Extracted from:

Create Your Successful Agile Project

Collaborate, Measure, Estimate, Deliver

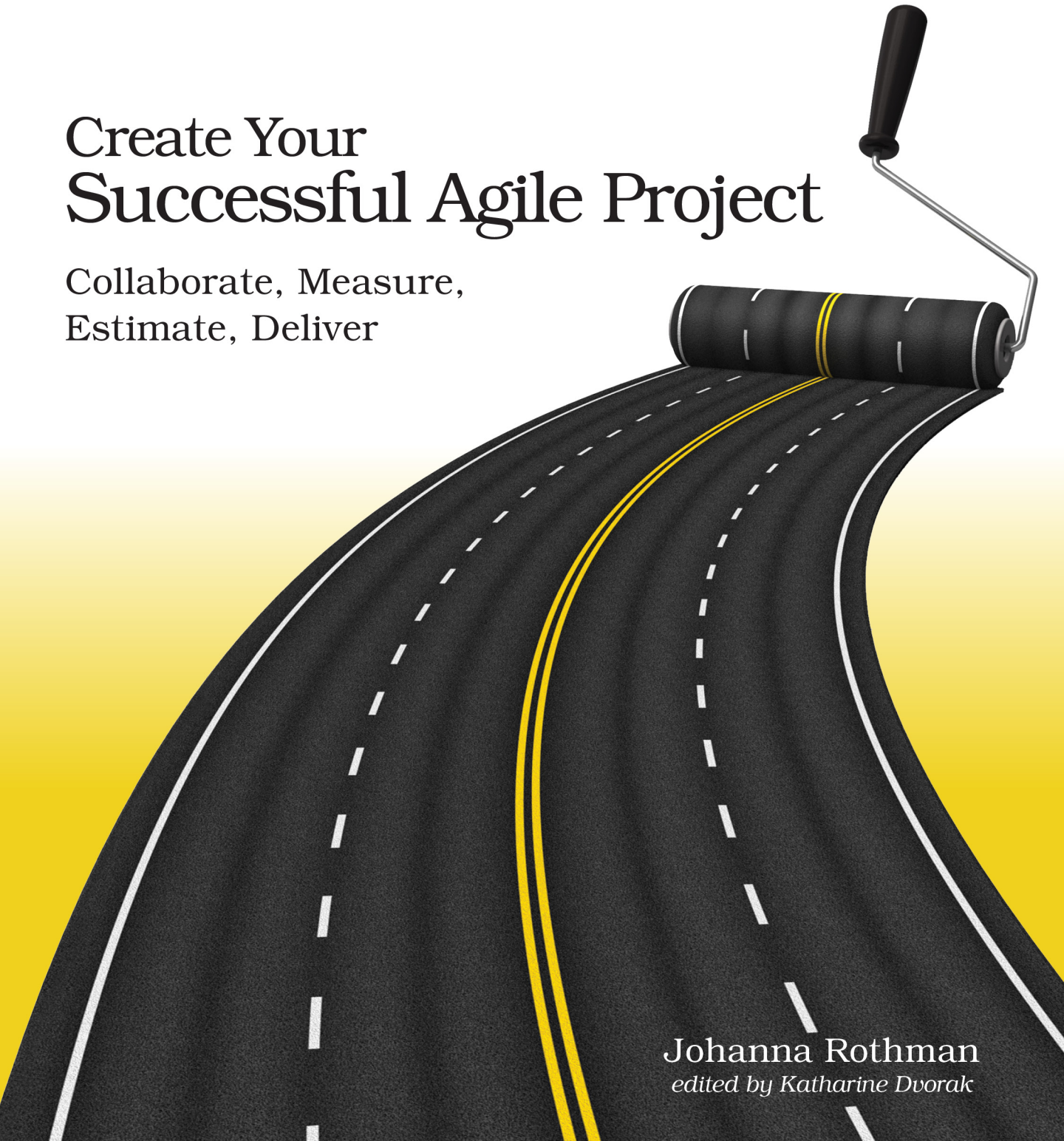
The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Create Your Successful Agile Project

Collaborate, Measure, Estimate, Deliver



Johanna Rothman
edited by Katharine Dvorak

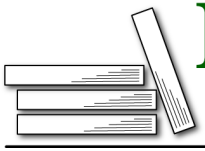
Create Your Successful Agile Project

Collaborate, Measure, Estimate, Deliver

Johanna Rothman

The Pragmatic Bookshelf

Raleigh, North Carolina



Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Development Editor: Katharine Dvorak
Indexing: Potomac Indexing, LLC
Copy Editor: Candace Cunningham
Layout: Gilson Graphics

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

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: P1.0—October 2017

To Mark, as always.

Velocity Is a Capacity Measurement

I already said velocity is not acceleration back in *Understand Velocity*, on page ?. Velocity can be a measure of capacity once the team is stable and understands how to work together.

A team's velocity (regardless of whether it uses points or story counts) will vary while it learns how to work together. (Many people think it takes somewhere between six and nine iterations for a team to learn how to work together.) A team's velocity might vary under other circumstances, as well:

- The team is learning a new domain.
- The stories vary in size and may be quite large.
- The team changed the iteration duration.

There are two potential drawbacks to measuring velocity. I said that velocity is the rate of change. That's what you deliver (features or story points) over time. If you change your iteration duration, you change the time, one of the inputs to the rate of change. If you use story points and you decrease or increase the size of the stories, you change one of the inputs to the rate of change. If you maintain the same iteration duration and use features instead of points, your velocity measurements will be more accurate.

Measure Cycle Time to Understand Capacity

Many teams, especially those working in iterations, measure their velocity in story points to understand their capacity. Instead of points, consider measuring cycle time. When the team measures its cycle time—possibly different times for stories, fixes, and interruptions—the team has a better idea about its capacity. For me, points are too variable, especially in a team new to agile approaches.

Agile Approaches Change the Meaning of Defect Measurements

Assuming the team meets the acceptance criteria on a story, the team won't have defects during the iteration or while it works on the story. Any defects the team finds before it decides the story is done is work in progress. It's not until the team says the story is done that we need to count defects.

This means the defect counts should be much lower than on a more traditional project. However, the product owner and the team might not understand how

the user will use the story. It's possible that in the entire product context, the team made a mistake and created a defect. One mistake I've seen is the product environment: the team assumed one environment and the customer uses the product in a different environment. That's a design defect.

You might find several defect measurements helpful: defect escape rates, the cost to fix a defect, the defect cumulative flow, and the fault feedback ratio.

Measure Defect Escapes

Sometimes the team doesn't fully understand some aspect of a story. Or maybe the product owner didn't fully explain the acceptance criteria. Or maybe the product owner or the team made a mistake writing the story and the acceptance criteria. Any number of things might occur and a defect escapes the team, even though the story met the acceptance criteria and the product owner accepted it.

Measuring the team's defect escapes can help the team see what's going on.

I would hope that the number of escaping defects is zero. However, if you are starting your agile journey, you might encounter several traps that prevent you from keeping your defect levels to zero or close:

- *Trap: Developers and Testers Work in Successive or Staggered Iterations*, on page ?
- *Trap: Your Team Has No Product Owner*, on page ?
- *Trap: You Have Iterations of Waterfalls*, on page ?

When you measure the number of defects that escape over time, you can then perform a retrospective to see what you might do.

In an agile approach, escaped defects are not the norm. If you see defects escape, address that as an impediment to fix.

Measure Your Cost to Fix a Defect

The software-engineering literature has references to exponential costs through the life cycle for how much it costs to fix a defect: If you have a cost of 1 in requirements, it's now 10x in analysis, 100x in coding, 1,000x in testing, and a whopping 10,000x post-release. Those costs have been debunked by *The Leprechauns of Software Engineering [Bos14]*.

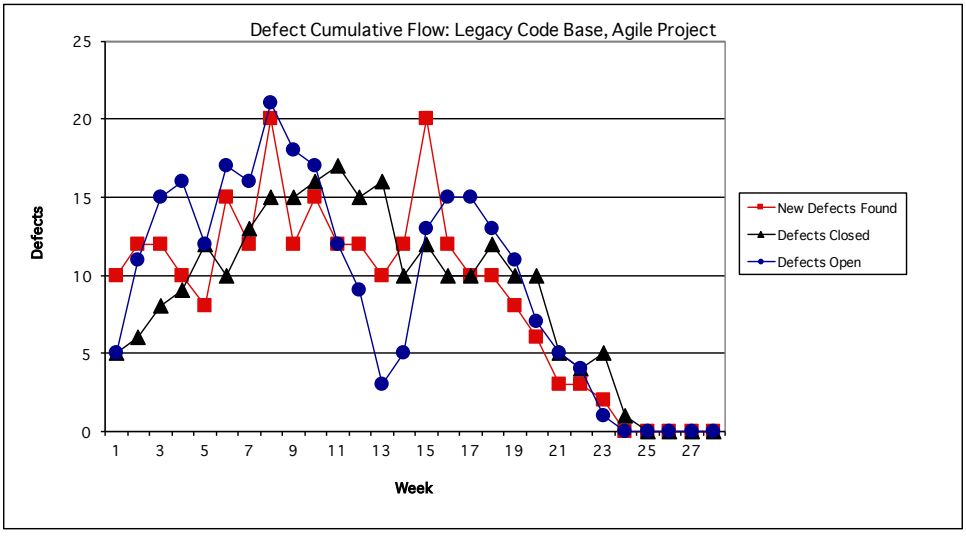
My experience is that it does cost more, but not nearly that much. Even in one of my projects that did very little review or testing, the actual cost post-release was 16x. The problem was that there were so many defects that the

overall cost to fix a defect was quite high. And given the number of defects, it was quite difficult to release and make progress on the *next* project. (See *Manage It! [Rot07]* for more details.)

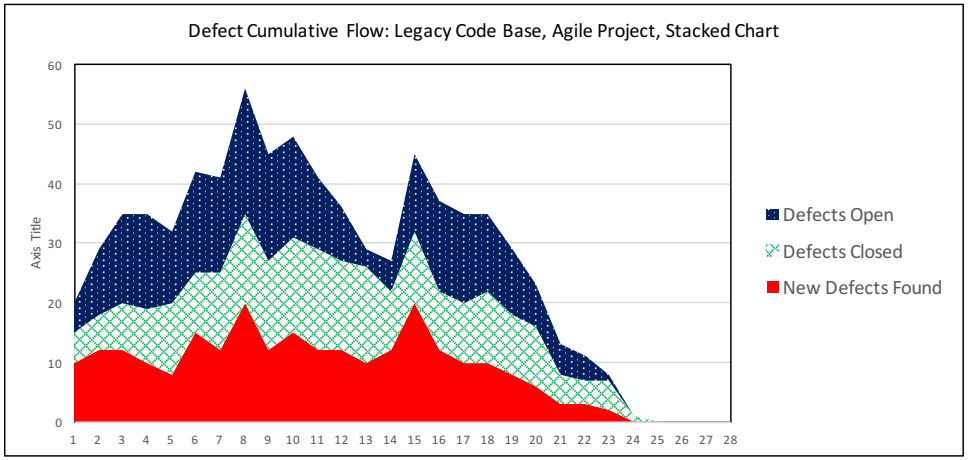
The more the team works in a way to promote technical excellence, the fewer defects the team will see. That will make the number of escaped defects low. It should keep the cost to fix a defect low. If your team has trouble with escaped defects, start to measure the cycle time of defect-fixing. Once the team has the data, ask what actions the team might consider in a retrospective.

Measure the Defect Cumulative Flow

Especially if you have a legacy product where defect discovery overwhelmed the team, you might need to measure defect cumulative flow. The following chart illustrates the defects found, closed, and remaining as a line chart.



Your team might prefer a stacked chart, such as the one shown in the following figure, so that it can see the contribution from each part of the cumulative flow.



This team discovered previously undetected defects as it progressed through the project. That's because the team increased its test automation and refactored the code base as it worked on one small feature at a time.

Some weeks the team discovered 15 to 20 defects. The team learned, as it added those defects to the backlog, that it had cascading defects. It discovered and fixed one defect. Then it discovered that defect masked several more underlying defects.

As the team progressed through the project, it realized that it was able to stabilize the code base. As the number of discovered defects went to zero, the team's capacity for more features increased.

The team used this chart to help its management see that the team had unfinished work. It had debt from previous projects that it paid off in this project.

Measure the Fault Feedback Ratio

I bet you've seen problems that the team swore it fixed—and the problem returned. The team fixed it in one place and it popped back up in another place. What the team thought fixed the problem didn't. The team (and possibly the PO) rejects that fix.

We can measure the ratio of rejected fixes to total fixes. This is the fault feedback ratio (FFR).

The FFR is the ratio of the number of rejected fixes (fixes that don't actually fix the problem) to the total number of fixes. My experience is that an FFR of more than 10% says the developers don't make enough progress on finding and fixing the problems. They spend their time trying to discover the cause(s)

of the problems and creating even more tests. I've also seen this occur when fixing one problem uncovered more problems.

If the team discovers defects as a matter of course, here are some possible actions to offer the team:

- Discuss the defects and their causes at a retrospective. What creates the defects?
- Is the team's definition of "done" sufficient for these kinds of problems? (See Chapter 11, *Know What "Done" Means*, on page ?, for more details.)
- Can the team pair or mob on the defects to discover the root cause and fix it?

The team may have other ideas once it realizes it has a problem that doesn't want to stay fixed.

If the team accomplishes the acceptance criteria (what "done" means for a story) and meets the "done" criteria for an iteration, the team tends not to have many defects. That means the escaped defects should be quite low.