Extracted from:

Create Your Successful Agile Project

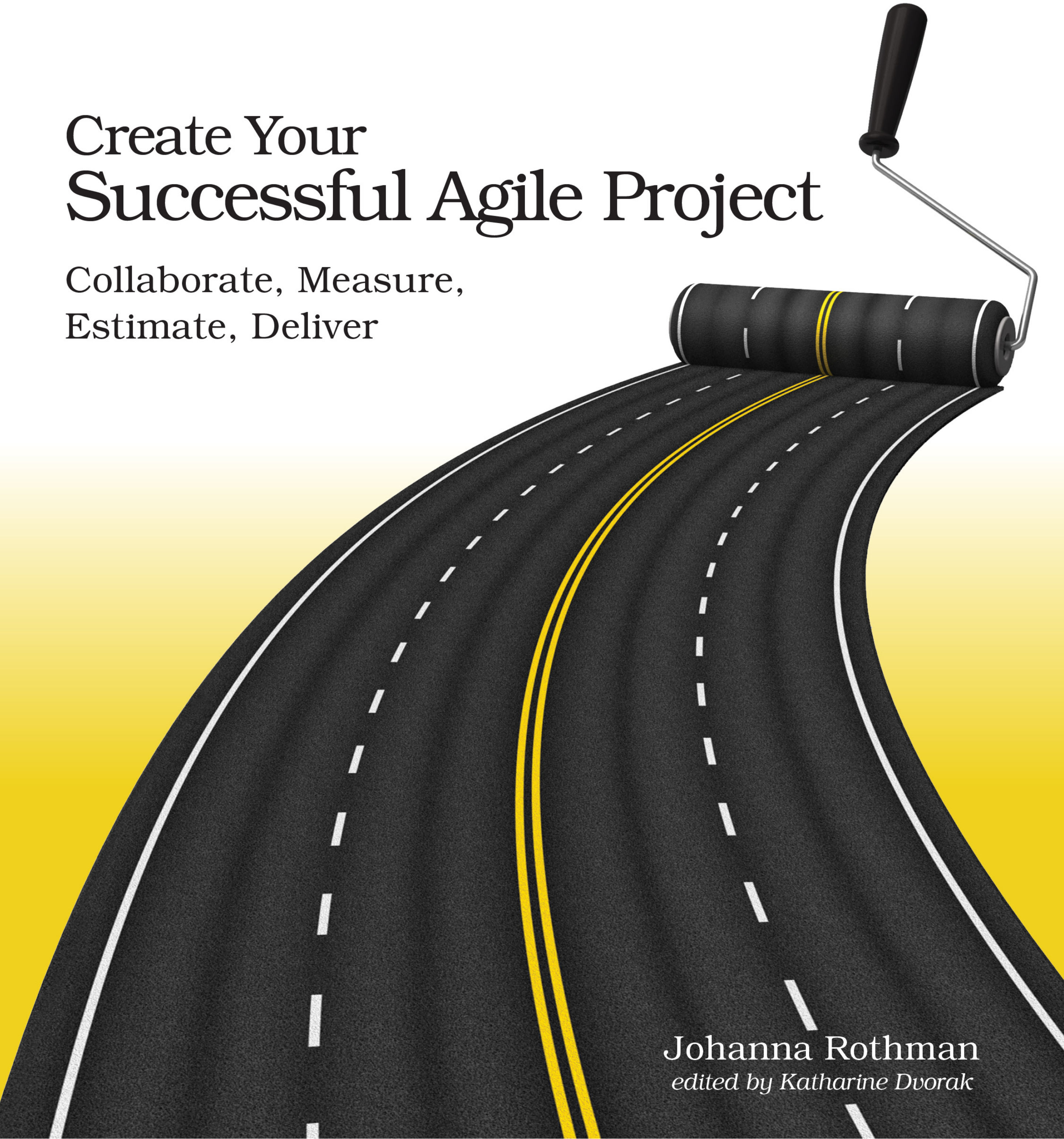
Collaborate, Measure, Estimate, Deliver

The Pragmatic Bookshelf
Raleigh, North Carolina

The Pragmatic Programmers

Create Your Successful Agile Project

Collaborate, Measure, Estimate, Deliver



Johanna Rothman

edited by Katharine Dvorak

Create Your Successful Agile Project

Collaborate, Measure, Estimate, Deliver

Johanna Rothman

The Pragmatic Bookshelf

Raleigh, North Carolina

Printed in the United States of America.
ISBN-13: 978-1-68050-260-2
Encoded using the finest acid-free high-entropy binary digits.
Book version: B1.0—July 19, 2017

The Twelve Principles of Agile Software Development

Agile principles help a team collaborate. If you live up to the principles, you will see increments of your product working every day or so. Those increments allow you to get feedback from your customer and learning within the team. The following list paraphrases the twelve principles of agile software development.[2]

1. Deliver early and often to satisfy the customer.
2. Welcome changing requirements.
3. Deliver working software frequently.
4. Business people and developers must work together.
5. Trust motivated people to do their jobs.
6. Face-to-face conversation is the most efficient and effective method of conveying information.
7. Working software is the primary measure of progress.
8. Maintain a sustainable pace.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. Reflect and adjust at regular intervals.

Part of agile is the idea of sustainable pace and continuous attention to technical excellence. When you build small increments and ask for feedback often, you can welcome change. The change might be in the product or in the team's process. Agile teams fine-tune their work and the product when they reflect and adjust at regular intervals. The principles create the conditions for success, as mentioned in *Agile Is a Cultural Change*, on page ?.

2. http://www.agilemanifesto.org/principles.html

The Two Pillars of Lean

For many teams and organizations, the agile principles are not enough. These teams and organizations need lean thinking to help solidify how agile might work for them.

For many years, people assumed lean was about manufacturing. Because of the Toyota Production System (see for example *Toyota Production System: Beyond Large Scale Production [Ohn88]*), some people thought lean was a bunch of tools, specifically for manufacturing. Lean is much more than specific tools such as a kanban board or a principle such as single-piece flow.

There are two pillars of lean: respect for people and a commitment to continuous improvement. (See the "Lean Primer"[3] for the pillars as well as an excellent and brief explanation of lean.) The two pillars of lean help us create an agile culture. Lean principles apply to knowledge work as well as manufacturing.

Many teams stumble if they try to use only the twelve principles of agile software development when they adopt an agile approach. When people add the lean pillars and lean thinking, they can make agile work better for their context.

Lean thinking—using the lean principles—helps agile teams use agile approaches to create better products and deliver more value. Lean thinking helps people and teams realize they need to visualize their flow of work and think about value as they apply the agile principles. If you're wondering about the source of lean thinking, I recommend reading *Lean Thinking [WJ96]*, *Lean Product and Process Development [War07]*, and *The Toyota Way [Lik04]* and especially *This Is Lean: Resolving the Efficiency Paradox [MÅ13]*.

These are the lean principles of software development from *Lean Software Development: An Agile Toolkit for Software Development Managers [PP03]*:

1. Eliminate waste.
2. Amplify learning.
3. Decide as late as possible.
4. Deliver as fast as possible.
5. Empower the team.
6. Build integrity in.

3. http://www.leanprimer.com/downloads/lean_primer.pdf

7. See the whole.

These principles specifically address the flow of work and the idea that the cross-functional team works together. The agile principles say to work as a collaborative cross-functional team and to deliver often.

Iteration- and Flow-Based Agile

Teams use agile approaches in one of two primary ways: iterations or flow. Yes, you can combine them, too.

An iteration-based agile approach means a team works in timeboxes of the same size for every iteration (as illustrated in the following diagram). The team fixes the duration of the iteration. Teams often work in one- or two-week iterations (the timebox). Every two weeks, by definition, the team is *done* because the timebox is over. The team doesn't change the duration because the team can't estimate what they can complete in a timebox if they keep changing the duration.

Iteration-Based Agile

Requirements Analysis Design Build Test Release Deploy	Requirements Analysis Design Build Test Release Deploy	Requirements Analysis Design Build Test Release Deploy	Requirements Analysis Design Build Test Release Deploy	Repeat as needed ...	Requirements Analysis Design Build Test Release Deploy	Requirements Analysis Design Build Test Release Deploy

Each timebox is the same size. Each timebox results in running tested features.

In iteration-based agile, the product owner and the team manage the work in progress by estimating the number of stories (and other work) the team can commit to in a timebox. When the team estimates, the product owner receives feedback on the estimated size of the work. The product owner then has choices to make more stories or ask the team to swarm or mob on the work.

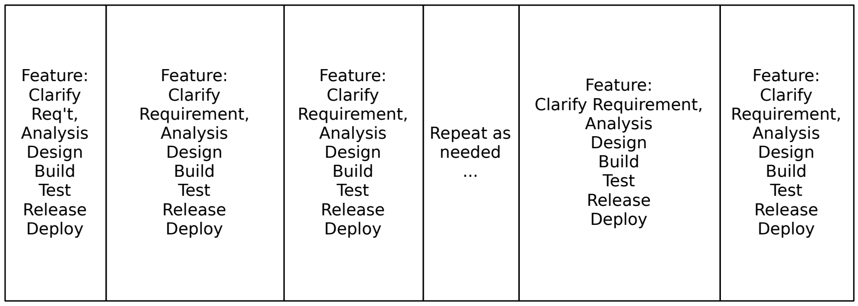
Note that I have included all of the work in a timebox: Requirements, Analysis, Design, Build, Test, Release and Deploy. Those are the activities a team requires to deliver finished value. (Sometimes, teams release internally but do not release to customers or deploy each iteration.)

You might think a team does these activities sequentially. Not necessarily. The team often performs these activities as a team, on one or two features at a time. I'll explain more in Chapter 6, *Teams Deliver Features*, on page ?. The team performs all these activities, but not necessarily in sequential order for a given feature. Here’s a quick example: During a planning meeting, the team—as a team—discusses a couple of possible designs for a given feature, because the time needed might change depending on the design. A tester might sketch some possible tests. Even before the team “starts” work on that feature, the team estimates, designs, and tests—just a little bit. That’s what I mean by a non-sequential approach to the work.

An iteration-based agile approach provides a cadence—a project rhythm—for teams to deliver and learn, retrospect, and plan.

In a flow-based agile approach, shown in the next diagram, the team maps the flow of value through their team. The team sets a WIP limit for each column on the board and tracks their cycle time—how long features take on average. The team and the product owner manage the work based on those limits. After finishing some work, the team delivers and learns, retrospects and reviews what they want to improve.

Flow-Based Agile


Feature: Clarify Req't, Analysis Design Build Test Release Deploy
Feature: Clarify Requirement, Analysis Design Build Test Release Deploy
Feature: Clarify Requirement, Analysis Design Build Test Release Deploy
Repeat as needed ...
Feature: Clarify Requirement, Analysis Design Build Test Release Deploy
Feature: Clarify Requirement, Analysis Design Build Test Release Deploy

In flow, the team limits the number of features active at any time with WIP limits for each team activity. There is no timeboxing built into flow.

Flow focuses on the continual pulling of work; iterations more often focuses on pulling a limited set of work into a defined timebox.

Neither flow nor iterations is right. Neither is wrong. It’s all about what your team needs to see the work, release valuable product often, and get feedback.

I happen to like a flow-based approach inside of some cadence. I like seeing working product *at least* as often as every two weeks, which is what I do for my collaborative projects. I deliver value more often for my personal projects—at least once a day. I want to see where the work starts, where it waits, how long it waits, if there are any cycles, and so on. Flow and kanban boards can show you that. Iterations—by themselves—don’t show you details of where work is stuck.

Distinguish Between Cadence and Iteration

I said that iterations are timeboxes of a week or two, maybe longer if you don't need more frequent feedback. I also said that a cadence provides a rhythm for a project. Let me explain the difference.

Many teams appreciate a cadence for delivery, retrospectives and more planning. Not every team needs the focus of a timebox to do that.

One team I know delivers several times during the week. They plan weekly, but not the same day each week. When they've finished three features, they plan for the next three. It takes them about 20-30 minutes to plan. It's not a big deal. This team retrospects every Friday morning. (I would select a different day, but they didn't ask me. See *Organize the Team's Meetings*, on page ? to see why I prefer mid-week cadences.)

Notice that they have two separate cadences: at least once a week for planning work, but not the same day; and once a week for retrospectives on the same day each week. They are not working in iterations; they work in flow. They use the idea of a cadence to provide a pulse, a rhythm for their project.

If they used iterations, they would always plan on the same day, at the beginning of the iteration. They would always have a retrospective on the same day at the end of the iteration. This team doesn't do that and that's great. Teams don't have to follow prescribed ceremonies, especially if the team's context is different from other teams.

A cadence is different from an iteration. Decide what fits for your team.

Integrate the Agile and Lean Principles

Think about what the agile and lean principles together buy you. These principles say to use collaborative cross-functional teams so that the entire team works on features. The principles emphasize seeing the work as it proceeds to get feedback on the work and the process. The principles caution you to not start more than the team can complete in a short period of time. And you deliver working product often and as fast as possible to see progress, increase customer collaboration, and receive feedback.

How can you think about the agile and lean principles for your project? How can you build respect for the people and continuous improvement into how your cross-functional, collaborative team can deliver small chunks of value often? That way, not only can your team deliver often, but your team can also receive feedback often.

Consider How an Agile Approach Might Work for You

I often hear people describe their projects as being "Agile/Scrum." Let me clarify: "Agile" is an umbrella term that encompasses many agile approaches, one of which is Scrum. Some of those approaches are defined in the following table.

Named Approach	How the Approach Works
eXtreme Programming	Primarily a collection of technical practices guided by these values: communication, simplicity, feedback, courage and respect.
Scrum	Timebox-based project management framework to delivering working product often.
DSDM (Dynamic Systems Development Method)	Timeboxed approach to delivering functionality. Facilitated workshops to determine the requirements and gain agreement on them.
Crystal	Focus on the people. Depending on the size of the project team and the product criticality, select the approach that fits for the team, the business people, and customers.
Feature-Driven Development	Deliver functionality incrementally after creating a (low fidelity) framework for the architecture or object modeling. Focus on building value for the customer.
Kanban	Visualize the flow of work, work by value, and manage the work in progress. Deliver incremental value as the team completes the value.

Table 1—Some Agile Approaches

There are more agile approaches, but these are well-known approaches for teams.

Scrum[4] is one project management framework that helps a team adopt agile techniques. Many teams start with Scrum because it is the most famous approach to agile. Scrum works for collocated teams who work on one project at a time, and where the team has all the cross-functional skills and capabilities the team requires. And Scrum creates the need for cultural change. However, I have found teams that meet the following criteria are not good candidates for Scrum:

4. http://www.scrumguides.org/index.html

- Your team works on more than one project at a time. Or, the team needs to provide significant interrupt-driven support or maintenance.
- Your team is distributed across more than four time zones.
- Your team is not cross-functional. That is, you have a team of developers trying to collaborate with a team of testers, often across time zones. Or, you have a workgroup instead of a team.
- Your team does not have the skills or capabilities it needs. You have a scarcity of some necessary role, such as UX or DBA.
- Your "team" does not need collaboration. All the work is independent, not interdependent.

In these cases, I do not recommend Scrum as your agile approach. (Can you make Scrum work? Of course. It's more difficult than an agile approach has to be, but you can try.) I recommend you integrate flow into your agile design. You might also use iterations for a cadence of planning and delivery.

As described in *Iteration- and Flow-Based Agile*, on page 3, iteration-based agile uses timeboxes to manage the scope of work the team will complete. Scrum is similar, and also offers specific roles, such as the Scrum Master and product owner. In addition, Scrum incorporates several events, including:

- The daily standup.
- The pre-iteration planning meeting (which Scrum calls the Sprint Planning meeting).
- The backlog refinement meeting before the next iteration.
- The demonstration to show and explain what the team completed in the last iteration (in Scrum, this is the Sprint Review and includes the team's assessment of whether they achieved the sprint goal).
- The retrospective at the end of the iteration.

You might decide these events are for you. However, you don't need to use Scrum to use an agile approach—even an iteration-based approach—that works for you. You might find that given your context, choosing an alternative or designing your agile approach works better.

I have seen people and teams use the agile and lean principles progressively move to agile approaches as it fits for them. I recommend you do, too. The more your team is resilient and adaptive, the easier it is to create a great project that delivers a successful product.

You do not need to name your agile approach. I recommend you design an agile approach that incorporates the values and principles of agile and lean, rather than try to stick to a framework, regardless of its fame.