Disciplined Agile Delivery

A Practitioner's Guide to Agile Software Delivery in the Enterprise

Scott W. Ambler • Mark Lines

Foreword by Dave West

IBM Press Program Managers: Steven Stansel, Ellice Uffer
Cover design: IBM Corporation
Publisher: Paul Boger
Marketing Manager: Stephane Nakib
Publicist: Heather Fox
Acquisitions Editor: Bernard Goodwin
Managing Editor: Kristy Hart
Designer: Alan Clements
Project Editor: Betsy Harris
Copy Editor: Geneil Breeze
Indexer: Erika Millen
Compositor: Nonie Ratcliff
Proofreader: Debbie Williams
Manufacturing Buyer: Dan Uhrig

Published by Pearson plc
Publishing as IBM Press

IBM Press 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
1-800-382-3419
corpsales@pearsontechgroup.com

For sales outside the U. S., please contact:

International Sales
international@pearsoned.com

Contents

Part 1: Introduction to Disciplined Agile Delivery (DAD)

Foreword

The process wars are over, and agile has won. While working at Forrester, we observed that agile methods had gone mainstream, with the majority of organizations saying that they were using agile on at least 38% of their projects. But the reality of agile usage, as Scott and Mark point out, is far from the original ideas described by the 17 thought leaders in 2001. Instead, agile is undermined by organizational inertia, politics, people's skills, management practices, vendors, and outsourced development. I observed that the reality of agile was something more akin to *water-scrum-fall*—water-scrum describing the inability of an organization to start any project without a lengthy phase up front that defined all the requirements, planning the project in detail, and even doing some of the design. Scrum-fall defines the release practices operated by most organizations in which software is released infrequently, with costly and complex release practices that include manual deployments and testing. Water-scrum-fall is not all bad, with some benefits to the development team working in an iterative, scrum-based way, but water-scrum-fall does not release the power of agile. Enterprise agile not only creates the most efficient software development process but more importantly delivers software of greater business value. It is my assertion that scaled, enterprise-level agile is therefore not just important for your software-delivery organization but crucial for business success. Fixing water-scrum-fall will increase business value and enable organizations to compete. And this book provides a framework to make that happen.

In this book, Scott and Mark, two very experienced software-delivery change agents, describe a detailed framework for how to scale agile to the enterprise. They show how change leaders can amplify agile, making it not just about teams but about the whole value stream of software delivery. In many books about agile adoption, the really tricky problems associated with governance and organizational control are often side-stepped, focusing on why it is stupid to do something rather than how to change that something. Scott and Mark have not done this. They have focused clearly on the gnarly problems of scale, describing practical ways of fixing governance models, staffing issues, and management approaches. Their use of lean positions their

framework in a broader context, allowing change leaders to not only improve their delivery capability but also connect it directly to business value. But be warned: These problems are not easily solved, and adopting these ideas does not just require agile skills but also draws on other process models, change techniques, and good engineering practices.

Scott and Mark not only made me think, but they also reminded me of lots of things that I had forgotten—things that the agile fashion police have made uncool to talk about. This book is not about fashionable agile; it is about serious change, and it should be required reading for any change leader.

Dave West @davidjwest
Chief Product Officer, Tasktop, and former VP and Research Director, Forrester Research

Preface

The information technology (IT) industry has an embarrassing reputation from the perspective of our customers. For decades we have squandered scarce budgets and resources, reneged on our promises, and delivered functionality that is not actually needed by the client. An outsider looking at our profession must be truly baffled. We have so many process frameworks and various bodies of knowledge such that we ourselves have difficulty keeping up with just the acronyms, let alone the wealth of material behind them. Consider: PMBOK, SWEBOK, BABOK, ITIL®, COBIT, RUP, CMMI, TOGAF, DODAF, EUP, UML, and BPMN, to name a few. Even within the narrow confines of the agile community, we have Scrum, XP, CI, CD, FDD, AMDD, TDD, and BDD, and many others. There is considerable overlap between these strategies but also considerable differences. We really need to get our act together.

Why Agile?

On traditional/classical projects, and sadly even on "heavy RUP" projects, basic business and system requirements often end up written in multiple documents in different fashions to suit the standards of the various standards bodies. Although in some regulatory environments this proves to be good practice, in many situations it proves to be a huge waste of time and effort that often provides little ultimate value—you must tailor your approach to meet the needs of your situation.

Fortunately, agile methods have surfaced over the past decade so that we can save ourselves from this madness. The beauty of agile methods is that they focus us on delivering working software of high business value to our customers early and often. We are free to adjust the project objectives at any time as the business needs change. We are encouraged to minimize documentation, to minimize if not eliminate the bureaucracy in general. Who doesn't like that?

More importantly, agile strategies seem to be working in practice. Scott has run surveys[1] within the IT industry for several years now, and he has consistently found that the agile and iterative strategies to software development have consistently outperformed both traditional and ad-hoc strategies. There's still room for improvement, and this book makes many suggestions for such improvements, but it seems clear that agile is a step in the right direction. For example, the 2011 IT Project Success Survey revealed that respondents felt that 67% of agile projects were considered successful (they met all of their success criteria), 27% were considered challenged (they delivered but didn't meet all success criteria), and only 6% were considered failures. The same survey showed that 50% of traditional projects were considered successful, 36% challenged, and 14% failures. The 2008 IT Project Success survey found that agile project teams were much more adept at delivering quality solutions, good return on investment (ROI), and solutions that stakeholders wanted to work with and did so faster than traditional teams. Granted, these are averages and your success at agile may vary, but they are compelling results. We're sharing these numbers with you now to motivate you to take agile seriously but, more importantly, to illustrate a common theme throughout this book: We do our best to shy away from the overly zealous "religious" discussions found in many software process books and instead strive to have fact-based discussions backed up by both experiential and research-based evidence. There are still some holes in the evidence because research is ongoing, but we're far past the "my process can beat up your process" arguments we see elsewhere.

Alistair Cockburn, one of the original drafters of the Agile Manifesto, has argued that there are three primary aspects of agile methodologies:

- Self-discipline, with Extreme Programming (XP) being the exemplar methodology
- Self-organization, with Scrum being the exemplar methodology
- Self-awareness, with Crystal being the exemplar methodology

As you'll see in this book, Disciplined Agile Delivery (DAD) addresses Cockburn's three aspects.

Why Disciplined Agile Delivery?

Although agile strategies appear to work better than traditional strategies, it has become clear to us that the pendulum has swung too far the other way. We have gone from overly bureaucratic and document-centric processes to almost nothing but code. To be fair, agile teams do invest in planning, although they are unlikely to create detailed plans; they do invest in modeling, although are unlikely to create detailed models; they do create deliverable documentation (such as operations manuals and system overview documents), although are unlikely to create detailed specifications. However, agile teams have barely improved upon the results of iterative approaches. The 2011 IT

1. The original questions, source data (without identifying information due to privacy concerns), and summary slide decks for all surveys can be downloaded free of charge from www.ambysoft.com/surveys/.

Project Success survey found that 69% of iterative projects were considered successful, 25% challenged, and 6% failures, statistically identical results as agile projects. Similarly, the 2008 IT Project Success survey found that agile and iterative teams were doing statistically the same when it came to quality, ability to deliver desired functionality, and timeliness of delivery and that agile was only slightly better than iterative when it came to ROI. The reality of agile hasn't lived up to the rhetoric, at least when we compare agile strategies with iterative strategies. The good news is that it is possible to do better.

Our experience is that "core" agile methods such as Scrum work wonderfully for small project teams addressing straightforward problems in which there is little risk or consequence of failure. However, "out of the box," these methods do not give adequate consideration to the risks associated with delivering solutions on larger enterprise projects, and as a result we're seeing organizations investing a lot of effort creating hybrid methodologies combining techniques from many sources. The Disciplined Agile Delivery (DAD) process framework, as described in this book, is a hybrid approach which extends Scrum with proven strategies from Agile Modeling (AM), Extreme Programming (XP), and Unified Process (UP), amongst other methods. DAD extends the construction-focused lifecycle of Scrum to address the full, end-to-end delivery lifecycle[2] from project initiation all the way to delivering the solution to its end users. The DAD process framework includes advice about the technical practices purposely missing from Scrum as well as the modeling, documentation, and governance strategies missing from both Scrum and XP. More importantly, in many cases DAD provides advice regarding viable alternatives and their trade-offs, enabling you to tailor DAD to effectively address the situation in which you find yourself. By describing what works, what doesn't work, and more importantly why, DAD helps you to increase your chance of adopting strategies that will work for you.

Indeed there are an increasing number of high-profile project failures associated with agile strategies that are coming to light. If we don't start supplementing core agile practices with a more disciplined approach to agile projects at scale, we risk losing the hard-earned momentum that the agile pioneers have generated.

This book does not attempt to rehash existing agile ideas that are described in many other books, examples of which can be found in the references sections. Rather, this book is intended to be a practical guide to getting started today with agile practices that are structured within a disciplined approach consistent with the needs of enterprise-scale, mission-critical projects.

What Is the History?

The Disciplined Agile Delivery (DAD) process framework began as a concept in 2007 that Scott worked on in his role as chief methodologist for agile and lean at IBM® Rational®. He was working with customers around the world to understand and apply agile techniques at scale, and he

2. A full system/product lifecycle goes from the initial idea for the product, through delivery, to operations and support and often has many iterations of the delivery lifecycle. Our focus in DAD is on delivery, although we discuss how the other aspects of the system lifecycle affect the delivery lifecycle.

noticed time and again that organizations were struggling to adopt mainstream agile methods such as Extreme Programming (XP) and Scrum. At the same time Mark, also working with organizations to adopt and apply agile techniques in practice, observed the same problems. In many cases, the organization's existing command-and-control culture hampered its adoption of these more chaordic techniques. Furthermore, although many organizations were successful at agile pilot projects, they struggled to roll out agile strategies beyond these pilot teams. A common root cause was that the methods did not address the broader range of issues faced by IT departments, let alone the broader organization. Something wasn't quite right.

Separately we began work on addressing these problems, with Scott taking a broad approach by observing and working with dozens of organizations and Mark taking a deep approach through long-term mentoring of agile teams at several organizations. In 2009 Scott led the development of the DAD process framework within IBM Rational, an effort that continues to this day. This work included the development of DAD courseware, whitepapers, and many blog postings on IBM developerWorks®.[3]

What About Lean?

There are several reasons why lean strategies are crucial for DAD:

- Lean provides insights for streamlining the way that DAD teams work.
- Lean provides a solid foundation for scaling DAD to address complex situations, a topic we touch on throughout the book but intend to address in greater detail in a future book.
- Lean principles explain why agile practices work, a common theme throughout this book.
- Lean strategies, particularly those encapsulated by Kanban, provide an advanced adoption strategy for DAD.

So why aren't we writing about Disciplined Lean Development (DLD) instead? Our experience is that lean strategies, as attractive and effective as they are, are likely beyond all but a small percentage of teams at this time. Perhaps this "small" percentage is 10% to 15%—it's certainly under 20%—but only time will tell. We've found that most development teams are better served with a lightweight, end-to-end process framework that provides coherent and integrated high-level advice for how to get the job done without getting bogged down in procedural details. Having said that, many of the options that we describe for addressing the goals of the DAD process framework are very clearly lean in nature, and we expect that many teams will evolve their process from a mostly agile one to a mostly lean one over time.

DAD is the happy medium between the extremes of Scrum, a lightweight process framework that focuses on only a small part of the delivery process, and RUP, a comprehensive process framework that covers the full delivery spectrum. DAD addresses the fundamentals of agile

3. https://www.ibm.com/developerworks/mydeveloperworks/blogs/ambler/

delivery while remaining flexible enough for you to tailor it to your own environment. In many ways, Scrum taught agilists how to crawl, DAD hopes to teach agilists how to walk, and agility@scale and lean approaches such as Kanban will teach us how to run.

How Does This Book Help?

We believe that there are several ways that you'll benefit from reading this book:

- It describes an end-to-end agile delivery lifecycle.
- It describes common agile practices, how they fit into the lifecycle, and how they work together.
- It describes how agile teams work effectively within your overall organization in an "enterprise aware" manner, without assuming everyone else is going to be agile, too.
- It uses consistent, sensible terminology but also provides a map to terminology used by other methods.
- It explains the trade-offs being made and in many cases gives you options for alternative strategies.
- It provides a foundation from which to scale your agile strategy to meet the real-world situations faced by your delivery teams.
- It goes beyond anecdotes to give fact-based reasons for why these techniques work.
- It really does answer the question "how do all these agile techniques fit together?"

Where Are We Coming From?

Both of us have seen organizations adopt Scrum and extend it with practices from XP, Agile Modeling, and other sources into something very similar to DAD or to tailor down the Unified Process into something similar to DAD. With either strategy, the organizations invested a lot of effort that could have been easily avoided. With DAD, we hope to help teams and organizations avoid the expense of a lengthy trial-and-error while still enabling teams to tailor the approach to meet their unique situation.

Scott led the development of DAD within IBM Rational and still leads its evolution, leveraging his experiences helping organizations understand and adopt agile strategies. This book also reflects lessons learned from within IBM Software Group, a diverse organization of 27,000 developers worldwide, and IBM's Agile with Discipline (AwD) methodology followed by professionals in IBM Global Service's Accelerated Solution Delivery (ASD) practice. In the autumn of 2009 DAD was captured in IBM Rational's three-day "Introduction to Disciplined Agile Delivery" workshop. This workshop was rolled out in the first quarter of 2010 to IBM business partners, including UPMentors, and Mark became one of the first non-IBMers to be qualified to deliver the workshop. Since then, Mark has made significant contributions to DAD, bringing his insights and experiences to bear.

What's The Best Way to Read this Book?

Most people will want to read this cover to cover. However, there are three exceptions:

- Experienced agile practitioners can start with Chapter 1, "Disciplined Agile Delivery in a Nutshell," which overviews DAD. Next, read Chapter 4, "Roles, Rights, and Responsibilities," to understand the team roles. Then, read Chapters 6 through 19 to understand in detail how DAD works.
- Senior IT managers should read Chapter 1 to understand how DAD works at a high level and then skip to Chapter 20, "Governing Disciplined Agile Teams," which focuses on governing[4] agile teams.
- People who prefer to work through an example of DAD in practice should read the case study chapters first. These are: Chapter 12, "Initiating a Disciplined Agile Delivery Project—Case Study"; Chapter 17, "Case Study: Construction Phase"; and Chapter 19, "Case Study: Transition Phase."

We hope that you embrace the core agile practices popularized by leading agile methods but choose to supplement them with some necessary rigor and tooling appropriate for your organization and project realities.

Incidentally, a portion of the proceeds from the sale of this book are going to the Cystic Fibrosis Foundation and Toronto Sick Kid's Hospital, so thank you for supporting these worthy causes.

The Disciplined Agile Delivery Web Site

www.DisciplinedAgileDelivery.com is the community Web site for anything related to DAD. Mark and Scott are the moderators. You will also find other resources such as information on DAD-related education, service providers, and supporting collateral that can be downloaded. We invite anyone who would like to contribute to DAD to participate as a blogger. Join the discussion!

4. Warning: Throughout the book we'll be using "agile swear words" such as governance, management, modeling, and yes, even the D-word—documentation. We'd like to apologize now for our use of foul language such as this.

Abbreviations Used in This Book

AD	Agile Data
AM	Agile Modeling
AMDD	Agile Model Driven Development
ASM	Agile Scaling Model
ATDD	Acceptance test driven development
AUP	Agile Unified Process
AwD	Agile with Discipline
BABOK	Business Analysis Book of Knowledge
BDD	Behavior driven development
BI	Business intelligence
BPMN	Business Process Modeling Notation
CASE	Computer aided software engineering
CD	Continuous deployment
CI	Continuous integration
CM	Configuration management
CMMI	Capability Maturity Model Integrated
COBIT	Control Objectives for Information and Related Technology
DAD	Disciplined Agile Delivery
DDJ	Dr. Dobb's Journal
DevOps	Development operations
DI	Development intelligence
DODAF	Department of Defense Architecture Framework
DSDM	Dynamic System Development Method
EUP	Enterprise Unified Process
EVM	Earned value management
FDD	Feature Driven Development
GQM	Goal question metric
HR	Human resources
IT	Information technology
ITIL	Information Technology Infrastructure Library
JIT	Just in time

MDD	Model driven development
MMR	Minimally marketable release
NFR	Non-functional requirement
NPV	Net present value
OSS	Open source software
PMBOK	Project Management Book of Knowledge
PMO	Project management office
ROI	Return on investment
RRC	Rational Requirements Composer
RSA	Rational Software Architect
RTC	Rational Team Concert™
RUP	Rational Unified Process
SCM	Software configuration management
SDLC	System development lifecycle
SLA	Service level agreement
SWEBOK	Software Engineering Book of Knowledge
TCO	Total cost of ownership
TDD	Test-driven development
TFD	Test first development
TOGAF	The Open Group Architecture Framework
T&M	Time and materials
TVO	Total value of ownership
UAT	User acceptance testing
UML	Unified Modeling Language
UI	User interface
UP	Unified Process
UX	User experience
WIP	Work in progress
XP	Extreme Programming

Acknowledgments

We'd like to thank the following people for their feedback regarding this book: Kevin Aguanno, Brad Appleton, Ned Bader, Joshua Barnes, Peter Bauwens, Robert Boyle, Alan L. Brown, David L. Brown, Murray Cantor, Nick Clare, Steven Crago, Diana Dehm, Jim Densmore, Paul Gorans, Leslie R. Gornig, Tony Grout, Carson Holmes, Julian Holmes, Mark Kennaley, Richard Knaster, Per Kroll, Cherifa Liamani, Christophe Lucas, Bruce MacIsaac, Trevor O. McCarthy, M.K. McHugh, Jean-Louise Marechaux, Evangelos Mavrogiannakis, Brian Merzbach, Berne C. Miller, Mike Perrow, Andy Pittaway, Emily J. Ratliff, Oliver Roehrsheim, Walker Royce, Chris Sibbald, Lauren Schaefer, Paul Sims, Paula Stack, Alban Tsui, Karthikeswari Vijayapandian, Lewis J. White, Elizabeth Woodward, and Ming Zhi Xie.

We'd also like to thank the following people for their ideas shared with us in online forums, which were incorporated into this book: Eric Jan Malotaux, Bob Marshall, Valentin Tudor Mocanu, Allan Shalloway, Steven Shaw, Horia Slusanschi, and Marvin Toll.

About the Authors

Scott W. Ambler is Chief Methodologist for IT with IBM Rational, working with IBM customers around the world to help them to improve their software processes. In addition to Disciplined Agile Delivery (DAD), he is the founder of the Agile Modeling (AM), Agile Data (AD), Agile Unified Process (AUP), and Enterprise Unified Process (EUP) methodologies and creator of the Agile Scaling Model (ASM). Scott is the (co-)author of 20 books, including *Refactoring Databases*, *Agile Modeling*, *Agile Database Techniques*, *The Object Primer,* 3rd Edition, and *The Enterprise Unified Process*. Scott is a senior contributing editor with *Dr. Dobb's Journal*. His personal home page is www.ambysoft.com.

Mark Lines co-founded UPMentors in 2007. He is a disciplined agile coach and mentors organizations on all aspects of software development. He is passionate about reducing the huge waste in most IT organizations and demonstrates hands-on approaches to speeding execution and improving quality with agile and lean techniques. Mark provides IT assessments and executes course corrections to turn around troubled projects. He writes for many publications and is a frequent speaker at industry conferences. Mark is also an instructor of IBM Rational and UPMentors courses on all aspects of software development. His Web site is www.UPMentors.com. Mark can be reached at Mark@UPMentors.com.

CHAPTER 1

Disciplined Agile Delivery in a Nutshell

For every complex problem there is a solution that is simple, neat, and wrong. —H L Mencken

The agile software development paradigm burst onto the scene in the spring of 2001 with the publication of the Agile Manifesto (www.agilemanifesto.org). The 17 authors of the manifesto captured strategies, in the form of four value statements and twelve supporting principles, which they had seen work in practice. These strategies promote close collaboration between developers and their stakeholders; evolutionary and regular creation of software that adds value to the organization; remaining steadfastly focused on quality; adopting practices that provide high value and avoiding those which provide little value (e.g., work smarter, not harder); and striving to improve your approach to development throughout the lifecycle. For anyone with experience on successful software development teams, these strategies likely sound familiar.

Make no mistake, agile is not a fad. When mainstream agile methods such as Scrum and Extreme Programming (XP) were introduced, the ideas contained in them were not new, nor were they even revolutionary at the time. In fact, many of them have been described in-depth in other methods such as Rapid Application Development (RAD), Evo, and various instantiations of the Unified Process, not to mention classic books such as Frederick Brooks' *The Mythical Man Month*. It should not be surprising that working together closely in collocated teams and collaborating in a unified manner toward a goal of producing working software produces results superior to those working in specialized silos concerned with individual rather than team performance. It should also come as no surprise that reducing documentation and administrative bureaucracy saves money and speeds up delivery.

While agile was once considered viable only for small, collocated teams, improvements in product quality, team efficiency, and on-time delivery have motivated larger teams to take a closer look at adopting agile principles in their environments. In fact, IBM has teams of several hundred

people, often distributed around the globe, that are working on complex products who are applying agile techniques—and have been doing so successfully for years. A recent study conducted by the *Agile Journal* determined that 88% of companies, many with more than 10,000 employees, are using or evaluating agile practices on their projects. Agile is becoming the dominant software development paradigm. This trend is also echoed in other industry studies, including one conducted by *Dr. Dobb's Journal (DDJ)*, which found a 76% adoption rate of agile techniques, and within those organizations doing agile, 44% of the project teams on average are applying agile techniques in some way.

Unfortunately, we need to take adoption rate survey results with a grain of salt: A subsequent *Ambysoft* survey found that only 53% of people claiming to be on "agile teams" actually were. It is clear that agile methods have been overly hyped by various media over the years, leading to abuse and misuse; in fact, the received message regarding agile appears to have justified using little or no process at all. For too many project teams this resulted in anarchy and chaos, leading to project failures and a backlash from the information technology (IT) and systems engineering communities that prefer more traditional approaches.

Properly executed, agile is not an excuse to be undisciplined. The execution of mainstream agile methods such as XP for example have always demanded a disciplined approach, certainly more than traditional approaches such as waterfall methods. Don't mistake the high ceremony of many traditional methods to be a sign of discipline, rather it's a sign of rampant and often out-of-control bureaucracy. However, mainstream agile methods don't provide enough guidance for typical enterprises. Mature implementations of agile recognize a basic need in enterprises for a level of rigor that core agile methods dismiss as not required such as governance, architectural planning, and modeling. Most mainstream agile methods admit that their strategies require significant additions and adjustments to scale beyond teams of about eight people who are working together in close proximity. Furthermore, most Fortune 1000 enterprises and government agencies have larger solution delivery teams that are often distributed, so the required tailoring efforts can prove both expensive and risky. The time is now for a new generation of agile process framework.

Figure 1.1 shows a mind map of the structure of this chapter. We describe each of the topics in the map in clockwise order, beginning at the top right.

The Big Ideas in This Chapter

- People are the primary determinant of success for IT delivery projects.
- Moving to a disciplined agile delivery process is the first step in scaling agile strategies.
- Disciplined Agile Delivery (DAD) is an enterprise-aware hybrid software process framework.
- Agile strategies should be applied throughout the entire delivery lifecycle.
- Agile teams are easier to govern than traditional teams.

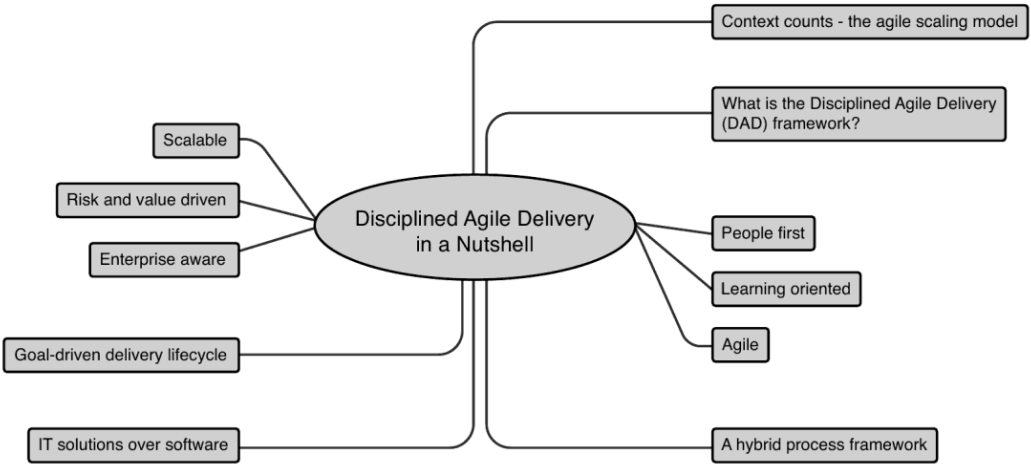

Figure 1.1 Outline of this chapter

Context Counts—The Agile Scaling Model

To understand the need for the Disciplined Agile Delivery (DAD) process framework you must start by recognizing the realities of the situation you face. The Agile Scaling Model (ASM) is a contextual framework that defines a roadmap to effectively adopt and tailor agile strategies to meet the unique challenges faced by an agile software development team. The first step to scaling agile strategies is to adopt a disciplined agile delivery lifecycle that scales mainstream agile construction strategies to address the full delivery process from project initiation to deployment into production. The second step is to recognize which scaling factors, if any, are applicable to your project team and then tailor your adopted strategies to address the range of complexities the team faces.

The ASM, depicted in Figure 1.2, defines three process categories:

1. **Core agile development.** Core agile methods—such as Scrum, XP, and Agile Modeling (AM)—focus on construction-oriented activities. They are characterized by value-driven lifecycles where high-quality potentially shippable software is produced on a regular basis by a highly collaborative, self-organizing team. The focus is on small (<15 member) teams that are collocated and are developing straightforward software.

2. **Agile delivery.** These methods—including the DAD process framework (described in this book) and Harmony/ESW—address the full delivery lifecycle from project initiation to production. They add appropriate, lean governance to balance self-organization and add a risk-driven viewpoint to the value-driven approach to increase the chance of project success. They focus on small-to-medium sized (up to 30 people) near-located teams (within driving distance) developing straightforward solutions. Ideally DAD teams are small and collocated.
3. **Agility@scale.** This is disciplined agile development where one or more scaling factors apply. The scaling factors that an agile team may face include team size, geographical distribution, organizational distribution (people working for different groups or companies), regulatory compliance, cultural or organizational complexity, technical complexity, and enterprise disciplines (such as enterprise architecture, strategic reuse, and portfolio management).

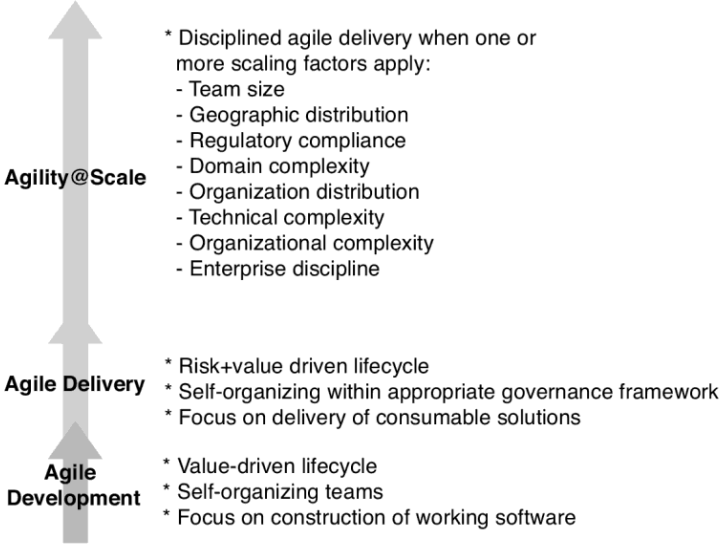

Figure 1.2 The Agile Scaling Model (ASM)

This book describes the DAD process framework. In most cases we assume that your team is small (<15 people) and is either collocated or near-located (within driving distance). Having

said that, we also discuss strategies for scaling agile practices throughout the book. The DAD process framework defines the foundation to scale agile strategies to more complex situations.

What Is the Disciplined Agile Delivery (DAD) Process Framework?

Let's begin with a definition:

> The Disciplined Agile Delivery (DAD) process framework is a people-first, learning-oriented hybrid agile approach to IT solution delivery. It has a risk-value lifecycle, is goal-driven, is scalable, and is enterprise aware.

From this definition, you can see that the DAD process framework has several important characteristics:

- People first
- Learning oriented
- Agile
- Hybrid
- IT solution focused
- Goal-driven
- Delivery focused
- Enterprise aware
- Risk and value driven
- Scalable

To gain a better understanding of DAD, let's explore each of these characteristics in greater detail.

People First

Alistair Cockburn refers to people as "non-linear, first-order components" in the software development process. His observation, based on years of ethnographic work, is that people and the way that they collaborate are the primary determinants of success in IT solution delivery efforts. This philosophy, reflected in the first value statement of the Agile Manifesto, permeates DAD. DAD team members should be self-disciplined, self-organizing, and self-aware. The DAD process framework provides guidance that DAD teams leverage to improve their effectiveness, but it does not prescribe mandatory procedures.

The traditional approach of having formal handoffs of work products (primarily documents) between different disciplines such as requirements, analysis, design, test, and development is a very poor way to transfer knowledge that creates bottlenecks and proves in practice to

be a huge source of waste of both time and money. The waste results from the loss of effort to create interim documentation, the cost to review the documentation, and the costs associated with updating the documentation. Yes, some documentation will be required, but rarely as much as is promoted by traditional techniques. Handoffs between people provide opportunities for misunderstandings and injection of defects and are described in lean software development as one of seven sources of waste. When we create a document we do not document our complete understanding of what we are describing, and inevitably some knowledge is "left behind" as tacit knowledge that is not passed on. It is easy to see that after many handoffs the eventual deliverable may bear little resemblance to the original intent. In an agile environment, the boundaries between disciplines should be torn down and handoffs minimized in the interest of working as a team rather than specialized individuals.

In DAD we foster the strategy of cross-functional teams made up of cross-functional people. There should be no hierarchy within the team, and team members are encouraged to be cross-functional in their skillset and indeed perform work related to disciplines other than their specialty. The increased understanding that the team members gain beyond their primary discipline results in more effective use of resources and reduced reliance on formal documentation and handoffs.

As such, agile methods deemphasize specific roles. In Scrum for instance, there are only three Scrum team roles: ScrumMaster, product owner, and team member. Nonteam roles can be extended to stakeholder and manager. The primary roles described by DAD are stakeholder, team lead, team member, product owner, and architecture owner. These roles are described in detail in Chapter 4, "Roles, Rights, and Responsibilities."

Notice that tester and business analyst are not primary roles in the DAD process framework. Rather, a generic team member should be capable of doing multiple things. A team member who specializes in testing might be expected to volunteer to help with requirements, or even take a turn at being the ScrumMaster (team lead). This doesn't imply that everyone needs to be an expert at everything, but it does imply that the team as a whole should cover the skills required of them and should be willing to pick up any missing skills as needed. However, as you learn in Chapter 4, DAD also defines several secondary roles often required in scaling situations.

Team members are often "generalizing specialists" in that they may be a specialist in one or more disciplines but should have general knowledge of other disciplines as well. More importantly, generalizing specialists are willing to collaborate closely with others, to share their skills and experiences with others, and to pick up new skills from the people they work with. A team made up of generalizing specialists requires few handoffs between people, enjoys improved collaboration because the individuals have a greater appreciation of the background skills and priorities of the various IT disciplines, and can focus on what needs to be done as opposed to focusing on whatever their specialties are.

However, there is still room for specialists. For example, your team may find that it needs to set up and configure a database server. Although you could figure it out yourselves, it's probably easier, faster, and less expensive if you could have someone with deep experience help your team

for a few days to work with you to do so. This person could be a specialist in database administration. In scaling situations you may find that your build becomes so complex that you need someone(s) specifically focused on doing just that. Or you may bring one or more business analyst specialists onto the team to help you explore the problem space in which you're working.

DAD teams and team members should be

- Self-disciplined in that they commit only to the work they can accomplish and then perform that work as effectively as possible
- Self-organizing, in that they estimate and plan their own work and then proceed to collaborate iteratively to do so
- Self-aware, in that they strive to identify what works well for them, what doesn't, and then learn and adjust accordingly

Although people are the primary determinant of success for IT solution delivery projects, in most situations it isn't effective to simply put together a good team of people and let them loose on the problem at hand. If you do this then the teams run several risks, including investing significant time in developing their own processes and practices, ramping up on processes or practices that more experienced agile teams have discovered are generally less effective or efficient, and not adapting their own processes and practices effectively. We can be smarter than that and recognize that although people are the primary determinant of success they aren't the only determinant. The DAD process framework provides coherent, proven advice that agile teams can leverage and thereby avoid or at least minimize the risks described previously.

Learning Oriented

In the years since the Agile Manifesto was written we've discovered that the most effective organizations are the ones that promote a learning environment for their staff. There are three key aspects that a learning environment must address. The first aspect is domain learning—how are you exploring and identifying what your stakeholders need, and perhaps more importantly how are you helping the team to do so? The second aspect is process learning, which focuses on learning to improve your process at the individual, team, and enterprise levels. The third aspect is technical learning, which focuses on understanding how to effectively work with the tools and technologies being used to craft the solution for your stakeholders.

The DAD process framework suggests several strategies to support domain learning, including initial requirements envisioning, incremental delivery of a potentially consumable solution, and active stakeholder participation through the lifecycle. To support process-focused learning DAD promotes the adoption of retrospectives where the team explicitly identifies potential process improvements, a common agile strategy, as well as continued tracking of those improvements. Within the IBM software group, a business unit with more than 35,000 development professionals responsible for delivering products, we've found that agile teams that held retrospectives improved their productivity more than teams that didn't, and teams that tracked

their implementation of the identified improvement strategies were even more successful. Technical learning often comes naturally to IT professionals, many of whom are often eager to work with and explore new tools, techniques, and technologies. This can be a double-edged sword—although they're learning new technical concepts they may not invest sufficient time to master a strategy before moving on to the next one or they may abandon a perfectly fine technology simply because they want to do something new.

There are many general strategies to improve your learning capability. Improved collaboration between people correspondingly increases the opportunities for people to learn from one another. Luckily high collaboration is a hallmark of agility. Investing in training, coaching, and mentoring are obvious learning strategies as well. What may not be so obvious is the move away from promoting specialization among your staff and instead fostering a move toward people with more robust skills, something called being a generalizing specialist (discussed in greater detail in Chapter 4). Progressive organizations aggressively promote learning opportunities for their people outside their specific areas of specialty as well as opportunities to actually apply these new skills.

If you're experienced with, or at least have read about, agile software development, the previous strategies should sound familiar. Where the DAD process framework takes learning further is through enterprise awareness. Core agile methods such as Scrum and XP are typically project focused, whereas DAD explicitly strives to both leverage and enhance the organizational ecosystem in which a team operates. So DAD teams should both leverage existing lessons learned from other agile teams and also take the time to share their own experiences. The implication is that your IT department needs to invest in a technology for socializing the learning experience across teams. In 2005 IBM Software Group implemented internal discussion forums, wikis, and a center of competency (some organizations call them centers of excellence) to support their agile learning efforts. A few years later they adopted a Web 2.0 strategy based on IBM Connections to support enterprise learning. When the people and teams within an organization choose a learning-oriented approach, providing them with the right tools and support can increase their success.

Agile

The DAD process framework adheres to, and as you learn in Chapter 2, "Introduction to Agile and Lean," enhances, the values and principles of the Agile Manifesto. Teams following either iterative or agile processes have been shown to produce higher quality solutions, provide greater return on investment (ROI), provide greater stakeholder satisfaction, and deliver these solutions quicker as compared to either a traditional/waterfall approach or an ad-hoc (no defined process) approach. High quality is achieved through techniques such as continuous integration (CI), developer regression testing, test-first development, and refactoring—these techniques, and more, are described later in the book. Improved ROI comes from a greater focus on high-value activities, working in priority order, automation of as much of the IT drudgery as possible, self-

organization, close collaboration, and in general working smarter not harder. Greater stakeholder satisfaction is increased through enabling active stakeholder participation, by incrementally delivering a potentially consumable solution each iteration, and by enabling stakeholders to evolve their requirements throughout the project.

A Hybrid Process Framework

DAD is the formulation of many strategies and practices from both mainstream agile methods as well as other sources. The DAD process framework extends the Scrum construction lifecycle to address the full delivery lifecycle while adopting strategies from several agile and lean methods. Many of the practices suggested by DAD are the ones commonly discussed in the agile community—such as continuous integration (CI), daily coordination meetings, and refactoring—and some are the "advanced" practices commonly applied but for some reason not commonly discussed. These advanced practices include initial requirements envisioning, initial architecture envisioning, and end-of-lifecycle testing to name a few.

The DAD process framework is a hybrid, meaning that it adopts and tailors strategies from a variety of sources. A common pattern that we've seen time and again within organizations is that they adopt the Scrum process framework and then do significant work to tailor ideas from other sources to flesh it out. This sounds like a great strategy. However, given that we repeatedly see new organizations tailoring Scrum in the same sort of way, why not start with a robust process framework that provides this common tailoring in the first place? The DAD process framework adopts strategies from the following methods:

- **Scrum.** Scrum provides an agile project management framework for complex projects. DAD adopts and tailors many ideas from Scrum, such as working from a stack of work items in priority order, having a product owner responsible for representing stakeholders, and producing a potentially consumable solution every iteration.
- **Extreme Programming (XP).** XP is an important source of development practices for DAD, including but not limited to continuous integration (CI), refactoring, test-driven development (TDD), collective ownership, and many more.
- **Agile Modeling (AM).** As the name implies, AM is the source for DAD's modeling and documentation practices. This includes requirements envisioning, architecture envisioning, iteration modeling, continuous documentation, and just-in-time (JIT) model storming.
- **Unified Process (UP).** DAD adopts many of its governance strategies from agile instantiations of the UP, including OpenUP and Agile Unified Process (AUP). In particular these strategies include having lightweight milestones and explicit phases. We also draw from the Unified Process focus on the importance of proving that the architecture works in the early iterations and reducing much of the business risk early in the lifecycle.

- **Agile Data (AD).** As the name implies AD is a source of agile database practices, such as database refactoring, database testing, and agile data modeling. It is also an important source of agile enterprise strategies, such as how agile teams can work effectively with enterprise architects and enterprise data administrators.
- **Kanban.** DAD adopts two critical concepts—limiting work in progress and visualizing work—from Kanban, which is a lean framework. These concepts are in addition to the seven principles of lean software development, as discussed in Chapter 2.

The concept of DAD being a hybrid of several existing agile methodologies is covered in greater detail in Chapter 3, "Foundations of Disciplined Agile Delivery."

Our Apologies

Throughout this book we'll be applying agile swear words such as *phase*, *serial*, and yes, even the "G word"—*governance*. Many mainstream agilists don't like these words and have gone to great lengths to find euphemisms for them. For example, in Scrum they talk about how a project begins with Sprint 0 (DAD's Inception phase), then the construction sprints follow, and finally you do one or more hardening/release sprints (DAD's Transition phase). Even though these sprint categories follow one another this clearly isn't serial, and the Scrum project team clearly isn't proceeding in phases. Or so goes the rhetoric. Sigh. We prefer plain, explicit language.

IT Solutions over Software

One aspect of adopting a DAD approach is to mature your focus from producing software to instead providing solutions that provide real business value to your stakeholders within the appropriate economic, cultural, and technical constraints. A fundamental observation is that as IT professionals we do far more than just develop software. Yes, software is clearly important, but in addressing the needs of our stakeholders we often provide new or upgraded hardware, change the business/operational processes that stakeholders follow, and even help change the organizational structure in which our stakeholders work.

This shift in focus requires your organization to address some of the biases that crept into the Agile Manifesto. The people who wrote the manifesto (which we fully endorse) were for the most part software developers, consultants, and in many cases both. It was natural that they focused on their software development strengths, but as the ten-year agile anniversary workshop (which Scott participated in) identified, the agile community needs to look beyond software development.

It's also important to note that the focus of this book is on IT application development. The focus is not on product development, even though a tailored form of DAD is being applied for

that within IBM, nor is it on systems engineering. For agile approaches to embedded software development or systems engineering we suggest you consider the IBM Harmony process framework.

Goal-Driven Delivery Lifecycle

DAD addresses the project lifecycle from the point of initiating the project to construction to releasing the solution into production. We explicitly observe that each iteration is *not* the same. Projects do evolve and the work emphasis changes as we move through the lifecycle. To make this clear, we carve the project into phases with lightweight milestones to ensure that we are focused on the right things at the right time. Such areas of focus include initial visioning, architectural modeling, risk management, and deployment planning. This differs from mainstream agile methods, which typically focus on the construction aspects of the lifecycle. Details about how to perform initiation and release activities, or even how they fit into the overall lifecycle, are typically vague and left up to you.

Time and again, whenever either one of us worked with a team that had adopted Scrum we found that they had tailored the Scrum lifecycle into something similar to Figure 1.3, which shows the lifecycle of a DAD project.[1] This lifecycle has several critical features:

- **It's a delivery lifecycle.** The DAD lifecycle extends the Scrum construction lifecycle to explicitly show the full delivery lifecycle from the beginning of a project to the release of the solution into production (or the marketplace).
- **There are explicit phases.** The DAD lifecycle is organized into three distinct, named phases, reflecting the agile coordinate-collaborate-conclude (3C) rhythm.
- **The delivery lifecycle is shown in context.** The DAD lifecycle recognizes that activities occur to identify and select projects long before their official start. It also recognizes that the solution produced by a DAD project team must be operated and supported once it is delivered into production (in some organizations called operations) or in some cases the marketplace, and that important feedback comes from people using previously released versions of the solution.
- **There are explicit milestones.** The milestones are an important governance and risk reduction strategy inherent in DAD.

The lifecycle of Figure 1.3, which we focus on throughout this book, is what we refer to as the basic agile version. This is what we believe should be the starting point for teams that are new to DAD or even new to agile. However, DAD is meant to be tailored to meet the needs of your situation. As your team gains more experience with DAD you may choose to adopt more and more lean strategies, and may eventually evolve your lifecycle into something closer to what you see in

1. Granted, in this version we're using the term "iteration" instead of "sprint," and "work item list" instead of "product backlog."

Figure 1.4. A primary difference of this lean version of the DAD lifecycle is that the phase and iteration cadence disappears in favor of a "do it when you need to do it" approach, a strategy that works well only for highly disciplined teams.

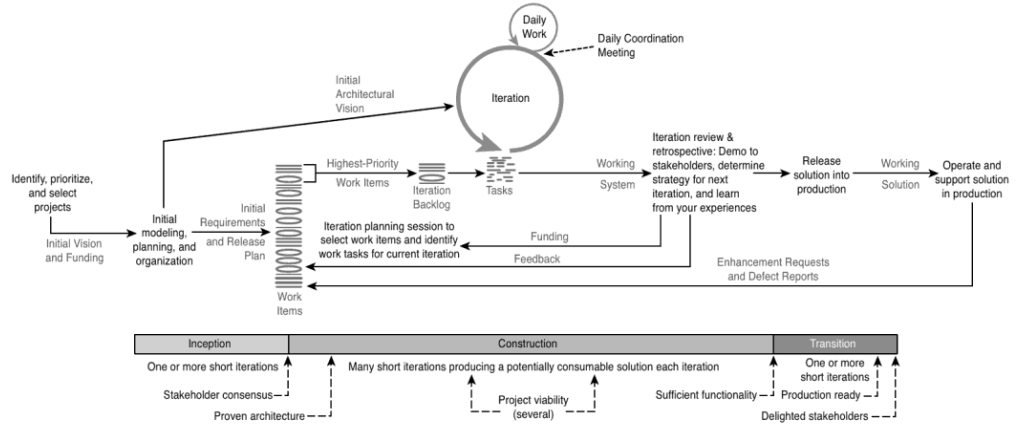

Figure 1.3 The Disciplined Agile Delivery (DAD) lifecycle

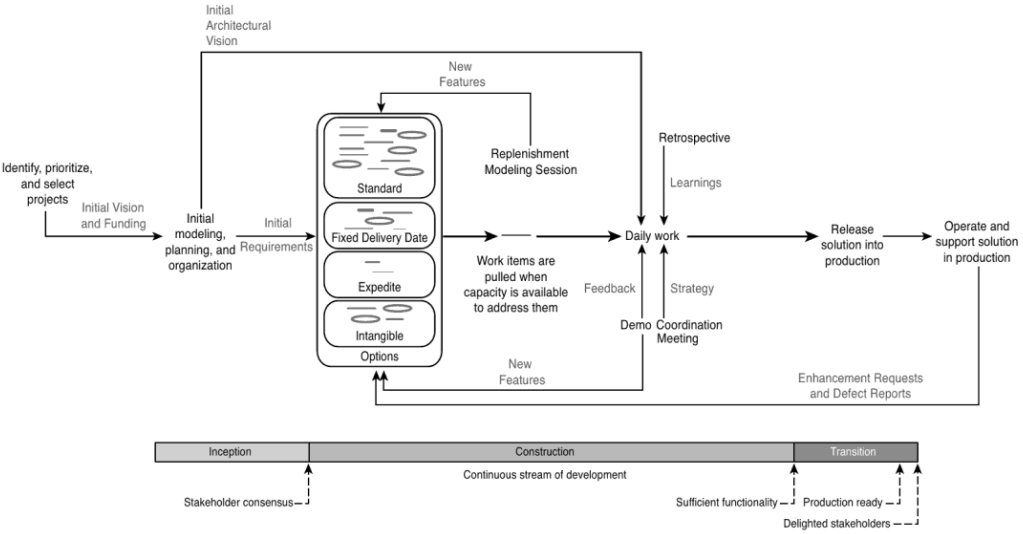

Figure 1.4 A lean version of the DAD lifecycle

One of the challenges with describing a process framework is that you need to provide sufficient guidance to help people understand the framework, but if you provide too much guidance

you become overly prescriptive. As we've helped various organizations improve their software processes over the years, we've come to the belief that the various process protagonists are coming from one extreme or the other. Either there are very detailed processes descriptions (the IBM Rational Unified Process [RUP] is one such example), or there are very lightweight process descriptions, with Scrum being a perfect example. The challenge with RUP is that many teams do not have the skill to tailor it down appropriately, often resulting in extra work being performed. On the other hand many Scrum teams had the opposite problem with not knowing how to tailor it up appropriately, resulting in significant effort reinventing or relearning techniques to address the myriad issues that Scrum doesn't cover (this becomes apparent in Chapter 3). Either way, a lot of waste could have been avoided if only there was an option between these two extremes.

To address this challenge the DAD process framework is goals driven, as summarized in Figure 1.5. There are of course many ways that these goals can be addressed, so simply indicating the goals is of little value. In Chapters 6 through 19 when we describe each of the phases in turn, we suggest strategies for addressing the goals and many times discuss several common strategies for doing so and the trade-offs between them. Our experience is that this goals-driven, suggestive approach provides just enough guidance for solution delivery teams while being sufficiently flexible so that teams can tailor the process to address the context of the situation in which they find themselves. The challenge is that it requires significant discipline by agile teams to consider the issues around each goal and then choose the strategy most appropriate for them. This may not be the snazzy new strategy that everyone is talking about online, and it may require the team to perform some work that they would prefer to avoid given the choice.

Goals for the Inception Phase	Goals for Construction Phase Iterations	Goals for the Transition Phase
- Form initial team - Identify the vision for the project - Bring stakeholders to agreement around the vision - Align with enterprise direction - Identify initial technical strategy, initial requirements, and initial release plan - Set up the work environment - Secure funding - Identify risks	- Produce a potentially consumable solution - Address changing stakeholder needs - Move closer to deployable release - Maintain or improve upon existing levels of quality - Prove architecture early	- Ensure the solution is production ready - Ensure the stakeholders are prepared to receive the solution - Deploy the solution into production
Ongoing Goals		
- Fulfill the project mission - Grow team members' skills - Enhance existing infrastructure	- Improve team process and environment - Leverage existing infrastructure - Address risk	

Figure 1.5 Goals addressed throughout a DAD project

Figure 1.5 doesn't provide a full listing of the goals your team will address. There are several personal goals of individuals, such as specific learning goals and the desire for interesting work, compensation, and public recognition of their work. There are also specific stakeholder goals, which will be unique to your project.

The Agile 3C Rhythm

Over the years we've noticed a distinct rhythm, or cadence, at different levels of the agile process. We call this the agile 3C rhythm, for coordinate, collaborate, and conclude. This is similar conceptually to Deming's Plan, Do, Check, Act (PDCA) cycle where coordinate maps to plan, collaborate maps to do, and conclude maps to check and act. The agile 3C rhythm occurs at three levels in the DAD process framework:

1. **Release.** The three phases of the delivery lifecycle—Inception, Construction, Transition—map directly to coordinate, collaborate, and conclude, respectively.
2. **Iteration.** DAD construction iterations begin with an iteration planning workshop (coordinate), doing the implementation work (collaborate), and then wrapping up the iteration with a demo and retrospective (conclude).
3. **Day.** A typical day begins with a short coordination meeting, is followed by the team collaborating to do their work, and concludes with a working build (hopefully) at the end of the day.

Let's overview the DAD phases to better understand the contents of the DAD process framework.

The Inception Phase

Before jumping into building or buying a solution, it is worthwhile to spend some time identifying the objectives for the project. Traditional methods invest a large amount of effort and time planning their projects up front. Agile approaches suggest that too much detail up front is not worthwhile since little is known about what is truly required as well as achievable within the time and budget constraints. Mainstream agile methods suggest that very little effort be invested in up-front planning. Their mantra can be loosely interpreted as "let's just get started and we will determine where we are going as we go." To be fair, some agile teams have a short planning iteration or do some planning before initiating the project. "Sprint 0" is a common misnomer used by some Scrum teams. Extreme Programming (XP) has the "Planning Game." In fact, a 2009 Ambysoft survey found that teams take on average 3.9 weeks to initiate their projects. In DAD, we recognize the need to point the ship in the right direction before going full-speed ahead—typically between a few days and a few weeks—to initiate the project. Figure 1.6 overviews the potential activities that occur during Inception, described in greater detail in Chapters 6 through 12. This phase ends when the team has developed a vision for the release that the stakeholders agree to and has obtained support for the rest of the project (or at least the next stage of it).

Figure 1.6 Inception phase overview

The Construction Phase

The Construction phase in DAD is the period of time during which the required functionality is built. The timeline is split up into a number of time-boxed iterations. These iterations, the potential activities of which are overviewed in Figure 1.7, should be the same duration for a particular project and typically do not overlap. Durations of an iteration for a certain project typically vary from one week to four weeks, with two and four weeks being the most common options. At the end of each iteration a demonstrable increment of a potentially consumable solution has been produced and regression tested. At this time we consider the strategy of how to move forward in the project. We could consider executing an additional iteration of construction, and whether to deploy the solution to the customer at this time. If we determine that there is sufficient functionality to justify the cost of transition, sometimes referred to as minimally marketable release (MMR), then our Construction phase ends and we move into the Transition phase. The Construction phase is covered in greater detail in Chapters 13 through 17.

The Transition Phase

The Transition phase focuses on delivering the system into production (or into the marketplace in the case of a consumer product). As you can see in Figure 1.8 there is more to transition than merely copying some files onto a server. The time and effort spent transitioning varies from project to project. Shrink-wrapped software entails the manufacturing and distribution of software and documentation. Internal systems are generally simpler to deploy than external systems. High visibility systems may require extensive beta testing by small groups before release to the larger population. The release of a brand new system may entail hardware purchase and setup while updating an existing system may entail data conversions and extensive coordination with the user community. Every project is different. From an agile point of view, the Transition phase ends when the stakeholders are ready and the system is fully deployed, although from a lean point

of view, the phase ends when your stakeholders have worked with the solution in production and are delighted by it. The Transition phase is covered in greater detail in Chapters 18 and 19.

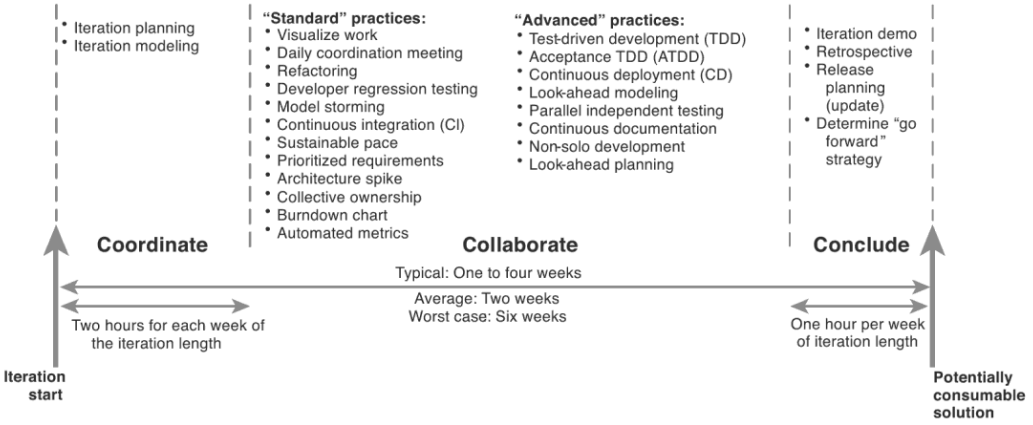

Figure 1.7 Construction iteration overview

Figure 1.8 Transition phase overview

Some agilists will look at the potential activities listed in Figure 1.8 and ask why you couldn't do these activities during construction iterations. The quick answer is yes, you should strive to do as much testing as possible throughout the lifecycle and you should strive to write and maintain required documentation throughout the lifecycle, and so on. You may even do some stakeholder training in later construction iterations and are more likely to do so once your solution has been released into production. The more of these things that you do during the Construction phase, the shorter the Transition phase will be, but the reality is that many organizations

require end-of-lifecycle testing (even if it's only one last run of your regression test suite), and there is often a need to tidy up supporting documentation. The November 2010 Ambysoft Agile State of the Art survey found that the average transition/release phase took 4.6 weeks.

Enterprise Aware

DAD teams work within your organization's enterprise ecosystem, as do other teams, and explicitly try to take advantage of the opportunities presented to them—to coin an environmental cliché "disciplined agilists act locally and think globally." This includes working closely with the following: enterprise technical architects and reuse engineers to leverage and enhance[2] the existing and "to be" technical infrastructure; enterprise business architects and portfolio managers to fit into the overall business ecosystem; senior managers who should be governing the various teams appropriately; operations staff to support your organization's overall development and operations (DevOps) efforts; data administrators to access and improve existing data sources; IT development support people to understand and follow enterprise IT guidance (such as coding, user interface, security, and data conventions to name a few); and business experts who share their market insights, sales forecasts, service forecasts, and other important concerns. In other words, DAD teams should adopt what Mark refers to as a "whole enterprise" mindset.

What Is Appropriate Governance?

Effective governance strategies should enhance that which is being governed. An appropriate approach to governing agile delivery projects, and we suspect other types of efforts, is based on motivating and then enabling people to do what is right for your organization. What is right of course varies, but this typically includes motivating teams to take advantage of, and to evolve, existing corporate assets following common guidelines to increase consistency, and working toward a shared vision for your organization. Appropriate governance is based on trust and collaboration. Appropriate governance strategies should enhance the ability of DAD teams to deliver business value to their stakeholders in a cost effective and timely manner.

Unfortunately many existing IT governance strategies are based on a command-and-control, bureaucratic approach that often proves ineffective in practice. Chapter 20, "Governing Disciplined Agile Teams," explores appropriate governance, the impact of traditional governance strategies, and how to adopt an appropriate governance strategy in greater detail.

With the exception of startup companies, agile delivery teams do not work in a vacuum. Often existing systems are currently in production, and minimally your solution shouldn't impact them. Granted, hopefully your solution will leverage existing functionality and data available in

2. Disciplined agile teams strive to reduce the level of technical debt in your enterprise by adopting the philosophy of mature campers and hikers around the world: Leave it better than how you found it.

production so there will always be at least a minor performance impact without intervention of some kind. You will often have other teams working in parallel to your team, and you may want to take advantage of a portion of what they're doing and vice versa. Your organizations may be working toward a vision to which your team should contribute. A governance strategy might be in place, although it may not be obvious to you, which hopefully enhances what your team is doing.

Enterprise awareness is an important aspect of self-discipline because as a professional you should strive to do what's right for your organization and not just what's interesting for you. Teams developing in isolation may choose to build something from scratch, or use different development tools, or create different data sources, when perfectly good ones that have been successfully installed, tested, configured, and fine-tuned already exist within the organization. We can and should do better by doing the following:

- **Leveraging enterprise assets.** There may be many enterprise assets, or at least there should be, that you can use and evolve. These include common development guidelines, such as coding standards, data conventions, security guidelines, and user interface standards. DAD teams strive to work to a common infrastructure; for example, they use the enterprise-approved technologies and data sources whenever possible, and better yet they work to the "to be" vision for your infrastructure. But enterprise assets are far more than standards. If your organization uses a disciplined architecture-centric approach to building enterprise software, there will be a growing library of service-based components to reuse and improve upon for the benefit of all current and future solutions. To do this DAD teams collaborate with enterprise professionals—including enterprise architects, enterprise business modelers, data administrators, operations staff, and reuse engineers—throughout the lifecycle and particularly during Inception during envisioning efforts. Leveraging enterprise assets increases consistency and thereby ease of maintenance, decreases development costs and time, and decreases operational costs.
- **Enhancing your organizational ecosystem.** The solution being delivered by a DAD team should minimally fit into the existing organizational ecosystem—the business processes and systems supporting them—it should better yet enhance that ecosystem. To do this, the first step is to leverage existing enterprise assets wherever possible as described earlier. DAD teams work with operations and support staff closely throughout the lifecycle, particularly the closer you get to releasing into production, to ensure that they understand the current state and direction of the organizational ecosystem. DAD teams often are supported by an additional independent test team—see Chapter 15, "A Typical Day of Construction"—that performs production integration testing (among other things) to ensure that your solution works within the target production environment it will face at deployment time.

- **Sharing learnings.** DAD teams are learning oriented, and one way to learn is to hear about the experiences of others. The implication is that DAD teams must also be prepared to share their own learnings with other teams. Within IBM we support agile discussion forums, informal presentations, training sessions delivered by senior team members, and internal conferences to name a few strategies.
- **Open and honest monitoring.** Although agile approaches are based on trust, smart governance strategies are based on a "trust but verify and then guide" mindset. An important aspect of appropriate governance is the monitoring of project teams through various means. One strategy is for anyone interested in the current status of a DAD project team to attend their daily coordination meeting and listen in, a strategy promoted by the Scrum community. Although it's a great strategy we highly recommend, it unfortunately doesn't scale very well because the senior managers responsible for governance are often busy people with many efforts to govern, not just your team. In fact Scott found exactly this in the 2010 How Agile Are You? survey. Another approach, one that we've seen to be incredibly effective, is for DAD teams to use instrumented and integrated tooling, such as Rational Team Concert (RTC), which generates metrics in real time that can be displayed on project dashboards. You can see an example of such a dashboard for the Jazz™ team itself at www.jazz.net, a team following an open commercial strategy. Such dashboards are incredibly useful for team members to know what is going on, let alone senior managers. A third strategy is to follow a risk-driven lifecycle, discussed in the next section, with explicit milestones that provide consistent and coherent feedback as to the project status to interested parties.

Risk and Value Driven

The DAD process framework adopts what is called a risk/value lifecycle, effectively a lightweight version of the strategy promoted by the Unified Process (UP). DAD teams strive to address common project risks, such as coming to stakeholder consensus around the vision and proving the architecture early in the lifecycle. DAD also includes explicit checks for continued project viability, whether sufficient functionality has been produced, and whether the solution is production ready. It is also value driven, a strategy that reduces delivery risk, in that DAD teams produce potentially consumable solutions on a regular basis.

It has been said "attack the risks before they attack you." This is a philosophy consistent with the DAD approach. DAD adopts what is called a risk-value driven lifecycle, an extension of the value-driven lifecycle common to methods such as Scrum and XP. With a value-driven lifecycle you produce potentially shippable software every iteration or, more accurately from a DAD perspective, a potentially consumable solution every iteration. The features delivered represent those in the requirements backlog that are of highest value from the perspective of the stakeholders. With a risk-value driven lifecycle you also consider features related to risk as high priority

items, not just high-value features. With this in mind we explicitly address risks common to IT delivery projects as soon as we possibly can. Value-driven lifecycles address three important risks—the risk of not delivering at all, the risk of delivering the wrong functionality, and political risks resulting from lack of visibility into what the team is producing. Addressing these risks is a great start, but it's not the full risk mitigation picture.

First and foremost, DAD includes and extends standard strategies of agile development methods to reduce common IT delivery risks:

- **Potentially consumable solutions.** DAD teams produce potentially consumable solutions every construction iteration, extending Scrum's strategy of potentially shippable software to address usability concerns (the consumability aspect) and the wider issue of producing solutions and not just software. This reduces delivery risk because the stakeholders are given the option to have the solution delivered into production when it makes sense to do so.
- **Iteration demos.** At the end of each construction iteration the team should demo what they have built to their key stakeholders. The primary goal is to obtain feedback from the stakeholders and thereby improve the solution they're producing, decreasing functionality risk. A secondary goal is to indicate the health of the project by showing their completed work, thereby decreasing political risk (assuming the team is working successfully).
- **Active stakeholder participation.** The basic idea is that not only should stakeholders, or their representatives (i.e., product owners), provide information and make decisions in a timely manner, they can also be actively involved in the development effort itself. For example, stakeholders can often be actively involved in modeling when inclusive tools such as paper and whiteboards are used. Active stakeholder involvement through the entire iteration, and not just at demos, helps to reduce both delivery and functionality risk due to the greater opportunities to provide feedback to the team.

DAD extends current agile strategies for addressing risk on IT delivery projects, but also adopts explicit, lightweight milestones to further reduce risk. At each of these milestones an explicit assessment as to the viability of the project is made by key stakeholders and a decision as to whether the project should proceed is made. These milestones, indicated on the DAD lifecycle depicted previously in Figure 1.3, are

- **Stakeholder consensus.** Held at the end of the Inception phase, the goal of this milestone is to ensure that the project stakeholders have come to a reasonable consensus as to the vision of the release. By coming to this agreement we reduce both functionality and delivery risk substantially even though little investment has been made to date in the development of a working solution. Note that the right outcome for the business may in fact be that stakeholder consensus cannot be reached for a given project vision. Our

experience is that you should actually expect to cancel upwards to10% of your projects at this milestone, and potentially 25% of projects that find themselves in scaling situations (and are therefore higher risk).

- **Proven architecture.** In the early Construction phase iterations we are concerned with reducing most of the risk and uncertainty related to the project. Risk can be related to many things, such as requirements uncertainty, team productivity, business risk, and schedule risk. However, at this point in time much of the risk on an IT delivery project is typically related to technology, specifically at the architecture level. Although the high-level architecture models created during the Inception phase are helpful for thinking through the architecture, the only way to be truly sure that the architecture can support the requirements is by proving it with working code. This is a vertical slice through the software and hardware tiers that touches all points of the architecture from end to end. In the UP this is referred to as "architectural coverage" and in XP as a "steel thread" or "tracer bullet." By writing software to prove out the architecture DAD teams greatly reduce a large source of technical risk and uncertainty by discovering and then addressing any deficiencies in their architecture early in the project.
- **Continued viability.** In Scrum the idea is that at the end of each sprint (iteration) your stakeholders consider the viability of your project. In theory this is a great idea, but in practice it rarely seems to happen. The cause of this problem is varied—perhaps the stakeholders being asked to make this decision have too much political stake in the project to back out of it unless things get really bad, and perhaps psychologically people don't notice that a project gets into trouble in the small periods of time typical of agile iterations. The implication is that you need to have purposeful milestone reviews where the viability of the project is explicitly considered. We suggest that for a given release you want to do this at least twice, so for a six month project you would do it every second month, and for longer projects minimally once a quarter.
- **Sufficient functionality.** The Construction phase milestone is reached when enough functionality has been completed to justify the expense of transitioning the solution into production. The solution must meet the acceptance criteria agreed to earlier in the project, or be close enough that it is likely any critical quality issues will be addressed during the Transition phase.
- **Production ready.** At the end of the Transition phase your key stakeholders need to determine whether the solution should be released into production. At this milestone, the business stakeholders are satisfied with and accept the solution and the operations and support staff are satisfied with the relevant procedures and documentation.
- **Delighted stakeholders.** The solution is running in production and stakeholders have indicated they are delighted with it.

Scalable

The DAD process framework provides a scalable foundation for agile IT and is an important part of the IBM agility@scale[3] strategy. This strategy makes it explicit that there is more to scaling than team size and that there are multiple scaling factors a team may need to address. These scaling factors are

- **Geographical distribution.** A team may be located in a single room, on the same floor but in different offices or cubes, in the same building, in the same city, or even in different cities around the globe.
- **Team size.** Agile teams may range from as small as two people to hundreds and potentially thousands of people.
- **Regulatory compliance.** Some agile teams must conform to industry regulations such as the Dodd-Frank act, Sarbanes-Oxley, or Food and Drug Administration (FDA) regulations.
- **Domain complexity.** Some teams apply agile techniques in straightforward situations, such as building an informational Web site, to more complex situations such as building an internal business application, and even in life-critical health-care systems.
- **Technical complexity.** Some agile teams build brand-new, "greenfield systems" from scratch running on a single technology platform with no need to integrate with other systems. At the other end of the spectrum some agile teams are working with multiple technologies, evolving and integrating with legacy systems, and evolving and accessing legacy data sources.
- **Organizational distribution.** Some agile teams are comprised of people who work for the same group in the same company. Other teams have people from different groups of the same company. Some teams are made up of people from similar organizations working together as a consortium. Some team members may be consultants or contractors. Sometimes some of the work is outsourced to one or more external service provider(s).
- **Organizational complexity.** In some organizations people work to the same vision and collaborate effectively. Other organizations suffer from politics. Some organizations have competing visions for how people should work and worse yet have various subgroups following and promoting those visions.
- **Enterprise discipline.** Many organizations want their teams to work toward a common enterprise architecture, take advantage of strategic reuse opportunities, and reflect their overall portfolio strategy.

3. The term "agility@scale" was first coined by Scott in his IBM developerWorks blog by the same name. The full term is now IBM agility@scale™.

Each team will find itself in a unique situation and will need to tailor its strategy accordingly. For example a team of 7 collocated people in a regulatory environment works differently than a team of 40 people spread out across several locations in a non-regulatory environment. Each of the eight scaling factors just presented will potentially motivate tailoring to DAD practices. For example, although all DAD teams do some sort of initial requirements envisioning during the Inception phase, a small team does so differently than a large team, a collocated team uses different tools (such as whiteboards and paper) than a distributed team (who might use IBM Rational Requirements Composer in addition), and a team in a life-critical regulatory environment would invest significantly more effort capturing requirements than a team in a nonregulatory environment. Although it's the same fundamental practice, identifying initial requirements, the way in which you do so will be tailored to reflect the situation you face.

Concluding Thoughts

The good news is that evidence clearly shows that agile methods deliver superior results compared to traditional approaches and that the majority of organizations are either using agile techniques or plan to in the near future. The bad news is that the mainstream agile methods—including Scrum, Extreme Programming (XP), and Agile Modeling (AM)—each provide only a part of the overall picture for IT solution delivery. Disciplined Agile Delivery (DAD) is a hybrid process framework that pulls together common practices and strategies from these methods and supplements these with others, such as Agile Data and Kanban, to address the full delivery lifecycle. DAD puts people first, recognizing that individuals and the way that they work together are the primary determinants of success on IT projects. DAD is enterprise aware, motivating teams to leverage and enhance their existing organizational ecosystem, to follow enterprise development guidelines, and to work with enterprise administration teams. The DAD lifecycle includes explicit milestones to reduce project risk and increase external visibility of key issues to support appropriate governance activities by senior management.

Additional Resources

For more detailed discussions about several of the topics covered in this chapter:

- **The Agile Manifesto.** The four values of the Agile Manifesto are posted at http://www.agilemanifesto.org/ and the twelve principles behind it at http://www.agilemanifesto.org/principles.html. Chapter 2 explores both in greater detail.
- **Agile surveys.** Throughout the chapter we referenced several surveys. The Agile Journal Survey is posted at http://www.agilejournal.com/. The results from the Dr. Dobb's Journal (DDJ) and Ambysoft surveys are posted at http://www.ambysoft.com/surveys/, including the original source data, questions as they were asked, as well as slide decks summarizing Scott Ambler's analysis.

- **People first.** The Alistair Cockburn paper, "Characterizing people as non-linear, first-order components in software development" at http://alistair.cockburn.us/Characterizing+people+as+non-linear%2c+first-order+components+in+software+development argues that people are the primary determinant of success on IT projects. In "Generalizing Specialists: Improving Your IT Skills" at http://www.agilemodeling.com/essays/generalizingSpecialists.htm Scott argues for the need to move away from building teams of overly specialized people.
- **The Agile Scaling Model (ASM).** The ASM is described in detail in the IBM white-paper "The Agile Scaling Model (ASM): Adapting Agile Methods for Complex Environments" at ftp://ftp.software.ibm.com/common/ssi/sa/wh/n/raw14204usen/RAW14204USEN.PDF.
- **Lean.** For more information about lean software development, Mary and Tom Poppendieck's *Implementing Lean Software Development: From Concept to Cash* (Addison Wesley, 2007) is the best place to start.
- **Hybrid processes.** In *SDLC 3.0: Beyond a Tacit Understanding of Agile* (Fourth Medium Press, 2010), Mark Kennaley summarizes the history of the software process movement and argues for the need for hybrid processes that combine the best ideas from the various process movements over the past few decades.

CHAPTER 2

Introduction to Agile and Lean

Important agile philosophy: If something is hard, do it more often so that you get good at it and thereby make it easy.

The first generation of software development methodologies has been described as "waterfall" or "traditional." They are plan-driven, serial processes that assume that software development is comprised of a series of tasks that are easily identified, predictable, and repeatable. History has shown that this is clearly not the case. Unlike other engineering disciplines, software is a creative science that requires some degree of invention and carries a material level of risk and uncertainty on almost every nontrivial project.

The second generation of software development methodologies has been described as "iterative." These methods acknowledge that breaking large projects into a series of time-boxed iterations allows opportunities to demonstrate progress to stakeholders, learn and adapt processes, and get an early insight into quality, among other benefits. Statistics show that iterative methods produce marked improvements in success over traditional approaches. However, project success level on iterative projects is still far from satisfactory.[1] We continue to miss deadlines, exceed budgets, and deliver solutions that do not meet the needs of our business stakeholders. Our productivity levels are poor and our processes and bureaucracies generate a phenomenal amount of waste in terms of unneeded documentation, meetings, signoffs, delayed feedback cycles, and handoffs. Additionally, we continue to have quality problems related to undetected defects and hard to maintain software.

In 2001, 17 thought leaders (self-described "organizational anarchists") got together in Snowbird, Utah, to brainstorm about a better way to work. What emerged was the Manifesto for

1. On a roughly annual basis Scott runs industry surveys that explore success rates by paradigm. The survey results are posted at www.ambysoft.com/surveys/.

Agile Software Development, published at www.agilemanifesto.org, often referred to simply as the Agile Manifesto. The Agile Manifesto defined four values that in turn are supported by twelve principles, and this concise publication has had a significant impact on the way IT professionals think about software development. This collection of ideas formed a rebellion of sorts against document-driven, heavyweight software development processes.

Some unfortunate consequences have resulted from the popularity of the Agile Manifesto. First, for a subset of the agile community the Agile Manifesto has become the equivalent of a religious document, from which one should not deviate. Second, biases captured in the manifesto toward software projects have narrowed the scope of the discussion within the agile community, hampering agile adoption in enterprise situations. (More on this bias later.) Third, although there have been many excellent suggestions over the years for improving the Agile Manifesto, the religious fervor surrounding it makes it all but impossible to change. Yet, as we argue below, changes to the Agile Manifesto are needed if we're to be truly effective at applying agile strategies in enterprise situations.

Figure 2.1 shows a mind map of the structure of this chapter. We describe each of the topics in the map in clockwise order, beginning at the top right.

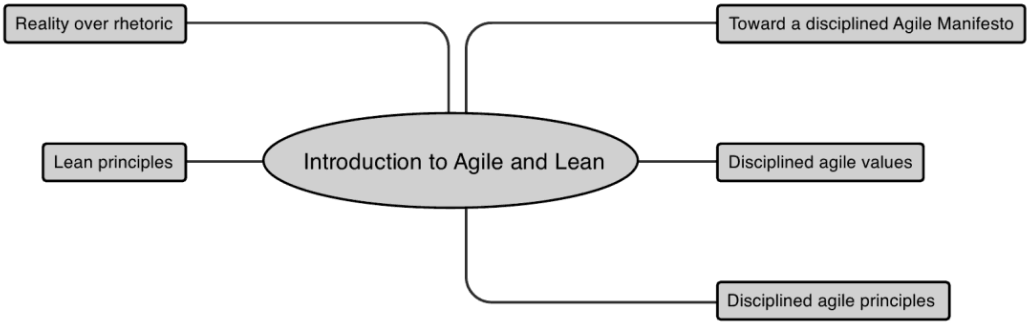

Figure 2.1 Outline of this chapter

The Big Ideas in This Chapter

- The values of, and principles behind, the Agile Manifesto provide an important philosophical foundation for agile.
- We have enhanced the Agile Manifesto to address the experiences of successful teams over the past 10 years taking a disciplined approach to agile delivery.
- The principles of lean software development are also an important part of the philosophical foundation for DAD.
- The rhetoric surrounding agile doesn't always reflect the realities of what people do on actual projects.

Toward a Disciplined Agile Manifesto

First of all, we are big fans of the ideas captured in the Agile Manifesto. Over the past decade we've applied the principles with great benefit for our customers and have learned from our experiences and from the experiences of other successful organizations doing so. However, what we've learned has motivated us to suggest enhancements to the manifesto to reflect the enterprise situations in which we have applied agile and lean strategies. These enhancements reflect the realities faced by most Disciplined Agile Delivery (DAD) teams.

We believe that the changes we're suggesting to the Agile Manifesto are straightforward:

1. Where the original manifesto focused on software development, a term that too many people have understood to mean *only* software development, we suggest that it should instead focus on solution delivery.
2. Where the original manifesto focused on customers, a word that for too many people appears to imply only the business stakeholders, we suggest that it focus on the full range of stakeholders instead.
3. Where the original manifesto focused on development teams, we suggest that the overall organizational ecosystem and its improvement be taken into consideration.

Furthermore, some interesting work has been done within the lean community since the Agile Manifesto was written, and we believe that it can benefit from these ideas. So let's explore what an updated Agile Manifesto might look like.

Disciplined Agile Values

Each of the four value statements of the Agile Manifesto is presented in the format X over Y. The important thing to understand about the statements is that while you should value the concepts on the right-hand side you should value the things on the left-hand side even more. A good way to think about the manifesto is that it defines preferences, not alternatives, encouraging a focus on certain areas but not eliminating others.

The four updated values—changes are indicated in italics—of the Agile Manifesto are as follows:

1. **Individuals and interactions over processes and tools.** Teams of people build software systems, and to do that they need to work together effectively. Who do you think would develop a better system: five skilled software developers with their own tools working together in a single room or ten low-skilled programmers with a well-defined process, the most sophisticated tools available, and each with their own best office money could buy? Our money would be on the smaller team of collocated software developers. Tools and processes are important; they're just not as important as working together effectively.

DAD differences: None. DAD whole-heartedly embraces individuals and interactions over processes and tools.

2. **Working *solutions* over comprehensive documentation.** When you ask someone whether they would want a 50-page document describing what you intend to build *or* the actual solution itself, what do you think they'll pick? Our guess is that 99 times out of 100 they'll choose the working solution. Doesn't working in such a manner that you produce a potentially consumable solution quickly and often make more sense? Furthermore, stakeholders will have a significantly easier time understanding any working solution that you produce rather than complex technical diagrams describing its internal workings or describing an abstraction of its usage, don't you think? Documentation has its place; deliverable documentation such as user manuals and operations manuals are in fact part of the overall solution, but it is only a small part. Never forget that the primary goal of IT delivery teams is to create solutions, not documents; otherwise it would be called documentation development wouldn't it? Note that the original Agile Manifesto used the term "software," not "solutions," for this value statement.

   DAD differences: It's not enough to just have working software, but instead a consumable (usable) solution that includes software as well as potential changes to the hardware it runs on, the business process it supports, the documentation that should be produced with it, and even the organization of the people working with it.

3. ***Stakeholder* collaboration over contract negotiation.** Only your stakeholders can tell you what they want. Yes, they likely do not have the skills to exactly specify the solution. Yes, they likely won't get it right the first time. Yes, they'll likely change their minds once they see what your team produces. Yes, there is a wide range of stakeholders, including end users, their managers, senior IT managers, enterprise architects, operations staff, support staff, regulatory auditors, and many more (Chapter 4, "Roles, Rights, and Responsibilities," goes into greater detail). Working together with your stakeholders is hard, but that's the reality of the job. Having a contract with your stakeholders is important, but a contract isn't a substitute for effective communication. Successful teams work closely with their stakeholders, they invest the effort to discover what their stakeholders need, and they educate their stakeholders as to the implications of their decisions along the way. Mainstream methods suggest that the team should be able to create the solution in isolation, needing only to collaborate with each other and the "one voice of the customer" known as the product owner. This is seldom the case on nontrivial projects. The team will need to collaborate with many types of stakeholders such as those listed previously, not just the customer (representative). Note that the original Agile Manifesto used the term "customer" instead of "stakeholder" for this value statement.

DAD differences: Explicit recognition that there is a wide range of potential stakeholders for a solution, not just the business customer, and that you will need to interact with more than just a stakeholder representative to understand their true needs.

4. **Responding to change over following a plan.** People change their priorities for a variety of reasons. As work progresses, stakeholders' understanding of the problem domain and of your solution changes, as does the business environment and even the underlying technology. Change is a reality of software development, a reality that your delivery process must reflect. There is nothing wrong with having a project plan; in fact, we would be worried about any project that didn't have one. However, a project plan must be malleable and should only be detailed for the near term (a few weeks or less).

   DAD differences: None.

The interesting thing about these value statements is they are something that almost everyone instantly agrees to, yet rarely adheres to in practice. Senior management always claims that its employees are the most important aspect of an organization, yet we still see instances in industry where they treat their staff as replaceable assets. An even more damaging situation arises when management refuses to provide sufficient resources to comply with the processes that they insist project teams follow. Everyone will readily agree that the creation of a consumable solution is the fundamental goal of delivery, yet insist on spending months producing documentation describing what the solution is and how it is going to be built instead of simply rolling up their sleeves and building it. You get the idea—people say one thing and do another. This has to stop now. Disciplined agile developers do what they say and say what they do.

Disciplined Agile Principles

To help people to gain a better understanding of what agile software development is all about, the members of the Agile Alliance refined the philosophies captured in their manifesto into a collection of twelve principles. We have modified several of these principles—changes to the wording are shown in italics—and have added three new ones. The fifteen disciplined agile principles are the following:

1. **Our highest priority is to satisfy the stakeholder through early and continuous delivery of valuable *solutions*.** We must remember that the goal of solution delivery should be the delivery of an actual consumable solution—not only are we developing software, but we're often improving the hardware it runs on, evolving the business processes around the usage of the software, evolving the operations and support processes required to run the solution (an aspect of a "DevOps" approach to delivery), and even changing the organization structure of the people working with the solution. We need to move away from a strategy where we try to think through all the details up

front, thereby increasing both project risk and cost, and instead invest a bit of time thinking through the critical issues but allowing the details to evolve over time as we learn through incremental creation of the solution.

2. **Welcome changing requirements, even late in the *solution delivery lifecycle*.** Agile processes harness change for the *stakeholders'* competitive advantage. Like it or not, requirements will change throughout a project. Traditional software teams often adopt change management processes designed to prevent/reduce scope creep, but when you think about it these are really change prevention processes, not change management processes. Disciplined agilists follow an agile change management approach where functionality is worked on in priority order and requirements evolve to reflect stakeholders' improved understanding of what they actually need. Taking into consideration all of the aspects of delivering a solution, we recognize that we have to engage representatives from all stakeholder groups who are impacted—end users, end user managers, senior managers, operations, support, enterprise architects, IT governance personnel, finance, marketing, and more.
3. **Deliver working *solutions* frequently, from a couple of weeks to a couple of months, with a preference to the shorter time scale.** Frequent delivery of a consumable solution provides stakeholders with the opportunity to provide timely feedback, making the current status of your project transparent while at the same time providing an opportunity for stakeholders to provide improved direction for the development team.
4. ***Stakeholders* and developers must work together daily throughout the project.** Your project is in serious trouble if you don't have regular access to your project stakeholders, or at least their representatives. Disciplined agile teams adopt practices such as on-site customer and active stakeholder participation, and adopt inclusive tools and techniques that enable stakeholders to be actively involved with solution delivery.
5. **Build projects around motivated individuals.** Give them the environment and support they need, and trust them to get the job done. Too many organizations have a vision that they can hire hordes of relatively unskilled people, provide them with a CMMI/ISO/…-compliant process description, and they will successfully develop solutions. This doesn't seem to work all that well in practice. Disciplined agile teams, on the other hand, realize that you need to build teams from people who are willing to work together collaboratively and learn from each other. They have the humility to respect one another and realize that people are a primary success factor in solution delivery. We should allow them to create an environment in which they will thrive as a team. This includes allowing them to set up a work environment that fosters collaboration, use of tooling that they find most effective, and the freedom to customize and optimize their team's development process.

6. **The most efficient and effective method of conveying information to and within a *delivery* team is face-to-face conversation.** For a delivery team to succeed its members must communicate and collaborate effectively. There are many ways that people can communicate together, and face-to-face communication at a shared drawing environment (such as paper or a whiteboard) is often the most effective way to do so. Sending endless emails and creating exhaustive documents are far less effective than the immediate feedback of conversation. Distributed teams are not an excuse for reverting back to extensive documentation practices since video chat can be used for face-to-face conversations.
7. ***Quantified business value* is the primary measure of progress.** The primary measure of a solution delivery project should be the delivery of a consumable solution that provides actual value to your stakeholders. This solution should meet the changing needs of its stakeholders, not some form of "earned value" measure based on the delivery of documentation or the holding of meetings. Note we have replaced the phrase "Working software" with "Quantified Business Value." Demonstrable working solutions are indeed a key measure of progress but give a false measure of success if they do not provide the expected business value.
8. **Agile processes promote sustainable *delivery*.** The sponsors, developers, and users should be able to maintain a constant pace indefinitely. Just like you can't sprint for an entire marathon, you can't successfully produce a consumable solution by forcing people to work overtime for months at a time. Our experience is that you can only do high-quality, intellectual work for 5 to 6 hours a day before burning yourself out. The rest of the day can be filled up with email, meetings, water cooler discussions, and so on, but people's ability to do "real work" is limited. Yes, you might be able to do high-quality work for 12 hours a day, and do so for a few days straight, but after a while you become exhausted, and all you accomplish is 12 hours of mediocre work a day.
9. **Continuous attention to technical excellence and good design enhances agility.** It's much easier to understand, maintain, and evolve high-quality source code and data sources than it is to work with low-quality ones. Therefore, agilists know that they need to start with high-quality work products, to keep the quality high via refactoring, and to have a full regression test suite so that they know at all times that their solutions work. Disciplined agilists also adopt and follow organizational guidelines, such as programming standards, data guidelines, security guidelines, and user interface conventions (to name a few).
10. **Simplicity—the art of maximizing the amount of work not done—is essential.** Agile developers focus on high value activities, strive to maximize our stakeholders' return on investment and either cut out or automate the drudge work. From a lean point of view, simplicity is essential, so only the most important things are worked on and delays to them (e.g., by starting on less important work in parallel) are minimized.

11. **The best architectures, requirements, and designs emerge from self-organizing teams.** This is one of the most radical principles of the agile movement, one that we would love to see researched thoroughly by the academic community. The agile model driven development (AMDD) and test-driven design (TDD) methods are the primary approaches within the agile community to ensure the emergence of effective architectures, requirements, and designs.
12. **At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.** Software process improvement (SPI) is a continual effort, and techniques such as retrospectives should be adopted to enable you to improve your approach to software development.
13. **NEW: Leverage and evolve the assets within your organizational ecosystem, and collaborate with the people responsible for those assets to do so.** Disciplined agile teams recognize that they are not working in a vacuum. Existing systems, data sources, frameworks, services, and other assets can and should be used and better yet improved as part of the delivery process of their new solution. These existing assets will constrain your solution architecture; much of the technical plumbing for your solution may have already been chosen for you. However, your productivity will potentially improve if you can focus on delivering new functionality and not on reinventing existing infrastructure.
14. **NEW: Visualize workflow to help achieve a smooth flow of delivery while keeping work in progress to a minimum.** DAD teams use status or dashboards to monitor their work in progress and to coordinate their activities, adjusting their approach to address bottlenecks or underutilization of people. This helps the team to maintain a consistent and sustainable delivery flow.
15. **NEW: The organizational ecosystem must evolve to reflect and enhance the efforts of agile teams, yet be sufficiently flexible to still support non-agile or hybrid teams.** Like it or not, organizations have a range of approaches when it comes to IT delivery. The overall IT strategy, including your governance strategy, must reflect this fact for the individual teams to work together effectively. Too many organizations run into trouble when they try to apply a repeatable, one-size-fits-all strategy to IT projects.

Stop for a moment and think about these principles. Is this the way that your IT delivery projects actually work? If not, is this the way that you think they should work? Read the principles once again. Are they radical and impossible goals as some people would claim, are they meaningless abstract principles, or are they simply common sense? Our belief is that these principles form a foundation of common sense upon which you can base successful DAD projects.

Lean Principles

In *Implementing Lean Software Development*, Mary and Tom Poppendieck show how the seven principles of lean manufacturing can be applied to optimize the whole IT value stream. We believe that these principles help to provide a foundation for effective agile development and also provide an explanation for why many of the agile techniques work in practice. The lean software development principles are the following:

- **Eliminate waste.** Lean thinking advocates regard any activity that does not directly add value to the finished product as waste. The three biggest sources of waste in software development are the addition of features that are not required, project churn, and crossing organizational boundaries (particularly between stakeholders and development teams). It is interesting to note that Walker Royce, chief software economist at IBM Rational, argues that the primary benefit of modern iterative/agile techniques is the reduction of scrap and rework late in the lifecycle. To reduce waste it is critical that DAD teams be allowed to self-organize and operate in a manner that reflects the work they're trying to accomplish.
- **Build in quality.** Your process should incorporate practices that minimize the number of defects that occur in the first place, but when this isn't possible you should work in such a way that you do a bit of work, validate it, fix any issues that you find, and then iterate. Inspecting after the fact, and queuing up defects to be fixed at some time in the future, isn't as effective. Agile practices that build quality into your process include refactoring, test-driven development (TDD), and non-solo development practices such as pair programming and modeling with others.
- **Create knowledge.** Planning is useful, but learning is essential. You want to promote strategies, such as iterative development, that help teams discover what stakeholders really want and act on that knowledge. It's also important for a team to regularly reflect on what they're doing and then act to improve their approach.
- **Defer commitment.** It's not necessary to start software development by defining a complete specification, and in fact we know it to be a questionable strategy at best. You can support the business effectively through flexible architectures that are change tolerant and by scheduling irreversible decisions to the last possible moment. Frequently, deferring commitment requires the ability to closely couple end-to-end business scenarios to capabilities developed in multiple applications by multiple projects. The DAD process framework adopts strategies from Agile Modeling that enable teams to think through issues and make commitments at appropriate points in time throughout the delivery lifecycle.

- **Deliver quickly.** Delivering high-quality solutions quickly is an achievable goal. By limiting the work of a team to its capacity, which is reflected by the team's velocity (this is the number of "points" of functionality a team delivers each iteration), you can establish a reliable and repeatable flow of work. An effective organization doesn't demand teams do more than they are capable of, but instead asks them to self-organize and determine what they can accomplish. Constraining these teams to delivering potentially consumable solutions on a regular basis motivates them to stay focused on continuously adding value. This strategy is reflected in the Disciplined Agile principle #14 described earlier.
- **Respect people.** The Poppendiecks also observe that sustainable advantage is gained from engaged, thinking people. The DAD "people first" tenet reflects this principle, the implication being that how you form and then support your delivery teams is critical to your success. Another implication is that you need a governance strategy that focuses on motivating and enabling IT teams, not on controlling them.
- **Optimize the whole.** If you want to be effective at a solution you must look at the bigger picture. You need to understand the high-level business processes that individual projects support—processes that often cross multiple systems. You need to manage programs of interrelated systems so you can deliver a complete solution to your stakeholders. This is a major difference from mainstream agile approaches, which tend to have a project focus and as a result will suboptimize around it. DAD avoids this problem by being enterprise aware and suggesting lean techniques such as value stream mapping, a stylized form of process modeling, to identify potential bottlenecks in the overall process so that they may be addressed. Measurements should address how well you're delivering business value, a motivation for the rewording of agile principle #7, because that is the sole reason for your IT department. A lean measurement system is one aspect of appropriate governance explicitly built into the DAD process framework.

The Kanban method, a lean methodology, describes several principles and techniques for improving your approach to software development. We want to share two Kanban principles that we believe are critical to your success. These principles are the following:

- **Visualize workflow.** Teams use a Kanban board, often a physical whiteboard or corkboard although electronic boards can also be used, that displays indications, called kanbans, of where in the process a piece of work is. The board is typically organized into columns, each of which represents a stage in the process or a work buffer or queue, and optionally rows indicating the allocation of capacity to classes of service. Each Kanban should have sufficient information, such as the name and ID of the work item and due date (if any), to enable decisions by the team without the direction of a manager. The goal is to visually communicate enough information to make the process self-organizing

and self-expediting at the team level. The board is directly updated by team members throughout the day as the work proceeds, and blocking issues are identified during daily coordination meetings.

- **Limit work in progress (WIP).** Limiting work in progress reduces average lead time, which improves the quality of the work produced and thereby increases the overall productivity of your team. Reducing lead time also increases your ability to deliver valuable functionality frequently, which helps to build trust with your stakeholders. To limit work in progress you need to understand where your blocking issues are, address them quickly, and reduce queue and buffer sizes wherever you can. There are some interesting trade-offs: Although buffers and queues add WIP, and therefore increase lead time, they also smooth over the workflow and increase the predictability of lead time. The implication is that because every team is different, you will have different WIP limits that you'll need to set and then evolve yourself based on empirical results from experimentation.

Lean thinking is important for DAD teams, and particularly when you find yourself in a scaling situation, in several ways:

- **Lean provides an explanation for why many of the agile practices work.** For example, Agile Modeling's practices of lightweight, initial requirements envisioning followed by iteration modeling and just-in-time (JIT) model storming work because they reflect deferment of commitment regarding what needs to be built until it's actually needed, and the practices help eliminate waste because you're only modeling what needs to be built.
- **Lean offers insight into strategies for improving your software process.** For example, by understanding the source of waste in IT you can begin to identify it and then eliminate it.
- **Lean principles provide a philosophical foundation for scaling agile approaches.** Instead of optimizing software development, lean promotes optimizing the whole. That means that it's important to look at delivery of the overall solution to the customer, rather than suboptimizing by looking only at delivering software.
- **Lean provides techniques for identifying waste.** Value stream mapping, a technique common within the lean community, whereby you model a process and then identify how much time is spent on value-added work versus wait time, helps calculate overall time efficiency of what you're doing. Value stream maps are a straightforward way to illuminate your IT processes, providing insight into where significant problems exist. Scott has created value stream maps with several customers around the world where they analyzed their existing processes that some of their more traditional staff believed worked well only to discover they had efficiency ratings of 20% to 30%. You can't fix problems to which you are blind.

The implications of lean thinking are profound, and many organizations take some time to fully appreciate them. Think of everything that you do prior to delivering functionality to your customers as work in progress (WIP). This includes work products such as project plans, requirements, tests, defects, and designs. Lean properly adopted means that we vigorously reduce queues of any sort as they are sources of waste. Rather than *push* requirements to our developers, we *pull* requirements from a small queue of highest priorities and then deliver them quickly to the customer. Batching up a large inventory of requirements for delivery is not consistent with this lean approach. Most organizations currently consider it to be a normal practice to take months to gather a batch of requirements for the next release and then proceed with an elaborate plan to deliver these fixed requirements. Organizations that continue to use this approach risk becoming outdated by companies that release multiple versions of their solution in the time that they themselves release their first version.

While both agile and lean approaches consider writing detailed requirements up front to be unacceptable, lean is even more stringent than agile. Agile replaces detailed requirements with a requirement placeholder such as a user story in the work item list. The work item list represents a complete list of the backlog of all high level requirements currently known by the product owner (see Chapter 4 for a detailed discussion of this role). Lean advocates pruning this backlog to a few items only, and only adding new requirements to the list as items are delivered to the customer. An analogy might be a queue of pies on display at a bakery. The baker won't bake a week's worth of pies. Rather, as pies are removed from the displayed queue, he bakes more and replenishes the queue as needed. This is like pulling features from a queue and only then considering what might be a good replacement for the delivered feature.

You might consider the idea to have only a dozen or so features known at a time in advance (with no supporting details) a radical approach, and you would be right. Most organizations do not yet seem ready to adopt this just-in-time approach to eliciting and delivering functionality, which is adapted from just-in-time supply chains in manufacturing and retailing environments.

Perhaps in a few years, progressive organizations will feel more comfortable with supplementing their agile practices with ideas like these from lean thinking.

Reality over Rhetoric

There is a fair bit of rhetoric surrounding agile methods, some of which we subscribe to and some of which we don't. This section briefly examines the rhetoric we've found to be the most misleading for people trying to be effective at adopting agile techniques. The following list is in the format *X although Y*, where *X* is the rhetoric and *Y* is the strategy promoted by the DAD process framework. This includes

- **Requirements evolve throughout the lifecycle *although* the initial scope should still be agreed to at the beginning of the project.** You achieve this by formulating an initial vision for your project, a vision that your stakeholders should help define and then agree

to. To come to that vision you need to perform some initial requirements envisioning—a list of high level features is part of this initial vision. Yes, the details are very likely to evolve over time, but the fundamental goals of your project and scope of your effort need to be defined early in your project. In a small minority of situations you may not be able to get the right people together, either physically or virtually, to define the initial vision—this should be seen as a significant project risk.

- **Simple designs are best *although* the architecture should be thought out early in the lifecycle.** Too many developers interpret the advice to focus on simple designs to mean that they should build everything from scratch. Yet more often than not the simplest design is to take advantage of what is already there, and the best way to do that is to work closely with people who understand your existing technical infrastructure. Investing in a little bit of architectural envisioning early in the lifecycle enables your team to identify existing enterprise assets that you can leverage, to identify your architectural options, and to select what appears to be the best option available to you. The details will still emerge over time, and some decisions will be deferred until a later date when it's more appropriate to make them, but the bottom line is that disciplined agilists think before they act.
- **Teams should be self-organizing *although* they are still constrained (and enhanced) by your organizational ecosystem.** Intellectual workers, including development professionals, are most effective when they have a say in what work they do and how they do it. Development professionals can improve their productivity by following common conventions, leveraging and building out a common "DevOps" infrastructure, and by working to common business and technical visions.
- **Delivery teams don't need prescriptive process definitions *although* they do need some high-level guidance to help organize their work.** Individual development professionals are typically highly skilled and highly educated people often with years of experience, and teams of such people clearly have a wide range of knowledge. As a result of this knowledge it is incredibly rare for such people to read detailed procedures for how to do their work. However, they often still require some high-level advice to help them organize their work effectively. Teams can often benefit from techniques and patterns used by other teams, and this knowledge sharing should be encouraged.
- **Development professionals know what to do *although* they're still not process experts.** A decade ago the strategy was to provide detailed process advice to teams, but recently the pendulum has swung the other way to provide little or no defined process at all. Over the last few years there's been a trend within the agile community to advise teams to define their own process so that it's tailored to their own unique situation. While this clearly strokes people's egos, it's relatively poor advice for several reasons. First, although every team is in a unique situation there is significant commonality, so

having at least a high-level process framework from which to start makes sense. Having a baseline means that teams have a starting point that has delivered successful results for many other agile teams and can evolve beyond that baseline to optimize the value they are able to deliver. Second, although these teams have a wide range of knowledge it might not be complete, nor consistent, nor is it clear what the trade-offs are of combining all the really good techniques that people know about. There is significant benefit in having a flexible process framework such as DAD that shows how everything fits together.

- **Development professionals should validate their own work to the best of their ability *although* they likely aren't testing experts so therefore need help picking up the appropriate skills.** The mantra in the agile community is to test often and test early, and better yet to test first. As a result agile teams have adopted a "whole team" approach where the development team does its own testing. This works when there are people on the team with sufficient testing skills and more importantly can transfer those skills to others. Minimally you need to embed testers into your delivery teams, but you should also consider explicit training and mentoring of everyone on the team in testing and quality skills.
- **Disciplined agile teams work in an iterative manner *although* still follow a lifecycle that is serial[2] over time.** On any given day people on a DAD project team may be performing analysis, testing, design, programming, deployment, or a myriad of other activities and iterating back and forth between them. But, as you saw in Chapter 1, "Disciplined Agile Delivery in a Nutshell," the DAD lifecycle includes three distinct phases, which are performed in order. So, DAD is both iterative in the small but serial in the large.

Concluding Thoughts

Properly executed agile methods definitely deliver business value quicker, more frequently, and with higher quality and provide greater stakeholder satisfaction than traditional methods. Disciplined agile delivery teams produce solutions with features that are needed (not just wanted) with the highest return on investment in a timely just-in-time fashion. The solution is easy to maintain due to the greater focus on quality and has fewer defects than solutions produced with traditional methods. Most importantly, the solution delivered meets the expectations of the stakeholders since they participated actively in the development process, and what is delivered meets their conditions of satisfaction defined merely weeks ago.

As we show in subsequent chapters it is *unusual* to find defects for a feature in iterations subsequent to the feature being implemented when proper regression testing and better yet

2. We appreciate that "serial" is one of those agile swear words that cause some people a bit of heartburn.

Table 3.1 Sources of Agile Practices Adopted by Disciplined Agile Delivery (continued)

Agile Source	Strengths
Extreme Programming (XP)	Technical aspects of software development with specific practices defined for fine-scale feedback, continuous integration, shared understanding, and programmer welfare
Agile Modeling	Lightweight requirements, architecture, and design modeling and documentation
Agile Data	Database architecture, design, and development
Lean software development	A collection of principles and strategies that help streamline software development and provide advice for scaling agile approaches
IBM Practices Library	A collection of practices, ranging from very traditional to very agile, documented by IBM
OpenUP	Full delivery lifecycle planning, modeling, development, testing, deployment, and governance

Fortunately, many aspects of mainstream agile are consistent across all methods. However, in many cases they have different terms for these common practices. In many cases DAD uses an existing term if it makes sense, and in some cases DAD uses a more generic term. The chapter begins by addressing the terminology issue and then overviews each method and its key practices in greater detail. The goal is to provide you with an overview of the wealth of agile practices available to you. Your team may not adopt all of these practices; some are mutually exclusive, and some would be adopted only in specific situations, but you will want to adopt many of them. Future chapters refer to the practices described in this chapter and provide advice for when you may want to apply them.

Figure 3.1 shows a mind map of the structure of this chapter. We describe each of the topics in the map in clockwise order, beginning at the top right.

The Big Ideas in This Chapter

- Mainstream agile methods focus on different portions of the agile delivery lifecycle. Most have some overlap and some gaps.
- The practices of each agile methodology are complementary and easily combined, although sometimes the terminology varies.
- Your project team will want to adopt and tailor a subset of the practices described in this chapter depending on the context you find yourself in.
- DAD combines common agile practices from various sources into a single, end-to-end delivery process.