



Rally Software Development Corporation

Whitepaper

# Mastering the Iteration: An Agile White Paper

Dean Leffingwell

**Abstract:**

The heartbeat of Agile development is the iteration – the ability of the team to create working, tested, value-delivered code in a short time box – with the goal of producing an increment of potentially shippable code at the end of each iteration. This is a significant challenge for the team, and mastering this skill takes guidance and practice. In this white paper, Dean Leffingwell describes the basic iteration pattern and the activities that a team engages in to meet this key challenge. This paper is an excerpt from Dean's latest book, *Scaling Software Agility: Best Practices for Large Enterprises*, Addison-Wesley, 2007.

---

## Table of Contents

Table of Contents .....	2
Iteration: The Heartbeat of Agility .....	3
The Standard, Two-Week Iteration? .....	3
Planning and Executing the Iteration .....	4
Iteration Planning .....	4
Iteration Execution .....	7
Iteration Tracking and Adjusting .....	10
Iteration Cadence Calendar .....	12
About the Book .....	14

---

## Iteration: The Heartbeat of Agility

The base construct of agile and iterative development is the iteration—the ability of the team to create working, tested, value-delivered code in a short time box—with the goal producing an increment of potentially shippable code at the end of each iteration. This is a significant challenge for the team, and mastering the process takes some time to accomplish. In this chapter, we describe the basic iteration pattern and the activities that a team engages in to meet this key challenge.

---

### The Standard, Two-Week Iteration?

Before we begin, however, we must first entertain another small debate: What is the optimal length of an iteration? Most people agree that iterations are a fixed, constant length and typically should not vary in length over the course of a release or a project. But the length of the iteration is an agile variable. From the literature, XP recommends a length of 1 to 4 weeks, Scrum recommends 30 day sprints, and RUP recommends flexibility of from 2 to 6 weeks.

In practice, however, all but a very few teams we have worked with have come to the same conclusion over time: *a week may be too short and 30 days is too long*. The conclusion they typically arrive at is to standardize on iterations of 2 weeks in length, and this is our general recommendation.<sup>1</sup>

There are many advantages to this approach:

- There is some overhead in planning and closing an iteration; in a 2-week iteration, the overhead is well proportioned to the amount of work that can be accomplished in the period.
- It forces work into byte-sized chunks where the define/build/test cycle has to be concurrent. With longer iterations, there is a tendency for teams to build a more waterfall-like process.
- Two weeks is sufficient time to get some amount of meaningful development done.
- This cycle provides more opportunities to succeed or fail early. For example, with releases of approximately 90 days, 2-week iterations give teams time for five “construction” iterations and one “hardening” iteration at the end (see Chapter 13), so there are a number of intermediate checkpoints on the way to the release.
- The 2-week rhythm is a natural calendar cycle that is easy for all participants to remember. Scheduling is trivial: “If it’s the second Wednesday, I need to be preparing my demo for Friday.” “Demo day” occurs every other week, at exactly the same time and place, allowing key stakeholders to attend.
- The cycle lines up well with typical 1- or 2-week vacation cycles, simplifying capacity estimates for the team.
- Velocity can be measured and scope can be adjusted more quickly.

For these and many other reasons, we recommend that iterations be standardized to 2 weeks. Also, in order to facilitate coordination with other teams and larger releases, we recommend that all teams on a project apply this same iteration time box where possible, although start and stop days may be different to support those managers and team leaders engaged with more than one team.

---

<sup>1</sup> In practice in my personal experience set, only one team operates in 1-week iterations; all the others use a 2-week cycle.

## Planning and Executing the Iteration

No matter the length, all iterations have the same pattern, and that is part of the discipline and manufacturing-like routine of agile development. An iteration consists of three phases, as Figure 11–1 illustrates. The first phase is a short planning session (less than a day) during which the iteration backlog is reviewed and prioritized, estimates are established, and the team commits to the work in the iteration. The second is the development phase, when the backlog items are implemented in code and tests. The final phase involves delivery of the new system increment built during the iteration and assessment of the iteration.

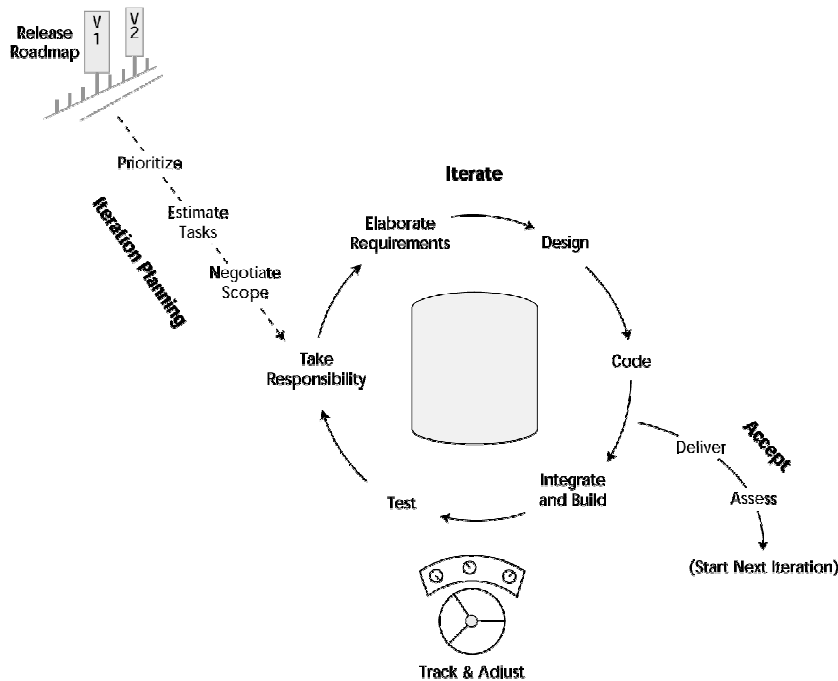


Figure 11–1 Iteration process model

## Iteration Planning

At the beginning of each iteration, the team holds an iteration planning session during which it reviews the prioritized items in the backlog, selects and reviews the stories for the current iteration, and defines and estimates the tasks necessary to deliver the increment of work. There is a high degree of granularity involved because tasks are typically estimated in “ideal developer days” or even hours. In keeping with the agile practice of just-in-time design, it is during this session that the details of the requirements are discussed and negotiated.

### Preparation for the Iteration Planning Meeting

Because the iteration planning meeting is short and time-boxed, the team has to enter the meeting in a prepared state, and all team members have some responsibility to prepare for iteration planning. For example, the development team and the product owners have specific areas of responsibility, which are outlined in Table 11–1.

Table 11–1 Iteration Planning Meeting Responsibilities

**Product Owner Responsibilities**

Review the release plan to make sure the vision and goals are still appropriate.

**Development Team Responsibilities**

Review the top priority items in the backlog and prepare any questions.

Review the items in the backlog and reprioritize if necessary. This includes stories that (a) were already there (originally defined in previous release planning sessions); (b) have been added since the last release planning session; (c) failed acceptance in a prior iteration; (d) are generated from defects or bugs.

Consider technical issues, constraints, and dependencies, and be prepared to share these concerns.

Understand how the reprioritization may affect other teams who are dependent on a deliverable committed to during release planning. Coordinate with other product managers as needed to resolve dependency issues.

Think about the work involved in delivering the functionality in the stories, in order to be prepared to make estimates in the meeting.

Understand the customer needs and the business value that each story is to deliver.

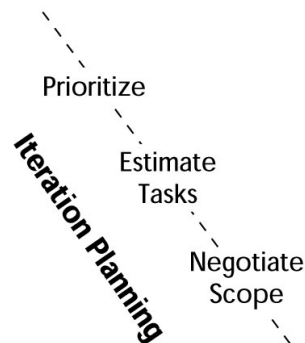
Understand what your iteration velocity should be for the upcoming iteration, based on team discussions at the last review.

Be prepared to further elaborate on the details of each story.

**Participants**

Everyone who will directly affect the product outcome should attend the iteration planning meeting. Those who may indirectly affect the product outcome are strongly encouraged to attend as well. Outside stakeholders are also welcome to attend, but once planning begins, they must take on the role of chickens<sup>2</sup> and refrain from speaking. Required attendees typically include the Agile team leader/Scrum Master; product owner/business analyst; developer; test, QA, and documentation personnel; and architect.

**Iteration Planning Meeting**



The primary concern of iteration planning is to define and accept a reasonable scope for the iteration. Iteration planning begins with revising and refining the list of prioritized work in the backlog. Product owners and the development team may add or reduce features, defects, and other infrastructure work on the basis of the

<sup>2</sup> This metaphor is based on the “chicken and pig” joke discussed in Scrum. A chicken and pig are discussing opening a restaurant to serve ham and eggs. The pig thinks for a second and says, “Wait a minute, if we do this, you are only involved, but I am committed!”

current business situation. Business priority, risk, and rough estimates are assigned to new items or may be revised for existing items. The product owners then rerank the work items and select a scope of work to propose for the iteration.

The development team is given an opportunity to discuss the proposed work with product owners until each item is well enough understood for the development team to prepare a list of engineering tasks and provide detailed estimates. The development team then estimates the engineering tasks for each proposed backlog item. The development team then presents its estimates to the product owners.

By adding up the development team's estimates, the team can calculate the apparent scope of the iteration and have an indication of whether the scope can be achieved. However, the final scope of the iteration is the result of a negotiation between product owners and the development team. During this negotiation, product owners may adjust certain backlog items in ways that make them less costly to develop, trade out entire backlog items for others, or ask for adjustments to certain estimates provided by development.

Mary and Tom Poppendieck [2003] point out the benefits of this form of collaborative iteration planning:

*If you ask a team to choose items from the top of a list that the members believe they can do in a short time-box, the team will probably choose and commit to a reasonable set of features. Once the team members have committed to a set of features that they think they can complete, they will probably figure out how to get those features done within the time-box.*

At the end of the iteration planning meeting, the product owners and development team jointly commit to the iteration plan. Then, it is usually a rule that only the development team can change the scope of the iteration. Product owners must wait until the next iteration to change the direction of the development effort. (In a sea of change, some things must be constant.)

## **Result: The Iteration Plan**

The end result of an iteration planning meeting is an iteration plan that contains:

- An iteration theme—a statement of what the iteration is intended to accomplish.
- A prioritized list of stories to work on for the iteration.
- The stories' estimated tasks and each task's assignments (task owner).
- A commitment by the team to the objectives of the iteration.
- Documentation of the plan in a visible place or in a widely accessible tool.

## **Additional Iteration Planning Guidelines**

In addition, the team should keep the following guidelines in mind during the iteration planning meeting:

- Have the iteration planning meeting on the first day of the iteration, first thing in the morning. The meeting should last no more than 4 hours.
- Create task estimates for each story on the basis of ideal effort hours, or points.
- When estimating, if a story breaks out into seven or more tasks, consider splitting the story.
- Be sure that there is at least one story with a demonstrable function included in the iteration.
- Remember that the product manager owns the story priorities, and the development team owns the tasks and the estimates for those stories.
- When first implementing agile practices, consider setting a "code freeze" a few days prior to the end of the iteration, because the team is likely still practicing waterfall within the iteration.

- Remember that the team's velocity (available resources factored by existing velocity) changes from iteration to iteration.
- Once the iteration is underway, no change requests by the product manager are allowed. Any new or changed stories should go into the backlog.

## Iteration Planning with Distributed Teams

When the teams are distributed, it is preferable to bring the entire team together in one location for planning sessions, especially if the team is new to agile software development or the team itself is new. However, doing so often isn't feasible for teams with staff spread across continents. If your project team has limitations that prevent the team members from gathering together in one location for planning meetings, then attending the release planning meeting (Chapter 12) should be considered the highest priority, followed by the iteration planning meeting.

If using an agile project management tool that holds stories, iterations, and releases, have the team log in. Identify one driver to update the tool as discussions occur, and be sure to limit conversations to one at a time so that everyone can hear what is being said.

If using sticky notes and flip-charts, take pictures and e-mail/post so absent members can see the progress and results. The Agile/Scrum Master should later phone those not in attendance and review the pictures to ensure clear understanding.

---

## Iteration Execution

Having committed to the iteration plan, the team starts development. Each developer (or pair of developers in organizations doing pair programming) will follow the same basic process (see Figure 11–2) repeatedly throughout the iteration until there is no more work in the backlog:

1. **Take responsibility** for an assigned backlog item (e.g., story, use case, defect fix, other).
2. **Develop** (design, code, integrate, and test) the backlog item.
3. **Deliver** the backlog item by integrating it into a system build.
4. **Declare** the backlog item as developed, signaling that it is ready for acceptance testing.

This cycle repeats within an iteration as each developer ultimately takes responsibility for all the backlog items in his or her queue. In most organizations, developers also support the management of the process by estimating actual and remaining effort for the backlog items that they are responsible for.

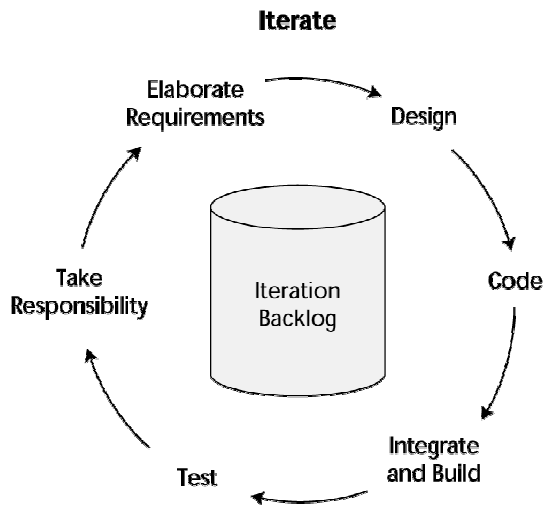


Figure 11-1 Iteration cycle is repeated until all backlog items are completed

## Take Responsibility

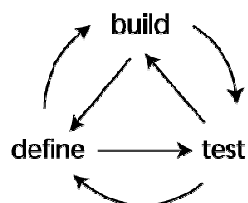
Having committed to the iteration plan, the team is faced with the question of how to allocate that work to the members of the development team. The two basic approaches to allocating work are that the Agile/Scrum Master can assign work to developers or the developers can choose the work they will do.

*When things are happening quickly, there is not enough time for information to travel up the chain of command and then come back down as directives.*

—Poppendieck and Poppendieck [2003]

The process of developers taking responsibility for work must be supported by a visible indicator showing who is responsible for what work and by daily status meetings during which status and issues can be discussed.

## Develop



Once a developer takes responsibility for a backlog item, he or she then:

- Elaborates requirements (if not already elaborated).
- Designs.
- Writes the test (first in some practices) and writes the code.
- Executes the test suite on the build.
- Integrates program code and tests into a build of the system.

As described earlier, because the story is pliable, these activities happen in parallel, and the objective is to deliver a working story (as it evolves) into the baseline. (Definition affects design, design affects test, test affects design, etc.) During a typical development iteration, the developer cycles through most of these activities many times for each story. The order in which these activities occur is a matter of the developer's programming practices and the particular situation. The developer continues the activities until his or her tests for the new functionality or defect pass. At that point, the developer can confidently include the new functionality in a build of the system.

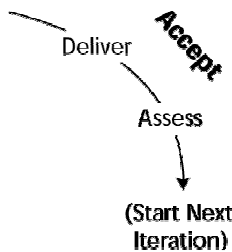
## Deliver Story

The developer delivers the new functionality or defect fix by checking the code into the source control system and including it in a build of the system. The unit test suite and other appropriate tests are run before the code is checked in to ensure that the changes do not break the build.

## Declare Story Completion

Once the developer has integrated his or her work into a build of the system, the backlog item is declared complete, signaling other members of the development organization that the backlog item is ready. For example, the testing group now knows it can include the new functionality or defect fix in its testing efforts and can start automated tests of various kinds (functional, acceptance, performance, etc.).

## Accepting the Iteration



The primary mechanism that allows a team to steer toward its release goals is demonstrating working software early and often to product owners, customers, and hopefully to end users. It is a reality of software development that customers' understanding of their requirements for a software system tend to evolve as they see and use the software.

Thus, every iteration is an opportunity for the team to get feedback and guidance from customers about how to make the system the most valuable that it can be to the customer. This feedback is typically structured as a 1-hour demonstration at the end of the iteration. The format for this meeting is as follows:

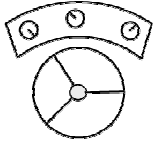
- Presentation of each story by the responsible party
- Discussion and feedback with stakeholders
- Product owners move story to accepted state or split story to be worked on in the next iteration

The final activity in an iteration is to reflect on and assess the results. The goal of assessment is to mine the lessons learned during the iteration and then adapt the development process accordingly. The assessment allows the team to continually improve the throughput of the development process and the quality of the resulting system.

The other major activity that occurs during assessment is a "closing" process whereby unfinished items are put back into the release backlog as work to be done. The iteration is concluded, and the closed iteration backlog becomes a record of the work completed during the iteration.

---

## Iteration Tracking and Adjusting



### Track & Adjust

Underlying the iteration-based development process is the continuous activity of tracking status and adjusting course. Even within the course of a short iteration, scope must be managed, and deviations from plan will occur. The tracking and adjusting activity is focused on getting an objective, real-time picture of where the software development effort is and whether the team is likely to “land” (complete on time) the iteration in process.

Tracking progress of the current iteration requires having visibility into the status of the stories, defects, and other tasks that are being worked on during the iteration. In particular, it’s important to be able to understand how quickly the team is moving through the scheduled work and how accurate its estimates were.

The progress toward the release can be understood by considering the status of the stories, defects, and other tasks across the iterations in the release. Iterative and incremental processes tend to favor a schedule-driven approach, so at the release level, it is most important to understand which chunks of planned work are done and how fast the team is producing work. This information allows the team to deliver the most valuable software on the committed release date by making decisions about what work to do next and what work to defer.

### Tracking in Daily Stand-ups

One of key heartbeats of agile development is the practice of daily stand-up meetings, a daily event that all team members attend, and while remaining standing, relate their status to the other team members. The standing part helps keep the meetings short—stand-ups should run only 10 to 15 minutes. The stand-up’s primary purpose is to quickly share information about the progress being made by each individual in the current iteration.

Participants can be classified as pigs or chickens. The development team consists of pigs: coders, testers, analysts, tech writers, architects, product owners, and the team leader.

### Guidelines for Daily Stand-ups

- Have the meeting at the same time every day; the team decides when this should be.
- Have the meeting in the same location every day; this avoids the frustration of having to secure a room, find the room, and communicate room shifts to team members.
- Make sure everyone stands up; sitting promotes problem solving, discussions that should be held *after* the stand-up.
- Limit the meeting to a 15-minute time box; stick to 1 to 2 minutes per report.
- Start on time; late-comers can adversely affect your ability to keep the meeting to 15 minutes.
- Participants should include all members of the development team: the folks who have their hands on the keyboards and the folks who understand the details behind the user stories.

Scrum prescribes a standard process whereby each team member reports on:

- What I did yesterday.

- What I am doing today.
- What is getting in my way? (Am I blocked?)

Pay attention to what is related—if the tasks on which individuals are reporting are not part of what was committed to for the iteration, this deviation should raise a red flag.

The stand-up is for the benefit of the team, so members should address each other and not just the team leader. The team leader, however, should pay careful attention to how things are going, make sure the team stays within the time box, and then be prepared to afterwards tackle the roadblocks that are preventing team members from doing their work.

## Tracking Iteration Status

Since there are relatively few stories in an iteration, tracking the status of each is a fairly simple matter, and it provides an objective look at progress. Status can be tracked by hanging visual indicators of state information (defined, complete, accepted) on the wall, moving cards from one area of a wall to another, or for larger and distributed teams, using automated tools, as Figure 11–3 illustrates.

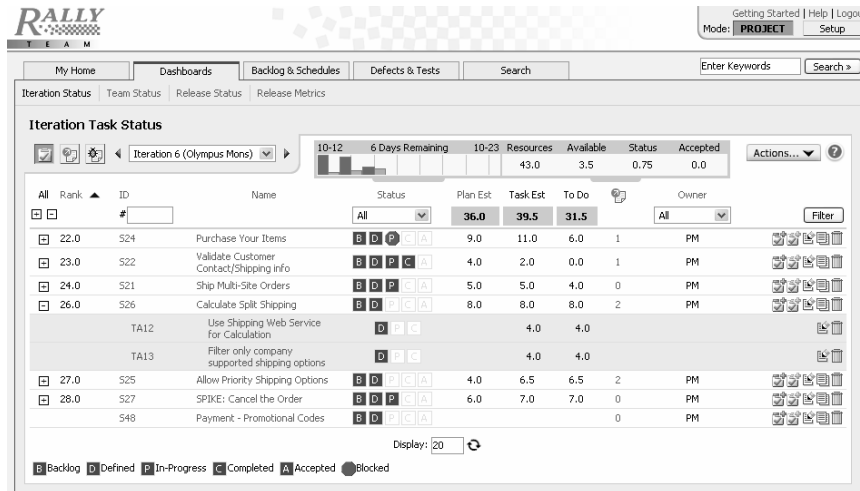


Figure 11–3 Iteration status in an agile project management tool

## Tracking with Burn-Down Charts

Since iterations are fixed in duration, another primary way the team and its managers can gauge progress in the aggregate is to continuously monitor current status and also estimate how much work remains. Computing the estimated remaining work in the iteration at a given point in time requires two pieces of information: the total of the estimates for all backlog items that are not yet started and the estimated remaining effort for any in-progress backlog item. The sum of those two amounts represents the estimated remaining work to be completed during the iteration. Plotting this value each day of the iteration produces what is called a burn-down chart, as the example in Figure 11–4 illustrates.

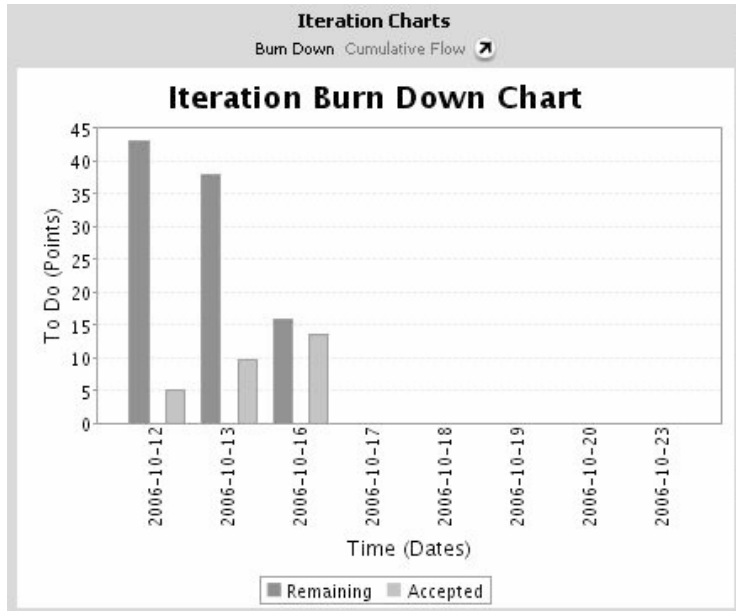


Figure 11-4 Sample burn-down chart

## Iteration Cadence Calendar

As part of establishing a rhythm of agile development time boxes, meetings, and checkpoints, teams typically establish a “cadence calendar” at the start of each project. The cadence calendar helps the team set its schedules so that members can set aside time for planning meetings, daily stand-ups, demos, reviews, and retrospectives.

Figure 11-5 provides a high-level outline of a 2-week iteration, N, as well as the major milestones that occur in that iteration. It also provides a picture of what should happen in preparation for iteration N, as illustrated on the left in N-1.

Release "X"			
Iteration "N-1"		Iteration "N"	
		Day 1	Day 1-10
		Day 10	
<ul style="list-style-type: none"> <li>Verify Iteration "N" priorities</li> <li>Confirm resource assumptions for Iteration "N"</li> <li>Compile requirements, design and testing elaborations in preparation for detailed planning and estimating</li> </ul>		Iteration planning meeting	Design, develop, test, fix, and accept
			Demo and review retrospective

Figure 11-5 Two iterations in a snapshot

Table 11–2 Two-Week Meeting Schedule

<b>Week 1</b>				
<b>Day 1</b>	<b>Day 2</b>	<b>Day 3</b>	<b>Day 4</b>	<b>Day 5</b>
Iteration N–1: demo and acceptance <sup>(1)</sup>	Stand-up	Stand-up	Stand-up	Stand-up
Iteration N–1: retrospective <sup>(2)</sup>				
Iteration N: planning and story review <sup>(3)</sup>				
Iteration N: estimating <sup>(4)</sup>				
Iteration N: refactor and commit <sup>(5)</sup>				

Agendas

- (1) 5-min. demo for each story. Update acceptance. Test automation demo.
- (2) Grade iteration (metrics)  
What went well?  
What didn't go well?  
What will we do different next time?
- (3) Product owner presents stories for next iteration.
- (4) Teams separate and bid stories; take responsibility.
- (5) Team reconvenes, assesses estimates and velocity. Refactor plan as necessary. Commit.
- 

Table 11–2 Two-Week Meeting Schedule (*continued*)

<b>Week 2</b>				
<b>Day 1</b>	<b>Day 2</b>	<b>Day 3</b>	<b>Day 4</b>	<b>Day 5</b>
Mid-iteration review <sup>(6)</sup>	Stand-up	Stand-up	Stand-up	Stand-up
		Iteration N + 1 planning <sup>(7)</sup>		

Agendas

- (6) Assess each story for the iteration; take any corrective action.
- (7) Brief meeting led by product owner highlighting plans for next iteration.
-

---

## About the Book

“Companies have been implementing large agile projects for a number of years, but the ‘stigma’ of ‘agile only works for small projects’ continues to be a frequent barrier for newcomers and a rallying cry for agile critics. What has been missing from the agile literature is a solid, practical book on the specifics of developing large projects in an agile way. Dean Leffingwell’s book *Scaling Software Agility* fills this gap admirably. It offers a practical guide to large project issues such as architecture, requirements development, multi-level release planning, and team organization. Leffingwell’s book is a necessary guide for large projects and large organizations making the transition to agile development.”

—Jim Highsmith, director, Agile Practice, Cutter Consortium, author of *Agile Project Management*

Agile development practices, while still controversial in some circles, offer undeniable benefits: faster time to market, better responsiveness to changing customer requirements, and higher quality. However, agile practices have been defined and recommended primarily to small teams. In *Scaling Software Agility*, Dean Leffingwell describes how agile methods can be applied to enterprise-class development.

- Part I provides an overview of the most common and effective agile methods.
- Part II describes seven best practices of agility that natively scale to the enterprise level.
- Part III describes an additional set of seven organizational capabilities that companies can master to achieve the full benefits of software agility on an enterprise scale.

This book is invaluable to software developers, testers and QA personnel, managers and team leads, as well as to executives of software organizations whose objective is to increase the quality and productivity of the software development process but who are faced with all the challenges of developing software on an enterprise scale.

### Copyright Disclaimer:

This chapter is excerpted with permission from the book, "Scaling Software Agility: Best Practices for Large Enterprises," authored by Dean Leffingwell, published as part of the Addison-Wesley Professional Agile Software Development Series, Copyright 2007 Pearson Education, Inc. ISBN0321458192 For more information, please visit: [www.awprofessional.com](http://www.awprofessional.com). It may not be reproduced, in whole or in part, without express permission from the publisher.