrum, you have no significant problems with planning and no distributed teams, then skip or skim those chapters. I do, however, recommend that everyone read at least the first four chapters and read them in order. They lay the foundation for much of what follows.

In Chapter 4 you will be introduced to the idea of improvement communities and improvement backlogs. An improvement community is a group of like-minded individuals who are passionate about driving improvements in a particular area. One improvement community could form when three people passionate about the product backlog decide to collect best practices and advice to share across teams. Another improvement community could include hundreds of people interested in improving how your organization tests its applications. An improvement backlog is exactly what it sounds like—a prioritized list of things that an improvement community would like to help the organization get better at.

One of my hopes is that improvement communities—including the Enterprise Transition Community that guides and energizes the transition effort—will use this book to load their improvement backlogs. In fact, many of the top-level section headings have been deliberately worded so that those headings can go right onto an improvement backlog. As examples, consider "Shift from Documents to Discussions" in Chapter 13, "Prepare in This Sprint for the Next" in Chapter 14, and "Automate at Different Levels" in Chapter 16.

As a long-time Scrum trainer and consultant, I have worked with hundreds of teams and organizations, and I've come to believe that success with Scrum is possible for every organization. Some will have a harder time than others. Some will be challenged by a rigid corporate culture. Others will confront entrenched, difficult personalities facing personal loss. The lucky ones will have supportive leadership and passionately engaged employees. What each of these organizations will have in common, though, is the need for pragmatic and proven advice. I have written this book with the hope of providing it.

Chapter 10

Team Structure

It is perhaps a myth, but an enduring one, that people and their pets resemble one another. The same has been said of products and the teams that build them.

> The system being produced will tend to have a structure that mirrors the structure of the group that is producing it, whether or not this was intended. One should take advantage of this fact and then deliberately design the group structure so as to achieve the desired system structure. (Conway 1968; commonly referred to as "Conway's Law")

If it is true that a product reflects the structure of the team that built it, then an important decision for any Scrum project is how to organize those individuals into teams. Factoring into this decision are considerations of team size, familiarity with the domain, the channels of communication, the technical design of the system, individual experience levels, the technologies involved, the newness of those technologies, where team members are located, competitive and market pressures, expectations about project schedule, and much more.

In this chapter we look at the importance of two critical factors to be considered when deciding how to structure Scrum teams: keeping teams small and orienting each team around the delivery of end-to-end user-visible functionality. We also look at the importance of having the right people on each team and not overloading those individuals by forcing them to split time among too many teams. We conclude the chapter with nine questions to ask when starting a multi-team project.

Feed Them Two Pizzas

I was working on a project for a bioinformatics company when the CEO asked me to provide her with an estimate of how long the project would take. The application was large, the domain complicated, and the team mostly new. Because the domain was so complicated, our team was made up of some very smart Ph.D. scientists, who knew only a little about programming, and some very smart programmers, most of whom had taken no more than a class or two in biology or genetics. No one on the team was great at both the science and the development.

After a bit of research and work with the team I returned to the CEO with an estimate of something like 100 person-years. In other words, if we used all 40 people on the team, we could finish the project in about two and a half years. I don't think that number was too shocking to her, but it was a big number, so she asked me, "What's the cheapest way we could write it?" My answer: "Take Steve, the scientist with the best understanding and aptitude for programming, and have him go spend 10 years working in a great software company doing nothing but learning how to be a great programmer. Then have him return to our company and spend 30 years working alone to write the program. It'll take 40 years, but it's your cheapest option." She should have been quite pleased with my answer—after all, I'd taken the 100 person-year initial estimate and offered her a way to cut it by more than half. Alas, 40 years was just a bit too long for her to wait.

As this story illustrates, a team offers the advantage of getting things done far more quickly than one person could, but with that advantage comes a potentially large amount of communication overhead. Knowing that, what is the ideal team size for Scrum projects? Generally accepted advice is that the ideal Scrum team size is five to nine individuals. While I agree with this, putting a number to it makes me nervous. If you're thinking about your ten-person team right now you may feel inclined to return this book, demand a refund, and give up on Scrum.

Don't.

Rather than take the five-to-nine person guideline too literally, I prefer how Amazon.com thinks its about its teams. Amazon refers to them as "two-pizza teams," meaning a team that can be fed with two pizzas (Deutschman 2007). As humorous as that is, it's actually useful. If ordering food for the occasional team lunch is a hassle, it could be a good indicator that the team has become too large.

The largest single Scrum team that I worked with where I was content to leave them alone was 14 people. The team, its ScrumMaster, and I had all looked at possible ways to split them up, but no solutions we came up with seemed better than leaving them intact. I've also worked with one team of 25 that insisted it should be one team rather than more. They were wrong; there was too much communication overhead on a single team of that size.

SEE ALSO

Scrum projects scale through the use of teams of teams. For information on large Scrum projects, see Chapter 17, "Scaling Scrum."

Why Two Pizzas Are Enough

To be fair, there are some advantages to large teams. Large teams may include members with more diverse skills, experiences, and approaches. Large teams are not as much at risk to the loss of a key person. They may also provide more opportunities for individuals to specialize in a technology or a subset of the application.

On the other hand, there are even more advantages to small teams. These include the following:

- **There is less social loafing.** Social loafing is the tendency for people to exert less effort when they believe there are others who will pick up the slack. Members of small teams are less prone to social loafing. Social loafing was first demonstrated by psychologist Max Ringelmann in the 1920s when he measured the pressure exerted by individuals and teams pulling on a rope. Groups of three exerted only two-and-a-half times (not three times) the average individual pressure. Groups of eight exhibited less than four times the individual average. Ringelmann's and related studies have shown that individual effort is inversely related to team size (Stangor 2004, 220).
- **Constructive interaction is more likely to occur on a small team.** Stephen Robbins, author of *Essentials of Organizational Behavior*, a best-selling textbook on organizational behavior, has concluded that teams of more than 10 to 12 people have a difficult time establishing feelings of trust, mutual accountability, and cohesiveness. Without these, constructive interaction is difficult (2005).
- **Less time is spent coordinating effort.** Small teams spend less time coordinating the efforts of team members. This is true both in the aggregate and as a percentage of total project time. As a simple example, we all know that the effort just to plan a meeting for a large team can be overwhelming.
- **No one can fade into the background.** With large teams, there is lower participation in group activities and discussions. Similarly, the disparity in the amount of participation among team members increases. The problems can prevent a group of individuals from jelling into a cohesive, high-performing team.
- **Small teams are more satisfying to their members.** With a small team, one person's contributions are more visible and meaningful. This is perhaps one reason why research has shown that participation on a large team is less satisfying to team members (Steiner 1972).
- **Harmful over-specialization is less likely to occur.** On a large project, individuals are more likely to take on distinct roles (Shaw 1960). For example, one developer chooses to work only on the user interface. This creates wasteful hand-offs of work between team members and reduces the amount of learning that occurs when individuals are more willing and likely to work beyond specific job roles.

SEE ALSO

The problems with hand-offs will be considered in Chapter 11, "Teamwork."

One interesting study of team size looked at 109 different teams. The small teams had 4 to 9 members while the large teams had 14 to 18. The researchers reached several conclusions.

> Members of smaller teams participated more actively on their team; were more committed to their team; were more aware of

> the goals of the team; were better acquainted with other team members' personalities, work roles, and communication styles; and reported higher levels of rapport. The data also show that larger teams are more conscientious in preparing meeting agendas compared to smaller teams. (Bradner, Mark, and Hertel 2003, 7)

Hmm. With a small team I can have many compelling advantages. Or I can staff a larger team and get better meeting agendas.

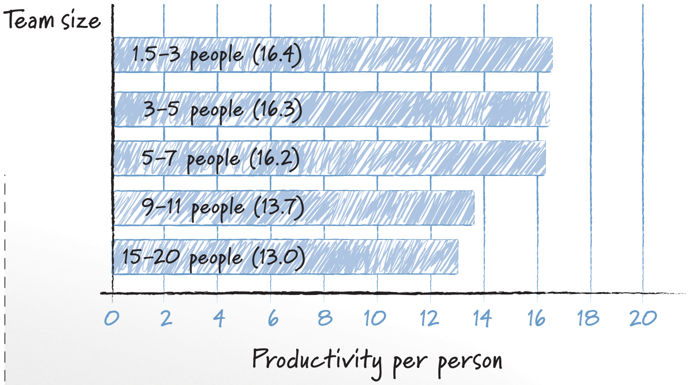
Small Team Productivity

Given the strength of these advantages to small teams, we would expect small teams to be more productive than large teams. Doug Putnam of QSM found exactly that after studying 491 projects with team sizes from 1 to 20 people. Since 1978 QSM has been collecting data on software productivity and estimates. The company maintains the software development industry's most thorough metrics database, including data on application size, effort, industry, and more. As such, the QSM database is uniquely valuable for comparing different types of projects.

From the QSM database of over 7,000 projects, Putnam narrowed the data set to 491 projects completed between 2003–2005 that delivered between 35,000 and 95,000 new or modified lines of source code.[1] Project sizes were evenly distributed from 1 to 20 team members. As shown in Figure 10.1, Putnam found that the smaller the team size, the more productive each team member was. However, the difference between teams sized from 1.5 to 7 people was very small.

FIGURE 10.1

The average productivity per person on teams of various sizes. Printed with permission from QSM, Inc. All rights reserved.



---

[1] Lines of code is, of course, a much maligned metric and deservedly so in many cases. However, in a database of this size, I believe it is a reasonable proxy for the size of a project and can therefore be used in productivity calculations.

Putnam looked also at the total development effort that goes into projects. Not surprisingly, he found that smaller teams complete projects with less total effort. Putnam concluded that "larger teams translate into more effort and cost. The trend appears to have an exponential behavior. The most cost-effective strategy is the smallest team; however the extreme nonlinear effort increase doesn't seem to kick in until the team size approaches nine or more people." These results can be seen in Figure 10.2.

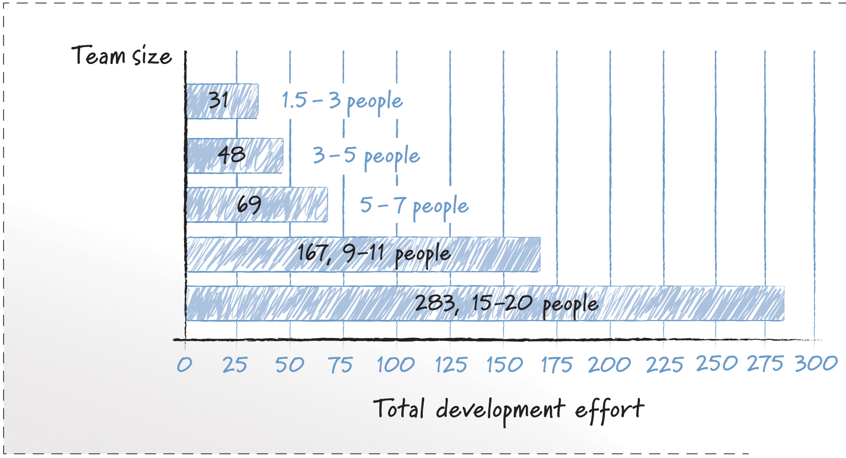


FIGURE 10.2

Smaller teams require less total effort to deliver the same size project. Printed with permission from QSM, Inc. All rights reserved.

In most cases, however, we are not concerned with minimizing the total development effort; schedule is always a major consideration. After all, we rarely have 40 years to wait for a lone developer to finish what we need by next spring. The impact of team size on overall schedule is shown in Figure 10.3. This figure shows that a 5- to 7-person team will complete an equivalently sized project in the shortest amount of time. Smaller teams took slightly longer. Notice again the dramatic increase with teams of 9 to 11 people.

An additional study described in the *Communications of the ACM* compared the productivity of large and small teams. Long-time industry veteran Phillip Armour writes of this research.

> Large teams (twenty-nine people) create around six times as many defects as small teams (three people) and obviously burn through a lot more money. Yet, the large team appears to produce about the same amount of output in only an average of twelve days less time. This is a truly astonishing finding, though it fits with my personal experience on projects over thirty-five years. (2006, 16)

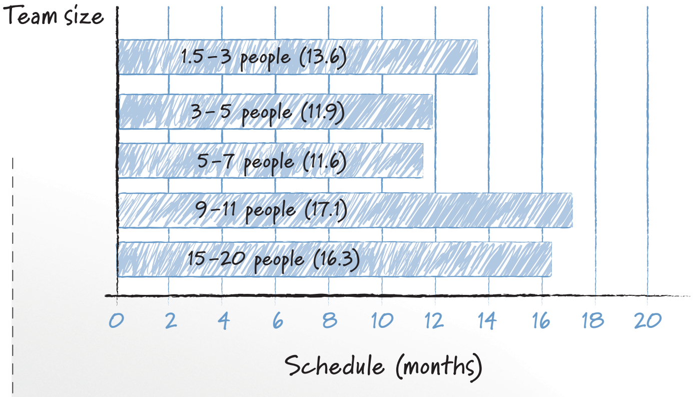
With all of the strong reasons in favor of small teams, I don't think I'll be placing any orders for three pizzas any time soon.

SEE ALSO

There are, of course, projects that cannot be done with a single two-pizza team. Scrum teams scale by having teams of teams rather than one immense team. For more on scaling, see Chapter 17.

FIGURE 10.3

Teams of five to seven people finished equivalently sized projects in the shortest amount of time. Printed with permission from QSM, Inc. All rights reserved.



OBJECTION

"There are too many disciplines on my project for us to have small teams. There are analysts, programmers, database developers, client-side programmers, middle-tier programmers, testers, test automation engineers, and more. I can't possibly have a five- to nine-person team."

Although a project may require work in that many disciplines, it almost certainly does not require a dedicated expert in each area. On a nine-person team with each person responsible solely for one discipline, it will be difficult or impossible to balance the workload of each team member. A team structure where some people may work only within one discipline but where others can move between two or more makes it much easier for the team to balance the workload of the different disciplines. Having at least some people work across disciplines also instills a better sense of whole-product responsibility rather than "I just do such-and-such."

THINGS TO TRY NOW

- ☐ If your team has nine or more people, try splitting into two teams after the current sprint. Work that way for at least two sprints before discussing whether it was better.
- ☐ For each team with five to nine people, consider splitting into two teams.

Favor Feature Teams

When I first began to consult for a certain California-based game studio, its teams were organized around the specific elements and objects that would exist in the video game it was developing. There was a separate team for each character. There were weapons teams, a vehicle team, and so on. This led to problems, such as weapons too weak to kill the monsters, colors too dark to show secret passages, and obstacles that frustrated even the most patient player.

On more traditional, corporate projects, we see equivalent problems when teams organize around the layers of an application. For example, a typical early-stage mistake for the project whose architecture is shown in Figure 10.4 would be to have four teams: a rich client team, a web client team, a middle-tier team, and a database team. Creating *component teams* such as these leads to a variety of problems including

- Reduced communication across the layers
- A feeling that design by contract is sufficient
- Ending sprints without a potentially shippable product increment

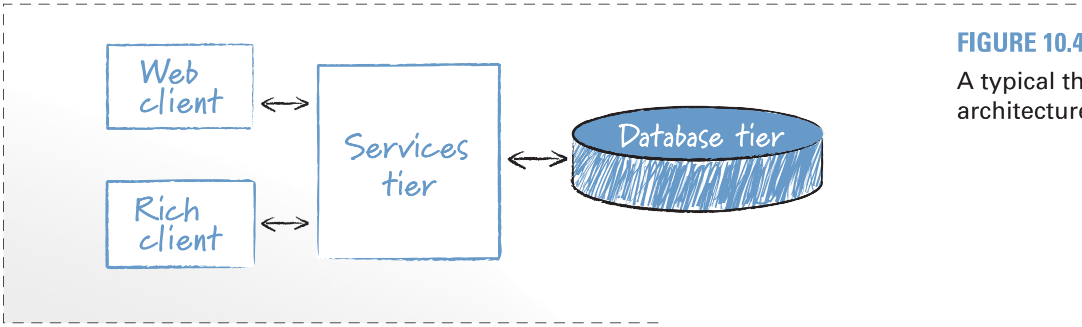


FIGURE 10.4

A typical three-tier architecture.

If structuring teams around the layers of an architecture is the wrong approach, what's better? Rather than organizing around components, each team on a project can ideally be responsible for end-to-end delivery of working (tested) features. A feature team working on the application shown in Figure 10.4 would, for example, work across all layers of the architecture. It might develop one feature that involves the database layer, the services tier, and the rich client user interface. In the same or next sprint, it would develop a feature going across the web client, services tier, and database tier.

SEE ALSO

The importance of delivering end-to-end functionality is discussed further in Chapter 14, "Sprints."

There are many advantages to organizing multiteam projects into feature teams:

- **Feature teams are better able to evaluate the impact of design decisions.** At the end of a sprint, a feature team will have built end-to-end functionality, traversing all levels of the technology stack of the application. This maximizes members' learning about the product design decisions they made (Do users like the functionality as developed?) and about technical design decisions (How well did this implementation approach work for us?).

SEE ALSO

More problems with hand-offs are described in Chapter 11.

- **Feature teams reduce waste created by hand-offs.** Handing work from one group or individual to another is wasteful. In the case of a component team, there is the risk that too much or too little functionality will have been developed, that the wrong functionality has been developed, that some of the functionality is no longer needed, and so on.
- **It ensures that the right people are talking.** Because a feature team includes all skills needed to go from idea to running, tested feature, it ensures that the individuals with those skills communicate at least daily.
- **Component teams create risk to the schedule.** The work of a component team is valuable only after it has been integrated into the product by a feature team. The effort to integrate the component team's work must be estimated by the feature team, whether it will occur in the same sprint during which it is developed (as is best) or in a later sprint. Estimating this type of effort is difficult because it requires the feature team to estimate the integration work without knowing the quality of the component.
- **It keeps the focus on delivering features.** It can be tempting for a team to fall back into its pre-Scrum habits. Organizing teams around the delivery of features, rather than around architectural elements or technologies, serves as a constant reminder of Scrum's focus on delivering features in each sprint.

OBJECTION

"My application is too complex; I can't possibly deliver end-to-end functionality in one sprint."

Learning how to identify small pieces of functionality is one of the first big hurdles for a new Scrum team. I remember my first Scrum project: Initially there were times we struggled to find anything we could deliver in less than six weeks. Looking back on that system many years later, I now see many ways we could have split that work. In fact, I see enough ways to split the work now that we could have done one-day sprints if we had wanted to.

As they gain experience, team members will find many more ways to split features while still delivering end-to-end functionality within each sprint. When doing so looks impossible, it is usually because teams are not structured appropriately. Before giving up, reconsider the individuals and skills on the team.

Use Component Teams Sparingly

Although you should strongly favor the use of feature teams, there will be occasions when creating a component team is appropriate. A component team, as I'm using the term here, is a team that develops software to be delivered to another team on the project rather than directly to users. Examples of component teams include a team developing an object-relational mapping layer between the application and the database or a reusable user interface widget team.

It is important that a component team still produce high-quality, tested, potentially shippable code by the end of each sprint. However, the new capabilities created by a component team are usually meaningless on their own. Think back for a moment to the examples I just gave. The object-relational mapping layer developed by one of the component teams is of interest to end users only through the context in which it is used by feature teams. But what about the team developing the reusable user interface widgets such as custom drop-down lists, data entry grids, and so on? These are certainly of interest to end users, right? Yes, but again only within the context of other features. An end user is not interested in a new data entry grid until it is embedded onto a page or screen.

Build Components Only As Feature Teams Ask for Them

Because the work of a component team is delivered to another team, it is those teams who usually act as the product owner for the component team. If your team needs deliverables from my team, then you will act as the product owner to my team. As such you will have all the responsibilities of a good product owner. At the start of a sprint, you will need to help prioritize what I work on. At the end of the sprint you will accept or reject it, providing feedback to me on what has been produced.

It will be hard for you to prioritize my work and provide feedback on it if my team is working far in advance of yours. Because of this, a component team should not develop new capabilities until one or more feature teams is ready for them. When a component team works far in advance of what feature teams need, it resorts to guessing at what capabilities are needed next. All too often this results in components or frameworks that are not usable by the feature teams. All new capabilities, including those built by component teams, should be developed within the context of externally visible functionality.

Rob was the senior developer on a component team developing an object-relational mapping layer that would be used by many of the 15 feature teams on the project. Rob's team was initially tasked with choosing between developing this technology in-house or using a commercial or open-source product. Members made the questionable decision to build it themselves. Anxious to prove the correctness of this decision, Rob and team tried aggressively to get ahead of the needs of the feature teams. Rather than working closely with one or more feature teams, Rob's component team made some big guesses about the grand design. For two months (two sprints) members didn't deliver anything to the feature teams. After the third month, when they finally delivered an initial version, it did not meet the needs or expectations of the feature teams.

What Rob's team should have done instead was work very closely with the feature teams and add new capabilities in the context of the features being delivered by the feature teams. This would have forced a much closer collaboration between the component team and the feature team, increasing the chances of delivering what was needed. Rob's team could have, for example, delivered only the ability to write fixed-length text data to the database in the first sprint. Feature teams who received that capability would not have been able to write numeric data, dates, and so on to the database. And they would not have been able to *read* any data. But, the feature teams could have done one thing—write fixed-length text data—and from that could have provided feedback to Rob and his team on the usability of the component.

Perhaps the best way to ensure that a component team hears the feedback it will need to create useful functionality is to staff the component team temporarily with people from the feature teams. A developer assigned to a component team who knows he will soon be moving back to a feature team will be more likely to make sure the work of the component team will be usable.

Deciding When a Component Team Is Appropriate

Whenever possible, form feature teams rather than component teams. I like to start out with the assumption that all teams on a multiteam project will be feature teams. I'm willing to back away from that assumption, but I only want to do so in the face of evidence that forming one or more component teams will be in the best interest of the product. I suggest considering a component team only when most of the following statements are true:

- **The component team will build something that will be used by multiple feature teams.** If a component will be used by only one feature team, have that feature team build it. This ensures that the new capability is built within the context of that team's needs and expectations, which makes the implementation more likely to be used. Even when a component team will build something useful to multiple teams, a better strategy is often to have one feature team build the functionality it needs and then have subsequent teams refactor and generalize the functionality as their needs arise.
- **Using a component team will reduce the sharing of specialists.** On some multiteam projects, some highly specialized disciplines are shared across many teams. Although some sharing of specialists is usually necessary, too much of it can be detrimental as the specialist's time becomes too fragmented. You may want to consider creating a component team if doing

SEE ALSO

For more on the evils of multitasking, see "Put People on One Project," later in this chapter.

so will make more manageable the extent to which specialists are shared across many teams.

- **The risk of multiple approaches outweighs the disadvantages of a component team.** If we choose to build a shared component or service by having multiple feature teams contribute to the effort, there are two related risks to be aware of. First is the risk that each feature team implements a different solution to the same problem. Second is the risk that the feature teams each build on top of what prior feature teams have done but do so without a cohesive vision. These risks could be great or small, depending on what shared functionality is being built. When the risk of multiple approaches is high, a component team is a valid option.
- **It will get people talking who might not talk otherwise.** People tend to talk more with those on their team than those outside their team. This is true even on a Scrum project. In fact, it may be especially true on a Scrum project because team members on Scrum projects come to identify so strongly with their teams. You can use this to your advantage by creating teams from people who need to work together but who might not naturally talk to each other. If past experience shows that a project's artificial intelligence programmers do not talk often enough, this can help justify the short-term use of a component team, as long as there are other reasons for doing so.
- **You can see an end to the need for the component team.** A component team should not linger around forever, like my in-laws after the holidays. The team should develop the functionality it has been pulled together to create and then disband as soon as possible. When first forming a component team, it is not necessary to know when it will disband; however, you should have some idea of either how long it will exist or what will be delivered by the time the team has fulfilled its purpose. Because a component team is a deviation from the ideal of having all feature teams, you should be reluctant to create a component team that looks as though it might exist forever.

While acknowledging the occasional benefits of using a component team, I want to stress again that the vast majority of teams on a large project should be feature teams. Wes Williams and Mike Stout have described what happened at Sabre Airline Solutions when beginning with component teams.

> Stories weren't complete from a user perspective. Teams were working on different features at different times with different acceptance criteria. There was a lot of rework coming back into the system. Teams were blaming each other for incomplete functionality, failing builds, tests, etc. In hindsight…the teams

> should have been structured along functional or feature lines. (2008, 359)

Who Makes These Decisions?

Ideally, the team makes decisions about how it is structured. If the team is to be trusted with solving the problem of how to build the product, it seems appropriate to trust it with the decision about how to structure itself to do so. However, though team members are accustomed to making technical decisions, they usually do not have a lot of experience making team organization decisions. So, initially the team may not be in the best position to design its own structure.

I've introduced Scrum to hundreds of teams. One of the things I've noticed is how frequently someone's initial exposure to Scrum results in an opinion like, "Scrum sounds wonderful for our company, and it will be great for all the other groups but not mine." Architects add, "After we do the up-front architecture, I can really see how this will help the programmers and testers." User experience designers say, "After we've done the up-front usability research, I can really see how this will work for the architects, programmers, and testers." Testers take the initial view, "It will be wonderful to have everyone working so closely together and then handing off to us for a big round of integration testing."

If we ask team members with these common initial mindsets to design the structure of their multiteam project, it shouldn't surprise us when they come back with plans for an architecture team, a programming team, a user experience team, and a test team. Of course I'm generalizing, but the tendency to think this way is so prevalent that it will be tempting to organize that way as well.

Initially, then, it is likely that functional managers, project managers, ScrumMasters, or those driving the transition to Scrum will make the decisions about how to organize the teams. These decision makers should solicit nonbinding input from their teams, especially from team members with past experience with Scrum or other agile methodologies.

What's Right Today May Be Wrong Tomorrow

An important thing to remember when selecting an appropriate team structure is that no team structure is forever. If the current team structure is impeding a team's or project's ability to use Scrum, that issue should be raised during an end-of-sprint retrospective. You don't want to continually change team structures, as team members need time to jell, but if the current structure is clearly wrong, change it.

As team members gain more experience with Scrum, it will be appropriate for them to become more involved in team structure decisions, including which teams are needed, whether each is a feature or component team, and who should be on each team.

THINGS TO TRY NOW

- ☐ Make a list of all teams on your current project. Identify whether each is a feature team or a component team. For each component team, consider the statements in the section, "Deciding When a Component Team Is Appropriate." Consider restructuring the team if not all statements were true.

Self-Organizing Doesn't Mean Randomly Assembled

The ability for a team to self-organize around the goals it has been given is fundamental to all agile methodologies, including Scrum. In fact, the Agile Manifesto includes self-organizing teams as a key principle, saying that "the best architectures, requirements, and designs emerge from self-organizing teams" (Beck et al. 2007). As part of deciding how best to achieve the goal given them, some teams will decide that all key technical decisions will be made by one person on the team. Other teams will decide to split the responsibility for technical decisions along technical boundaries: Our database expert makes database decisions, and our most experienced C# programmer makes C# decisions. Still other teams may decide that whoever is working on the feature makes the decision but has the responsibility of sharing the results of the decision with the team.

There are two key points here: First, not every team will choose to organize themselves the same way, and that's OK. Second, making use of the collective wisdom of the team will generally lead to a better way of organizing around the work than will relying solely on the wisdom of one personnel manager. However, the benefit of allowing a team to self-organize isn't that the team finds some optimal organization for its work that a manager may have missed. Rather, it is that by allowing the team to self-organize, it is encouraged to fully own the problem.

A common criticism of self-organizing teams is, "We cannot just put eight random individuals together, tell them to self-organize, and expect anything good to result." Well, I don't know if that's true, but when we are putting together a two-pizza Scrum team, we are definitely not doing so with eight randomly selected individuals. In fact, those in the organization responsible for initiating a Scrum project should expend a lot of effort in selecting the individuals who will comprise the team.

SEE ALSO

Chapter 12, "Leading a Self-Organizing Team," describes how leaders exert subtle, positive influence.

In the original paper describing Scrum, Takeuchi and Nonaka identified "subtle control" as one of its six principles. They list staffing decisions as a key management responsibility.

> Selecting the right people for the project team while monitoring shifts in group dynamics and adding or dropping members when necessary [is a key management responsibility]. "We would add an older and more conservative member to the team should the balance shift too much toward radicalism," said a Honda

> executive. "We carefully pick the project members after long deliberation. We analyze the different personalities to see if they would get along." (1986, 144)

Getting the Right People on the Team

If you are a personnel manager or otherwise influence team composition in your organization, some of the factors to consider are the following:

- **Include all needed disciplines.** As a cross-functional team, it is important that all skills necessary to go from idea to implemented feature be represented on the team. Initially this may mean that team size is slightly larger than desired. But, over time, individuals on a Scrum team will learn some of the skills possessed by their coworkers. This is a natural result of being on a Scrum team. As some team members develop broader skills, other individuals can be moved onto other teams.
- **Balance technical skill levels.** Subject to considerations of team size, you should strive to balance skill levels on the team. If a team has three senior programmers and no less-experienced programmers, the senior programmers will need to code some low-criticality features that they could find boring. Not only might a junior programmer have found such features enjoyable to work on, that programmer would also benefit from learning through association with the senior programmers.
- **Balance domain knowledge.** Just as we strive to balance technical skills, we should strive for a balance between those with deep knowledge of the domain in which we are working or the problem we are attempting to solve. This is not to say that if we have the opportunity to assemble a team entirely of domain experts we shouldn't take it. Rather, we should consider the long-term goals of our organization. One of those goals is likely the build up of domain knowledge throughout the organization. You'll have a hard time achieving that if you put all of the domain experts on one team.
- **Seek diversity.** Diversity can mean many different things—gender, race, and culture being just three among them. Perhaps equally important can be how individuals think about problems, how they make decisions, how much information they need before making a decision, and so on. Homogeneous teams reach consensus more quickly than do heterogeneous teams, but they do so by failing to consider all options (Mello and Ruckes 2006).
- **Consider persistence.** It takes time for team members to learn to work well together. Strive, therefore, to keep team members together who have worked well together in the past. When forming a new team, consider

how long members will be able to work together before some or all are dispersed to other commitments.

OBJECTION

"We can't self-organize because we have a dominating former technical lead who makes all decisions before we even have a chance to discuss the issue."

If possible, take the dominating personality aside and inform her of the issue. Let her know that even in situations where she may know the "right" thing to do, she should sometimes refrain from voicing her opinion before others have a chance to express their thoughts. Ask her if she thinks the team would make the right decision if she were to present her thoughts as an opinion rather than as an unchallengeable decision. Enlist her assistance as a mentor to the others—her job should be not just making sure the right decisions are made but that team members grow such that they will make the right decisions on their next projects, where she may not be there for them.

"My team won't self-organize; team members are too passive and look to me to lead."

If they look to you, look back right at them. If you are the team's ScrumMaster, make sure they know that your job is to support them, not to make decisions for them. If you are a team member, you do not need to subjugate your opinions and keep quiet all the time. However, you should look for ways to engage others by not making the decision in all cases. For example, try asking questions of others before giving your opinion.

"The team is too junior; members don't have enough experience to self-organize."

If they have enough experience to build a software product, they probably have enough experience to figure out how to organize themselves. If not, provide them with training or coaching. Often, this objection really masks the objection of, "I don't trust the team to self-organize in the way I want them to." Too bad. Exert subtle control over the team in who you put together to form the team and the goal you give that team, not in how it does its day-to-day work.

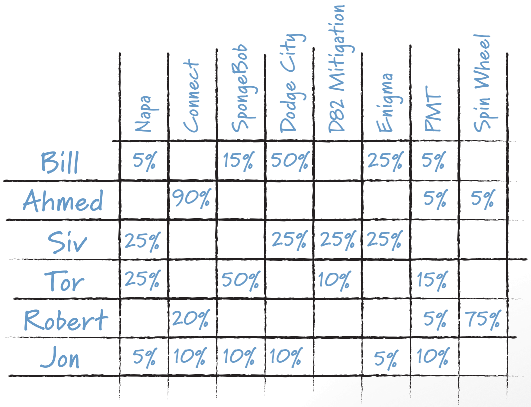
Put People on One Project

Individuals assigned to work on multiple projects inevitably get less done. Multitasking—attempting to work on two projects or two things at once—is one of the biggest drains on project team performance. Yet it has unfortunately become one of the busy manager's most frequently used tools. The reason for this, I believe, is that multitasking creates the illusion of progress and gives the manager the feeling that a problem has been solved. Really, though, in many cases the problem has been made worse.

Consider the case of Jon, a director of database engineering who managed a staff of database administrators (DBAs) who were woefully outnumbered by the programmers, testers, and other types of developers in his company. Jon was faced with allocating himself and his staff of five across more projects than they could handle. His solution was to create a spreadsheet like the one shown in Figure 10.5. Jon's spreadsheet allowed him to allocate DBAs across the various projects, which he did down to the 5% level. Five percent of an 8-hour day is 24 minutes. Through this spreadsheet Jon was telling Bill he could spend 24 minutes each on the Napa and PMT projects, Ahmed could spend the same on PMT and Spinwheel, and so on.

FIGURE 10.5

A portion of Jon's project staffing spreadsheet.



	Napa	Connect	SpongeBob	Dodge City	DB2 Mitigation	Enigma	PMT	Spin Wheel
Bill	5%		15%	50%		25%	5%	
Ahmed		90%					5%	5%
Siv	25%			25%	25%	25%		
Tor	25%		50%		10%		15%	
Robert		20%					5%	75%
Jon	5%	10%	10%	10%		5%	10%	

Did Jon really think that Bill would stop working on the Napa project after 24 minutes each day? Of course not. But he probably did think that Bill had enough control over his schedule that he could be close to 24 × 5 = 120 minutes in a week. What Jon was really doing in this situation was taking a problem (the correct allocation of resources) that he couldn't solve and pushing it down to the members of his team. What Jon should have done instead was push this problem up to his own manager.

Pushing problems toward the team is often a wonderful strategy. In fact, delegating problems to the team is at the heart of Scrum. However, when a problem is pushed toward the team, the team needs to be given the authority to solve the problem. In the case of Jon and his DBAs, it was obvious that one solution to consider was doing fewer concurrent projects. Without being empowered to enact that solution, they were put into an impossible-to-solve situation.

And they didn't solve it any better than Jon did. They invoked the age-old policy of "work on the project of whoever is screaming the loudest."

Time on Task Decreases with Too Many Tasks

Kim Clark and Steven Wheelwright studied the impact of multitasking on productivity. Their findings, shown in Figure 10.6, indicate that the total amount of time on task goes up when a person has two tasks to work on. After that, however, Clark and Wheelwright found that time on task decreased. In fact, with three tasks the amount of time on task decreased so much it was less than when an individual had only one task to work on (1992, 242).

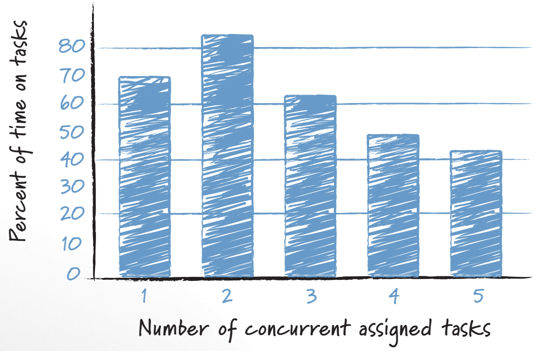


FIGURE 10.6

The amount of time spent on value-adding tasks decreases with three or more concurrent tasks.

If you have only one task to work on it is almost a certainty that you will occasionally be unable to work on that task. You will become blocked by waiting for someone to return a phone call, answer an e-mail, approve the design, or so on. And so it makes sense that the Clark and Wheelwright study shows that a person with two tasks to work on spent more time on task than did someone with only one task. However, consider that Clark and Wheelwright did this research in the early 1990s.

What's changed since then? For starters how about e-mail, instant messaging, the proliferation of mobile telephones, and any number of ways in which we communicate? My theory is that the bars in Figure 10.6 need to be shifted one space to the left to reflect today's faster pace. I remember clearly the job I had back in 1992 when Clark and Wheelwright published their results. I remember times back then when I was at my desk and thought, "I'm caught up; I have nothing to do right now." Of course, I haven't thought that since 1992.

The pace of the world has accelerated dramatically. Just being a good corporate citizen takes more time now than it did in 1992. There's more to read, more to process, and more for each person to do. Merely being an employee should count today as a first task for each of us. The first project we are on counts as a second, and we are then already optimally productive. Any further projects we are assigned just make us less productive.

One of the main reasons that multitasking is so horrible is the task-switching cost involved. There is tremendous overhead in getting started on one task, switching to another, and then switching back to the first. The more tasks or projects we are involved in, the more likely we are to be interrupted while working on them. One study of members of a software development team found that team members are interrupted every 11 minutes (Gonzales and Mark 2004). If you're reading this chapter at the office, it is likely that you were interrupted at least once while reading.

THINGS TO TRY NOW

- ☐ If you are a manager, make a list of your direct reports and the projects each is on. If anyone is on more than two projects, immediately find a way to change that. If you've already achieved this, see if you can reduce someone's allocation from two projects to one. Assess the situation after two sprints.

When Multitasking Is OK

All of this is not to say that we should never allow multitasking on our projects. It is sometimes helpful. The key is to remember that a person who is multitasking and shared across multiple projects is likely to get less total work done than if she had been dedicated fully to just one of those projects.

Let's again consider Jon and his DBAs. Suppose each DBA could complete "20 database tasks" per day assuming that all database tasks are the same size. A DBA fortunate enough to work on only 1 project would achieve this level of performance. However, a DBA on 2 projects might complete only 16 database tasks per day. And a DBA on 3 projects might complete only 14 database tasks per day.

Although these reduced levels of productivity may look quite bad, they may not be. Suppose 1 of our DBAs is assigned to 2 projects and is to split her time equally between them. She will be able to complete 8 database tasks on each project. This may be the optimal use of her time if neither of the projects needs 20 database tasks done in a day. If neither project needs more than 8 database tasks a day from her, then she is better split between both projects than dedicated entirely to 1. From this we can extract the following guidelines:

- In general, and for the majority of a project's team members, multitasking is to be avoided.
- Multitasking may be acceptable if a person cannot be fully or nearly fully utilized on a single project. If we look back to Figure 10.5 and Jon's DBAs, we see that the Connect project was allocated three people

with a total allocation greater than 100%. A better solution would likely have been to allocate a single person but for 100% of his time.

- Rather than have everyone multitask a little, it is better to have a few people multitask a lot. Figure 10.6 illustrates how the largest drop in time on task occurs after a person takes on the first task too many. In Jon's case, a better solution would have been to do anything possible to have two or three of his DBAs not have to multitask, even if that meant the others had to multitask even more.

The Corporate Form of Multitasking

Individuals feel compelled to multitask because the organizations in which we work attempt to multitask as well. The corporate form of multitasking is pursuing too many concurrent projects. When an organization takes on too many projects, people become shared across multiple projects, which leads to individual multitasking. The detrimental effect of multitasking then causes those projects to take longer, which leads to more multitasking near the end of the project when "we need to get started" on the next project.

An eight-year study of projects at a dozen companies and published in *Harvard Business Review* concluded that "projects get done faster if the organization takes on fewer at a time" (Adler et al. 1996). Corporate multitasking—attempting to make progress on too many concurrent projects—is what created the situation that Jon found himself in earlier in this chapter when he resorted to allocating his people to the 5% level.

Mary and Tom Poppendieck urge organizations to limit work to capacity. An organization that has more projects running concurrently than can be adequately staffed is attempting to work beyond its capacity. As they write, "If you expect teams to meet aggressive deadlines, *you must limit work to capacity* (2006, 134, emphasis is theirs).

Stopping the Treadmill

One of the happiest days of my life as a consultant was when I explained the impact of personal and corporate multitasking to the general manager of a large division of a big company. I could tell the message resonated with her. She asked me to follow her as she rose from her desk. We walked to a conference room near her office. She pointed toward a huge number of sticky notes stuck to the widest wall in the conference room and said, "We just made our plan for next year. There it is. Do you think we're doing too much?"

Her division had well over 100 developers but the wall was full. We talked about the plan, the number of concurrent projects, and the ripple effect that would occur if one project was substantially late. She knew they were planning to do too much, and I confirmed this for her. She convened a meeting for the next day of the vice presidents and directors who had made the plan and instructed them to start taking projects off the board. A look of relief (and surprise) went across the faces of everyone present. They had each known that the plan they had created the week before was overly ambitious and would not happen. However, no one had been willing to say so.

I checked back with this general manager a year later and was delighted—but not surprised—to hear that her division had just completed its most successful year ever. Part of that was attributable to the adoption of Scrum and the improvements it brought across her department. But an equal part of the success was attributable to the focus that was brought to each project by having fewer projects in progress at one time.

As this anecdote shows, often the best way to stop multitasking is to stop cold turkey. However, the reason I was so impressed with this general manager is that she is one of the few I have seen with the courage to do that. If you can't stop immediately, or if you're not in a position within the organization to make such a far-reaching decision, there are other things you may want to try.

Don't start a new project until it can be fully staffed. Avoid the temptation to start a new project with just a few analysts and maybe one programmer. Try to get everyone to agree that new projects will be started only when they can be staffed with all disciplines represented. This isn't to say you need to wait to start a large project until all 50 developers are available. Starting a new project only when at least one full team can be fully and appropriately staffed will help adjust the rate at which new projects are started to closer to the rate at which they can be developed.

Include ramp-up and wind-down time in enterprise plans. If, like the general manager in this section's story, you put together a big, annual plan, be sure to include the time necessary to start and stop the various projects. All too often a team provides an estimate of six months, and six months are reserved on an enterprise calendar. However, even on a Scrum project (especially from a new Scrum team), there may be a month or two of wind-down. During this time at least a subset of the team may be needed for high-priority bug fixes or to implement great, new ideas that were discovered only upon release. Failing to plan for some of this will cause unexpected periods of overlapping projects.

Institute simple rules. Gaining agreement on simple rules can help lead to the right organizational behavior. A simple rule such as "No one can be assigned to more than two projects" can work wonders. Johannes Brodwall, chief scientist with Steria in Norway, suggests one simple rule.

> Everyone on the team must be at least 60% allocated to the team. Sixty percent seems to be a magical number, which says to people, "This is the most important thing." With 60%, when one task suffers, it is usually one of those 10% or 20% tasks. So this structure guides people to be more dedicated to their primary team.

Go slow but go. I can totally respect the leap of faith required to believe that doing fewer concurrent projects will lead to more projects being completed. Even if they believe that completing projects more quickly will ultimately lead to increased productivity, people will be uncomfortable postponing or canceling large-scale projects. So, start small: Remove one project from the first quarter plan and see how it goes.

Guidelines for Good Team Structure

This section presents a set of guidelines to consider in designing an appropriate team structure. Each guideline is presented in the form of a question to be asked of a current or proposed team. The questions are intended to be asked iteratively. Ask each question of a current or proposed team, changing the structure as appropriate based on the answer. As the structure changes, reask the questions until you can answer "yes" to each.

Does the structure accentuate the strengths, shore up the weaknesses, and support the motivations of the team members? People don't enjoy being on a team where they are not able to make use of their strengths or are constantly required to do things they are bad at. Good team members are willing to do whatever is necessary for the success of the project, but that doesn't relieve us from the goal of trying to find a team structure that accentuates the strengths of as many team members as possible.

Does the structure minimize the number of people required to be on two teams (and avoid having anyone on three)? A well-conceived team structure for an organization that is not attempting to do too many concurrent projects will reduce multitasking to a tolerable level. If the organization is not attempting too many concurrent projects, yet more than 10–20% of all team members belong to more than one team, consider an alternative team design or deferring some projects.

Does the structure maximize the amount of time that teams will remain together? If other factors are equal, you should favor a design that allows team membership to persist over a longer period. It takes time for individuals to learn to work well together. Amortize the cost of that learning over a longer period by trying to leave teams together as long as possible, ideally even finding a team structure that can outlast the current project.

Are component teams used only in limited and easily justifiable cases? Most teams should be created around the end-to-end delivery of working features. In some cases, it is acceptable to have a component team developing reusable user interface components, providing access to a database, or similar functionality. But these should be exceptions.

Will you be able to feed most teams with two pizzas? Given the compelling productivity and quality advantages of small teams, the majority of teams in a good design should have five to nine members.

Does the structure minimize the number of communication paths between teams? A poor team structure design will result in a seemingly infinite number of communication paths between teams. Teams will find themselves unable to complete any work without coordinating first with too many other teams. Some interteam coordination will always be required. But, if a team that wants to add a new field on a form is required to coordinate that effort with three other teams, as I've seen, then the communication overhead is too high.

Does the structure encourage teams to communicate who wouldn't otherwise do so? Some teams will just naturally communicate with each other. An effective team design encourages communication among teams or individuals who should communicate but may not do so on their own accord. In fact, one valid reason to put someone on two teams is that doing so will increase the communication between those teams. If lack of communication between two teams is a concern, splitting a person's time between those two teams is easily justified.

Does the design support a clear understanding of accountability? A well-designed team structure will reinforce the concept of a shared, all-teams accountability for the overall success of the project while providing each team with clear indicators of its unique accountabilities.

Did team members have input into the design of the team? During the early stages of your transition to Scrum, this may not be possible. Individuals may not yet have enough experience delivering working, tested, ready-to-use products by the end of each sprint. Similarly, some individuals may be initially too resistant to Scrum to contribute to team structure discussions in constructive ways. In these cases, it is acceptable for managers outside the team to design an initial team structure.

While doing so, however, they should remember that this is a responsibility that will eventually need to be turned over to the team as a whole.

Onward

In this chapter we've looked at why Scrum teams should be kept small and used the analogy of being able to feed each team with two pizzas. To further enhance a team's ability to rapidly, correctly, and efficiently develop software products, we also considered whether teams should be structured around features or components. We concluded that in structuring multiple teams, we should seek to favor feature teams and try to avoid the use of component teams, while acknowledging they will occasionally be appropriate.

Next we dispensed with the myth that a self-organizing team is a random collection of individuals. As with any team, team members should be chosen with effort and care. We also looked in detail at the need to structure teams in such a way as to minimize the need for individuals to belong to two or more teams. Finally, we concluded with nine guidelines for structuring teams.

In the next chapter we turn our attention to the subject of teamwork. We look specifically at what the members of a single, two-pizza team can do to work well together during a sprint.

Additional Reading

DeMarco, Tom, and Timothy Lister. 1999. *Peopleware: Productive projects and teams*. 2nd ed. Dorset House.

It is impossible to say enough good things about this book. I remember the day in 1989 when my CEO told me, "After reading *Peopleware* this weekend, I am going to completely change our development group." She did, and the group excelled because of it. This book is full of advice on helping teams achieve their fullest potential.

Goldberg, Adele, and Kenneth S. Rubin. 1995. *Succeeding with objects: Decision frameworks for project management*. Addison-Wesley Professional.

This book precedes the agile movement but still contains some of the best advice on various team structures. Two chapters include a summary of various team structure options, how to choose among them, and case studies of how six teams chose to organize.

Hackman, J. Richard. 2002. *Leading Teams: Setting the stage for great performances*. Harvard Business School Press.

The premise of this book is that a leader's job is to design and support teams that can manage themselves. It includes an excellent chapter ("Enabling Structure") on how to structure teams.

Index

A
Abdelshafi, Ibrahim, 49
  rotating ScrumMasters, 123
ability, 31
  accountability, 32
  developing, 31-33
  sharing information, 32-33
  targets, 33
  training, 32
abnormal termination, 280
acceptance test-driven development. *See* ATDD
accountability
  ability, 32
  individuals versus teams, 406
ACH (Achievement Orientation), 359
achieving compliance, 400-401
ACT (Agile Champions Team), 65
ADAPT (Awareness, Desire, Ability, Promotion, Transfer), 21
  ability
    accountability, 32
    developing, 31-33
    sharing information, 32-33
    targets, 33
    training, 32
  awareness. *See* awareness
  desire, 26
    communication, 27-28
    engaging employees, 30-31
    fear, 29-30
    incentives, 29
    increasing, 27-30
    momentum, 28
    sense of urgency, 28
    test driving Scrum, 28-29
  promotion, 34-35
    agile safari, 36
    attracting attention, 36-37
    communication, 35-36
  transfer
    facilities, 38-39
    finance, 39-40
    human resources (HR), 38
    marketing, 39
adapting to Scrum, 21-23, 41
adding resources, 294
adjusting containers, 223-224
adopting Scrum
  levels of, 22
  reasons for moving slowly, 24
Adzic, Gojko, 250
agendas, scrum of scrum meetings, 342-343
Agile Champions Team (ACT), 65
agile:EF, general-purpose agility assessments, 432-433
agile phobias, resistance, 100-101
agile safari, promotion, 36
agility, public display of agility, 47-48
  reasons for using, 48-49
Allen-Meyer, Glenn, 34
  improvement community members, 76
all-in pattern
  reasons for using, 45-46
  versus start-small pattern, 46-47
altering exchanges, 226-227
alternatives to changing scope, 293-294
  adding resources, 294
  adjusting scope, 295
  cutting quality, 294
  extending schedules, 294-295
Amazon.com, two-pizza teams, 178
Ambler, Scott, 143
amplifying differences, 224-226
analysts, 137-139
Andersen Consulting, 164
Andersen, Ole, 369
Anderson, Philip, 220, 227
  self-organization,
Andres, Cynthia, 58
  overtime, 288
annual reviews, 405
anticipating resistance, 97-98
architects, 142-143
  non-coding architects, 143-144
areas of conflict, Scrum and sesquential development, 391-393
Armour, Phillip, 181
artificial intelligence (AI) programmers, 226
assessments
  comparative agility assessments, 434-436
  creating your own, 437-438
  general-purpose agility assessments, 430-431
    agile:EF, 432-433
    Shodan Adherance Survey, 431-432
ATDD (acceptance test-driven development), 317-318
  details, 318-320
attitudes, expectations (pilot projects), 91
attracting attention, promotion, 36-37
attributes of
  product owners, 130-131
  ScrumMasters, 118-120
automation, testing, 311-313
  benefits of, 316-317
  manual testing, 314
  sprints, 314-316
  user interface tests, 313-314

Avery, Christopher, 7, 61, 217
avoiding activity-specific sprints, 269-270
awareness, 23-24
  developing
    communication, 25
    exposure to new people and experiences, 25-26
    focusing on reasons for change, 26
    with metrics, 25
    pilot projects, 26

B

Babinet, Eric, 348
backlog management, 330
backlogs, improvement backlogs, 62-63
balanced scorecards
  metrics, benefits of, 443-444
  teams, 438-439
    constructing balanced scorecards, 439-440
    favoring simple metrics, 441-443
Barnett, Liz, 58
Beach Hut Deli, 205
Beck, Kent, 58
  overtime, 288-289
behavior, reinforcing learning, 210-211
Benefield, Gabrielle, 396, 410, 415
Biddle, Robert, 129
big-room approach, spring planning meetings, 346-347
billiard ball sprints, 266
BioWare, 25
Boehm, Barry, 168, 391
boundaries, responsibilities of product owners, 126-127
Boy Scout Rule, 159
Bridges, William, 30
Brodwall, Johannes, 264
Brooks, Fred, 158, 294
business processes, Scrum and sequential development, 391
business sponsor engagement, attributes of pilot projects, 83

C

Capability Maturity Model Integration (CMMI), 399-400
career paths, human resources, 411-412
Carmel, Erran, 362, 368
categorizing people, 99
Catmull, Ed, 214
CDE (Container, Differences, Exchange), 223-225
challenging teams, 211
change, 58
  communicating about, 101
    hearing from leaders, 101-102
    hearing from peers, 102-103
changing scope, alternatives to, 293-294
  adding resources, 294
  adjusting scope, 295
  cutting quality, 294
  extending schedules, 294-295
charts, 418
chief product owner, 329
China, cultural differences, 360-361
choosing when to start pilot projects, 84-85
Cichelli, Sharon, 357
Cirillo, Francesco; pomodoro, 292
Clark, Jonathan, 82
Clark, Kim, 193
CMMI (Capability Maturity Model Integration), 399-400
coaching, ability, 32
Cockburn, Alistair, 6, 257
coherence, distributed teams, 359
  cultural differences, 359-361
  team subcultures, 362-363
  trust, 365-366
collaborating collocated teams, 356-357
collaboration
  attributes of ScrumMaster, 119
  encouraging through commitment, 215-217
collective ownership, 160-161
commitment
  attributes of ScrumMaster, 120
  historical velocity, 301-302
  team size changes frequently, 303-304
  through encouraging collaboration, 215-217
  whole-team commitment, 204
commitments, making, 300-301
committing, 296-297
  data for, 297-300
communication, 236
  awareness, developing, 25
  about change, 101
    hearing from leaders, 101-102
    hearing from peers, 102-103
  desire, 27-28
  distributed teams
    documentation, 372-373
    lateral communication, 374
    product backlogs, 373
  emergent requirements, 242-243
  promotion, 35-36
  teams, 198
  user stories, product backlogs, 238-239
  written documents, 236-241
communities of practice, cultivating, 347-348
  creating environments for, 350-351
  formal or informal, 349-350
  participation, 351-352
comparative agility assessments, 434-436
compliance, 396-397
  achieving, 400-401
  CMMI (Capability Maturity Model Integration), 399-400

  ISO 9001, 397-399
component teams, 183-185
  building components as needed, 185-186
  deciding when to use, 186-187
components, building as needed, 185-186
conditions of satisfaction, 248
conference calls, sprint planning meetings, 378-379
confidence, 217
connectors, 358
Conner, Daryl, 91
conservers, 99
constructing balanced scorecards, 439-440
contact visits, 369-370
containers
  adjusting, 223-224
  influencing self-organization, 221-223
continuous integration, 162-163
Cooper, Dr. Robert, 394
coordinating work among teams, 340
  scrum of scrums meetings, 340-343
  synchronizing sprints, 343-345
Cornell, John, 417
corporate multitasking, 195
costs, benefits of transitioning, 11-12
Crispin, Lisa, 150, 202
Crosby, Philip, 148, 294
cross-functional teams, documentation, 253
cultivating communities of practice, 347-348
  creating environments for, 350-351
  formal or informal, 349-350
  participation, 351-352
cultural differences, 362
  distributed teams, 359-361
  holidays, 361-362
Cunningham, Ward, 320

D

daily scrum, distributed teams (meetings), 381-384
dampening differences, 224-226
data
  for committing, 297-300
  for estimating, 297-300
database administrators, 148
debt, technical debt. *See* technical debt
Deemer, Pete; pilot projects, 92
DEEP (Detailed Appropriately, Estimated, Emergent, Prioritized), 254
  product backlogs, 253-254
deliberately distributed teams, 357-358
delivering something valuable, sprints, 262-265
dependencies, managing, 333-334
  integration teams, 337-339
  kickoff meetings, 336-337
  rolling lookahead planning, 334-336
  sharing team members, 337
design, 166-167
  adapting to user needs, 167-168
  guiding, 169-170
designing teams for learning, 209-210
desire, 26
  communication, 27-28
  engaging employees, 30-31
  fear, 29-30
  incentives, 29
  increasing, 27-30
  momentum, 28
  sense of urgency, 28
  test driving Scrum, 28-29
desks, 416
details, ATDD, 318-320
development, Scrum and sequential development, 391
DHL, 206, 207
diehards, 110-112
differences
  amplifying, 224-226
  dampening, 224-226
  influencing self-organization, 221-223
Dinwiddie, George, 417
disbanding IC (improvement communities), 78
distributed teams
  coherence, 359
    cultural differences, 359-361
    team subcultures, 362-365
    trust, 365-366
  collaborating collocated teams, 356-357
  communication
    documentation, 372-373
    lateral communication, 374
    product backlogs, 373
  deliberately distributed teams, 357-358
  get-togethers
    contact visits, 369-370
    seeding visits, 367-369
    traveling ambassadors, 370-372
  meetings, 375-377
    daily scrum, 381-384
    low-fidelity videoconferencing, 378
    scrum of scrums, 384-385
    sharing the pain, 377
    small talk, 376
    sprint planning meetings, 378-381
    sprint reviews and retrospectives, 385-386
  subgroups, 366
diversity, finding the right people, 190
Doberman impression, 417
documentation
  communication, distributed teams, 372-373
  cross-functional teams, 253
Drummond, Brian, 365
Duarte, Deborah, 365
Dunbar, Robin, 332
duration, attributes of pilot projects, 83

E

Electronic Entertainment Expod (E3), 289
eliminating knowledge waste, 213-215
emergent requirements, 242-243
  product backlog, 242-243
employee engagement,
  benefits of, 13-14
  desire, 30-31
encouraging collaboration through commitment, 215-217
end of project testing, 308-310
energizing the system, 231-232
energy, planning, 291-292
engaging employees, desire, 30-31
Enterprise Transition Community (ETC), 63-65
  responsibilities of, 68-70
  sprints, 65-66
    sponsors and product owners, 66-67
Eoyang, Glenda, 68
epics, 247-248
ePlan Services, 259
estimating, 296-297
  data for, 297-300
ETC (Enterprise Transition Community), 63-65
  responsibilities of, 68-70
  sprints, 65-66
    sponsors and product owners, 66-67
evolution
  influencing, 227-228
    defining performance, 229
    energizing the system, 231-232
    introducing vicarious selection systems, 230-231
    managing meaning, 229-230
    selecting external environments, 228-229
excellence, striving for, 155-156
  collective ownership, 160-161
  continuous integration, 162-163
  pair programming, 164-166
  refactoring, 158-160
  test-driven development, 156-157
exchanges
  altering, 226-227
  influencing self-organization, 222-223
executive sponsorship, space (facilities), 415-416
expectations, pilot projects, 88-89
    attitudes, 91
    involvement, 91-92
    predictability, 90-91
    progress, 89-90
extending
  schedules, 294-295
  sprints, 278-279
external ScrumMasters, 122
extract method, 159

F

facilities, 412-413
  charts, 418
  feedback devices, 418
  food and drink, 420
  furniture, 416-417
  privacy, 419
  product backlog, 419
  proximity of team members, 418
  space, 413-414
    executive sponsorship, 415-416
    war rooms, 414-415
  spring backlog, 418-419
  task boards, 419
  transfer, 38-39
  whiteboards, 419
  windows, 420
facilities groups, 405
failure of pilot projects, 87-88
Farm Credit Services of America, ETC, 65
fear, desire, 29-30
feature teams, 182-184, 188-189
  decisionmakers, 188
Fecarotta, Joe, 400
FedEx, 206
feedback
  sprints, 283
  testing at end of project, 309
feedback devices, 418
Feynman, Richard, 277
Fichtner, Abby, 8, 157
finance, transfers, 39-40
followers, 112-114
food and drink, 420
Ford, Doris, 141
Fowler, Martin, 240, 367
  traveling ambassadors, 370
frequency of scrum of scrum meetings, 342
Fry, Chris, 32, 442
Fuji-Xerox, 127
functional managers, 144-145
  leadership, 145-146
  personnel responsibilities, 146
furniture, facilities, 416-417
future shock, transitioning, 9

G

gang programming, 158
Garbardini, Juan, 399
Gates, Bill, 231
general-purpose agility assessments, 430-431
  agile:EF, 432-433
  Shodan Adherance Survey, 431-432
get-togethers
  contact visits, 369-370
  seeding visits, 367-369

- traveling ambassadors, 370-372
- Gladwell, Malcolm, 358
- goals
  - ability, 33
  - IC (improvement communities), 76
  - sprints, 279-281
    - avoiding redirecting teams, 281-283
- Goldman, Sylvan, 103
- Goldstein, Professor Jeffrey, 73
- Google, improvement communities, 72-73
- governance, 394-396
  - non-agile governance, 395-396
- Gratton, Lynda, 216, 231
- Greene, Steve, 32, 442
- grooming product backlog, 244-245
- grouplets, Google, 72
- grow-and-split pattern, 51-52
  - reasons for preferring, 53
- guidelines for team structure, 197-199
- guiding design, 169-170

H

- Hackman, Richard, 212
- hand-offs, 214
- Haque, Nimat, 414
- Hewlett-Packard, 210
- High Moon Studios, 85
  - overtime, 289
- Highsmith, Jim, 231
- Hofstede, Geert, 359
- Hogan, Ben, 371
- holidays, cultural differences, 361-362
- Honious, Jeff, 82
- hot spots, 216
- Houle, Benoit
  - promotion, 36
  - transfers, facilities, 39
- human resources (HR), 406
  - career paths, 411-412
  - people issues, 412
  - performance reviews, 408-410
  - reporting structures, 407
    - reporting to product owners, 407-408
    - reporting to ScrumMaster, 407
  - reviews, 405
  - team members, removing, 410-411
  - transfer, 38
- humility, attributes of ScrumMaster, 119

I

- IBM
  - ability, 33
  - improvement communities, 71
- IC (improvement communities), 70-73
  - catalysts for improvement, 72-73
  - disbanding, 78
  - goals, 76
  - members, 76-77
  - sprints, 74-75
- impending doom, pilot projects, 84-85
- importance, attributes of pilot projects, 83
- improvement backlogs, 62-63
- improvement communities. *See* IC
- improving, 447-448
  - technical practices, 171-172
- incentives, desire, 29
- increasing
  - desire, 27-30
  - passion, 291-292
- IND (Individualism), 359
- individuals
  - accountability, 406
  - multitasking, 194-195
  - putting on one project, 191-192
  - putting on one task, 193-194
  - resistance, 98-100, 104-106
    - diehards, 110-112
    - followers, 112-114
    - saboteurs, 109-110
    - skeptics, 106-108
  - stopping the treadmill, 195-197
- influencing
  - evolution, 227-228
    - defining performance, 229
    - energizing the system, 231-232
    - introducing vicarious selection systems, 230-231
    - managing meaning, 229-230
    - selecting external environments, 228-229
  - self-organization, 220-221
    - containers, 221-223
    - exchanges, 222-223
- influential, attributes of ScrumMasters, 120
- integrating
  - testing in projects, 309-310
  - testing in the process, 308
- integration teams, 337-339
- internal coaching, reasons for preferring, 53
- internal ScrumMasters, 122
- interrupt-driven organizations, 281
- introducing new technical practices, 55
  - delaying, 56-57
  - starting soon, 55-56
- introducing vicarious selection systems, 230-231
- involvement, expectations (pilot projects), 91-92
- iron triangle, 292-293
  - changing scope. *See* changing scope
  - project context, 296
- ISO (International Organization for Standardization) 9001, compliance, 397-398

J

Jain, Apurva, 399
Jakobsen, Carsten, 400
Jeffries, Ron, 129, 418
job satisfaction, benefits of, 13-14
Jobs, Steve, 268-269
Johnson, Kent, 400
Johnston, Andrew, 143
Jones, Do-While, 166
Jones, Quinn, 65

K

Katzenbach, Jon, 202
Keith, Clinton, 17, 85, 289
Kerievsky, Joshua, 57
kickoff meetings, 336-337
knowledge
  attributes of ScrumMasters, 120
  sharing, 210
knowledge waste, eliminating, 213-215
Kofax, 417
Kotter, John, 5
Krebs, Bill, 431

L

large product backlogs, 330-333
Lasorda, Tommy, 215
lateral communication, distributed teams, 374
Lawrence, Paul, 97, 114
leaders, communicating about change, 101-102
leadership, 232-233
  functional managers, 145-146
learning, team learning, 209
  eliminating knowledge waste, 213-215
  ensuring learing conditions exist, 209-212
  supportive learning environments, creating, 212-213
levels of adopting Scrum, 22
Lewin, Kurt, 219
Liker, Jeffrey, 145
low-fidelity videoconferencing, 378
LTO (Long-Term Orientation), 359

M

MacDonald, John D., 27
Mah, Michael, 10
Management By Flying Around (MBFA), 370
Management By Walking Around (MBWA), 370
managing dependencies, 333-334
  integration teams, 337-339
  kickoff meetings, 336-337
  rolling lookahead planning, 334-336
  sharing team members, 337
managing meaning, 229-230
Mann, Chris, 13
Manns, Mary Lynn, 6
manual testing, 314
marketing, transfers, 39
Marsh, Stephen, 163
Martin, Angela, 129
Martin, Robert C., 159
Maurer, Frank, 13
MBFA (Management By Flying Around), 370
MBWA (Management By Walking Around), 370
McCarthy, Catherine, 291
McCarthy, Jim, 258
McClelland, Kent,
measurements, 430. *See also* assessments
  purpose of, 429-430
meetings
  distributed teams, 375-377
    daily scrum, 381-384
    low-fidelity videoconferencing, 378
    scrum of scrums, 384-385
    sharing the pain, 377
    small talk, 376
    sprint planning meetings, 378-381
    sprint reviews and retrospectives, 385-386
    telling everyone who is speaking, 377
  one-city retrospectives, 386
  regional meetings, 383-384
  writing meetings, 382
members, IC (improvement communities), 76-77
metrics
  benefits of, 443-444
  developing awareness, 25
milestones, sprints, 259
Miller, Ade, 368
  lateral communication, 374
Miller, Lynn, 151, 273
Mintzberg, Henry, 67
momentum, desire, 28
Moore, Pete, 118
motivation, 216
multitasking
  corporate multitasking, 195
  individuals, 194-195
mum effect, 374

N

Nessier, Roger, 379
new technical practices, introducing, 55
  delaying, 56-57
  starting soon, 55-56
Nielsen, Jakob, 153
Noble, James, 129
non-agile governance, 395-396
non-coding architects, 143-144

O

OCR (optical character recognition), 170
Olson, Edwin, 68, 222

one-city retrospectives, meetings, 386
open space, 336
optical character recognition (OCR), 170
organizational gravity, 405
originators, 99
Oticon, 369
overcoming resistance
  diehards, 111
  followers, 113
  saboteurs, 109-110
  from skeptics, 107
overlapping user experience design (UED), 271-272
overtime, 13-14, 287-291
  productivity and, 290-291

P

pair programming, 164-166
Pantazopoulos, Stelios, 163
Paranen, Jyri, 150-151
participation, communities of practice, 351-352
passion, increasing, 291-292
patterns
  all-in pattern, 44
    reasons for using, 45-46
    versus start-small pattern, 46-47
  choosing your approach to spreading Scrum, 54
  grow-and-split pattern, 51-52
    reasons for preferring, 53
  split-and-seed pattern, 50-51
    reasons for preferring, 52-53
  start-small pattern, 43
    versus all-in pattern, 46-47
    reasons for using, 44-45
Paulk, Mark, 399
paying down technical debt, 321-322
paying off technical debt, 320-321
PDI (Power Distance Index), 359
peers, communicating about change, 102-103
people
  categorizing, 99
  PMO, 421
  Scrum and sequential development, 391
people issues, human resources, 412
performance, influencing evolution, 229
performance reviews, human resources, 408-410
personnel responsibilities, functional managers, 146
PetroSleuth, 16
Philips Research, 52
phones, 416-417
Pichler, Roman, 253
piecemeal transitions, 57
pilot projects
  attributes of ideal pilot projects, 82-83
  awareness, developing, 26
  choosing when to start, 84-85
  expectations, 88-89
    attitudes, 91
    involvement, 91-92
    predictability, 90-91
    progress, 89-90
  failure of, 87-88
  selecting, 81-82
pilot teams, selecting, 86-87
planning, 285
  energy, 291-292
  overtime, 287-291
  progressively refining plans, 286-287
  rolling lookahead planning, 334-336
Platt, Lew, 210
PMI (Project Management Institute), 292
PMO (project management office), 405, 420
  people, 421
  processes, 422-423
  projects, 421-422
  renaming, 423
pomodoro, 292
Poppendieck, Mary, 195
Poppendieck, Tom, 195
potentially shippable, working software (sprints), 258-260
potentially shippable guidelines, identifying, 260-262
Power Distance Index (PDI), 359
predicatability, expectations (pilot projects), 90-91
preparing in this sprint for the next, 266-268
pragmatists, 99
Primavera Systems, 397
  ETC sprints, 67
privacy, 419
processes, PMO, 422-423
product backlog, 330-333, 419
  DEEP, 253-254
  distributed teams, communication, 373
  emergen requirements, 330-333, 419
  grooming, 244-245
  iceberg, 243-245
  items, 208
  refining requirements for, 245-246
  starting without specifications, 249-252
    cross-functional teams, 253
  user stories, 238-239
    refining, 246-248
  views, 332
productivity
  benefits of transitioning, 11-12
  overtime and, 290-291
  small teams, 180-182
product line owner, 329
product logs, starting without specifications (specifying by example), 250-251

product owners, 125, 128-129
  attributes of, 130-131
  ETC sprints, 66-67
  overcoming problems, 132-134
  reporting to, 407-408
  responsibilities of, 125-127
  scaling, 327-328
    sharing responsibility, dividing functionality, 328-329
  ScrumMasters as, 131-132
  teams, 129-130
programmers, 146-147
progress, expectations (pilot projects), 89-90
project context, iron triangle, 296
Project Management Institute (PMI), 292
project management office (PMO), 405, 420
  people, 421
  processes, 422-423
  projects, 421-422
  renaming, 423
project managers, 139-142
projects, PMO, 421-422
promotion, 34-35
  agile safari, 36
  attracting attention, 36-37
  communication, 35-36
PTON, Salesforce.com, 77
public display of agility, 47-48
  reasons for using, 48-49
  versus stealth transition, 50
Putnam, Doug, 180

Q

QSM, 180
quality, 293
  benefits of transitioning, 15-16
  reducing, 293-294
  teams, 323
  whole-team responsibility, 202

R

Ramanathan, Rajani, 348
Rayhan, Syed, 414
Reed Elsevier, 82
refactoring, 158-160
referencing success, 396
refining
  plans, 286-287
  user stories, 246-248
regional meetings, 383-384
reinforcing learning, behavior, 210-211
removing team members, 410-411
renaming PMO, 423
reporting structures, 407
  reporting to product owners, 407-408
  reporting to ScrumMaster, 407
requirements, refining for product backlog, 245-246
resistance
  agile phobias, 100-101
  anticipating, 97-98
  individuals, 98-100, 104-106
    diehards, 110-112
    followers, 112-114
    saboteurs, 109-110
    skeptics, 106-108
  as a useful red flag, 114
  waterfallacies, 100-101
resources, adding, 294
responsibilities of
  ETC, 68-70
  product owners, 125-127, 328-329
responsibility
  attributes of ScrumMaster, 119
  whole-team responsibility, 201-203
  written documents, 237
retention, 227
reviews, 405
  human resources, 408-410
Rico, David, 11, 12
Ringelmann, Max, 179
Rising, Linda, 6
Robarts, Jane, 364
  documentation, distributed teams, 373
  traveling ambassadors, 371
Robbins, Stephen, 179
roles, ScrumMasters. *See* ScrumMasters
rolling lookahead planning, 334-336
rotating ScrumMasters, 122-123
Rubin, Kenny, 358, 434

S

saboteurs, 105, 109-110
Sabre Airline Solutions, 187
Salesforce.com, 3
  ability, 32
  balanced scorecards, 442
  communities of practice, 348
  kickoff meetings, 336
  PTON, 77
  testing automation, benefits of, 316
SAS, career paths, 411-412
scaling
  product owners, 327-328
    sharing responsibility, dividing functionality, 328-329
  Scrum, 352
  spring planning meetings, 345
    big-room approach, 346-347
    staggering by a day, 345-346
scatter, 213
scenarios of interaction, Scrum and sequential development, 390-391

Schatz, Bob, 49
  rotating ScrumMasters, 123
Schubring, Lori, 21
  desire, 28
  promotion, attracting attention, 36
Schwaber, Ken, 142, 266
Schwartz, Tony, 291
scope, adjusting, 295
Scrum, sequential development and, 389-391
  areas of conflict, 391-393
  coexisting, 393-394
  governance, 394-396
  scenarios of interaction, 390-391
ScrumMasters, 117-118, 142
  attributes of, 118-120
  external, 122
  internal ScrumMasters, 122
  overcoming problems, 123-125
  as product owners, 131-132
  reporting to, 407
  rotating, 122-123
  tech leads as, 121-122
scrum of scrum meetings, 340-343
  agendas, 342-343
  distributed teams, 384-385
  frequency of, 342
Sears, 145
seeding visits, 367-369
Seffernick, Thomas, 65
selecting
  pilot projects, 81-82
  pilot teams, 86-87
selecting external environments, 228-229
selection, 227
self-organization, influencing, 220-221
  containers, 221-223
  differences, 221-223
  exchanges, 222-223
self-organizing teams, 189-190, 220
  finding the right people, 190-191
sense of urgency, desire, 28
separating estimating from committing
  commitment, 300-301
    data for, 297-300
    historical velocity, 301-302
    team size changes frequently, 303-304
sequential development, Scrum and, 389-391
  areas of conflict, 391-393
  coexisting, 393-394
  governance, 394-396
  scenarios of interaction, 390-391
Shamrock Foods, 61-62
shared visions, team subcultures, 363
sharing
  information, ability, 32-33
  knowledge, 210
  team members, 337
Shodan Adherance Survey, 431-432
size
  attributes of pilot projects, 83
  of teams, 178-179
skeptics, 105-108
Sliger, Michele, 26-27, 393
small talk, meetings (distributed teams), 376
Smith, Douglas, 202
Snyder, Nancy, 365
social loafing, 179
space (facilities), 413-414
  executive sponsorship, 415-416
  war rooms, 414-415
specialists, 204-205
split-and-seed pattern, 50-51
  reasons for preferring, 52-53
sponsors, ETC sprints, 66-67
spreading Scrum, choosing approach to, 54-55
sprint backlog, 418-419
sprint planning meetings
  distributed teams, 378-381
  scaling
    big-room approach, 346-347
    staggering by a day, 345-346
sprint reviews and retrospectives, distributed teams (meetings), 385-386
sprints
  automation, 314-316
  billiard ball sprints, 266
  delivering something valuable, 262-265
  ETC, 65-66
    sponsors and product owners, 66-67
  feedback, 283
  goals, 279-281
    avoiding redirecting teams, 281-283
  IC (improvement communities), 74-75
  preparing in this sprint for the next, 266-268
  synchronizing, 343-345
  team work, 268-269
    architecture and database design, 274-276
    avoiding activity-specific sprints, 269-270
    overlapping UED (user experience design), 271-272
    think holistically, work incrementally, 273-274
  timeboxes, 276-278
    extending, 278-279
  working software, 258
    defining potentially shippable, 258-260
    identifying potentially shippable guidelines, 260-262
stage-gate process, 394
stakeholder satisfaction, benefits of transitioning, 16

start-small pattern, 43
  reasons for using, 44-45
  versus all-in pattern, 46-47
stealth transition, 47-48
  reasons for using, 49-50
  versus public display of agility, 50
Stout, Mike, 187
subgroups, 366
success, referencing, 396
supportive learning environments, creating, 212-213
sustainable pace, 288-290
Sutherland, Jeff, 357, 368, 400
SW-CMM (Software Capability Maturity Model), 399
Sy, Desiree, 151, 271
synchronizing sprints, 343-345

T

Tabaka, Jean, 67
  ScrumMasters, 121
targets, ability, 33
task boards, 419
TDD (test-driven development), 156-158
team learning, 209
  eliminating knowledge waste, 213-215
  ensuring learing conditions exist, 209-212
team members
  analysts, 137-139
  architects, 142-143
    non-coding architects, 143-144
  database administrators, 148
  functional managers, 144-145
    leadership, 145-146
    personnel responsibilities, 146
  programmers, 146-147
  project managers, 139-142
  removing, 410-411
  sharing, 337
  testers, 148-151
  User Experience Designers (UEDs), 151-153
teams
  accountability, 406
  balanced scorecards, 438-439
    constructing, 439-440
    favoring simple metrics, 441-443
  communication, 198
  component teams, 183-185
    building components as needed, 185-186
    deciding when to use, 186-187
  coordinating among,
    scrum of scrums meetings, 340-343
    synchronizing sprints, 343-345
  cross-functional teams, 253
  designing for learning, 209-210
  distributed teams. *See* distributed teams
  feature teams, 182-184, 188-189
    decision makers, 188
  guidelines for structure, 197-199
  hand-offs, 206-207
  individuals
    multitasking, 194-195
    putting on one project, 191-192
    putting on one task, 193-194
    stopping the treadmill, 195-197
  motivating challenges, 211
  pilot teams, selecting, 86-87
  product backlog items, 208
  product owners, 129-130
  productivity, small teams, 180-182
  quality, 323
  Scrum and sequential development teams, 392
  self-organizing, 189-190
    finding the right people, 190-191
  self-organizing teams, 220
  size of, 178-179
  specialists, 204-205
  split-and-seed pattern, 51
  two-pizza teams, 177-178
  tying up loose ends, 207-208
  whole-team commitment, 204
  whole-team responsibility, 201-203
team subcultures
  distributed teams, 362-363
  reach agreements, 364-365
  shared visions, 363
teamwork, sprints, 268-269
  architecture and database design, 274-276
  avoiding activity-specific sprints, 269-270
  finish-to-finish relationships, 270-271
  overlapping UED (user experience design), 271-272
  think holistically, work incrementally, 273-274
teamwork factors, performance reviews, 408
tech leads as ScrumMasters, 121-122
technical debt
  paying down, 321-322
  paying off, 320-321
technical practices, improving, 171-172
Terashima, Hiroki, 376
test automation pyramid, 311-313
test-driven development (TDD), 156-158
test driving Scrum, desire, 28-29
testers, 148-151
testing
  automation, 311-313
    benefits of, 316-317
    manual testing, 314
    sprints, 314-316
    user interface tests, 313-314
  end of project testing, 308-310
  integrating into processes, 308-310
  manual testing, 314
tests, ATDD. *See* ATDD

Thierren, Elaine, 363
Thomas, Dave, 171
time to market, benefits of transitioning, 14-15
timeboxes, sprints, 276-278
  extending, 278-279
Toffler, Alvin, 9
Topp, Gregory, 138
training
  ability, 32
  PMO (project management office), 421
transfer, 37-38
  facilities, 38-39
  finance, 39-40
  human resources (HR), 38
  marketing, 39
transitioning, 5
  benefits of, 10-11
    current process no longer working, 17
    employee engagement and job satisfaction, 13-14
    productivity and costs, 11-12
    quality, 15-16
    stakeholder satisfaction, 16
    time to market, 14-15
  end state is unpredictable, 6-7
  future shock, 9
  pervasiveness of Scrum, 7-8
  to Scrum, 8-9
  successful change is not entirely top-down or bottom-up, 5-6
transitions
  piecemeal, 57
  stealth, 47-48
    reasons for using, 49-50
    versus public display of agility, 50
traveling ambassadors, 370-372
trust, distributed teams, 365-366
Truxaw, Matt, 377
  promotion, 36
Turner, Richard, 391
  CMMI, 399
two-pizza teams, 177-178

U

UAI (Uncertainty Avoidance Index), 359
UED (user experience design), 151-154
  overlapping, 271-272
unattended interfaces, integation teams, 338
Uncertainty Avoidance Index (UAI), 359
United States, cultural differences, 360-361
unobservable features, 264
Unson, J. F., 33, 365
  improvement communities, 73
User Experience Designers (UEDs), 151-153
  overlapping, 271-272
user interface tests, 313-314
user stories, 241
  product backlogs, 238-239
  refining, 246-248

V

"Valley of Death," 23
variation, 227
velocity, 297-298
  calculating averages, 305
  historical velocity, 301-302
VersionOne survey, 12, 16
vicarious selection systems, 230-231
views, product backlog, 332
visions, responsibilities of product owners, 125

W

Ward, Allen, 213-215
war rooms, 414-415
waterfallacies, resistance, 100-101
waterfall-at-end, Scrum and sequential development, 390
waterfall-in-tandem, Scrum and sequential development, 391
waterfall-up-front, Scrum and sequential development, 390
Weinberg, Gerald, 413
Wenger, Etienne, 349
Wheelwright, Steven, 193
whiteboards, 419
whole-team commitment, 204
whole-team responsibility, 201-203
Williams, Wes, 187
windows, 420
Wingard, Trond, 84, 252
Woodward, Elizabeth
  ability, 32
  ETC, 66
working software, 258
  sprints
    defining potentially shippable, 258-260
    identifying potentially shippable guidelines, 260-262
writing meetings, 382
written documentation, 236-241

X

Xebia, 368

Y-Z

Yahoo!
  ability, 33
  expectations, 91
  improvement communities, 73
Young, Cynick, 376