Extracted from:

Fixing Your Scrum

Practical Solutions to Common Scrum Problems

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Fixing Your Scrum

Practical Solutions to Common Scrum Problems

Ryan Ripley
Todd Miller
edited by Dawn Schanafelt

Fixing Your Scrum

Practical Solutions to Common Scrum Problems

Ryan Ripley
Todd Miller

The Pragmatic Bookshelf
Raleigh, North Carolina

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Executive Editor: Dave Rankin
Development Editor: Dawn Schanafelt
Copy Editor: Sean Dennis
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics

ISBN-13: 978-1-68050-697-6
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—January 2020

To my wife, Kristin. You make it all worth it.

—Ryan

To Jess, who makes the impossible possible for me.

—Todd

Todd worked with a Scrum team that was adding new features to a fitness application. They had just completed their second sprint review, with many stakeholders present. There were a few tense moments between the stakeholders and the product owner but overall the sprint review was a success.

Immediately afterward, the development team and Scrum master gathered for the sprint retrospective. Here's how it unfolded:

> ***Matt [developer]:*** *I'd really like to try some pair programming next sprint. I'm struggling a bit to understand the one pattern that we're using. I mean, I'm doing it, but I'm just not sure it's the best way. It would be nice to have an extra set of eyes.*
>
> ***Peter [developer]:*** *I disagree. We're getting a ton done working the way we are now. You saw it—the stakeholders and product owner are happy, so why mess with our approach? Matt, I'll send you an online article about the pattern. You should be understanding this by now. I don't understand why you're not picking it up.*
>
> ***Other developers:*** *[Silent, awkwardly staring down at notepads.]*
>
> ***Scrum Master:*** *This seems like an opportunity, perhaps a chance to try some pair programming. Who wants to try that with Matt?*
>
> ***Peter:*** *I'm out. I don't have time for that—it's a complete waste of time. My improvement for this next sprint is for us to get better at estimating. I've got to go. It's time for lunch, and my time is better spent coding than sitting in meetings. You're the Scrum master—you keep the product owner and stakeholders happy while we code. Your job is to keep us out of meetings. My job is to actually deliver things. [Storms out.]*

Every sprint after that, the tension kept rising until development team members rarely spoke. The product they delivered to the customer reflected the lack of communication within the team—the application was buggy and the user interface wasn't cohesive. After several conversations and chances, the customer fired the entire team because of their inability to get the job done.

Each sprint retrospective is a chance for the Scrum team to reaffirm its commitment to continuous improvement. A team uses the time to reflect on team dynamics, quality, defining "done," and seeking ways to improve the way we deliver potentially releasable products every sprint. Without approaching this event openly and finding ways to improve every sprint, your product, its quality, and the people working together on it will suffer.

Scrum is just a framework. People are the power behind the roles, events, and artifacts. If we as a team are not collaborating, helping, supporting, and challenging ourselves to improve, we're missing a crucial opportunity to improve the way we serve our customers.

In this chapter, we explore some of the sprint retrospective anti-patterns that Scrum teams experience. Remember that there are two sides to Scrum: the people and the technology—and during a retrospective, the team needs to explore both.

Few Bother to Attend

Some Scrum teams take an unenthusiastic approach to the sprint retrospective. For example, the product owner we mentioned at the beginning of this chapter attended the sprint review but not the sprint retrospective. That was a big missed opportunity, as the retrospective would have been the opportune time for the PO to talk to the development team about how to mitigate future stakeholder surprises. Similar problems may pop up if development team members don't attend the retrospective.

The sprint retrospective is an activity for the *whole* Scrum team. It should include all development team members, the product owner, and the Scrum master—every time. The only way the team can improve is if the whole team engages in this event. Dysfunction cannot be fixed when there is only partial attendance.

Also, keep in mind that this event is for the Scrum team *only*. When stakeholders or managers outside the team attend, that makes it harder for the team to effectively collaborate. It's hard enough to get a group of people from different backgrounds, cultures, and experiences to collaborate. Adding another layer of difficulty doesn't help.

Consider the reason behind the sprint retrospective: You're meeting to inspect the way the Scrum team works together, as team members strive to deliver a done increment of product by the end of the sprint. The adaptations (a.k.a. improvements) that the team agrees upon and commits to during the retrospective can help the team deliver value to customers more quickly.

You should also use the sprint retrospective to consider quality. Discuss the current definition of done and decide whether it needs to be modified based on what you learned about your product during the sprint review.

If the entire team isn't present for these discussions, you could lose transparency in the way you're working; any experiments that you try might not get the intended results. So get your entire Scrum team to attend the retrospective.

Joe asks:

When should the sprint retrospective happen?

It should always happen after the sprint review but before the next sprint planning. That's because, if you haven't inspected the increment and how you're collaborating with stakeholders, then you haven't fully inspected the sprint. This can lead to adaptations that aren't appropriate for your situation.

As far as duration, The Scrum Guide states that a sprint retrospective time box is a maximum of three hours for a one-month sprint. Adjust that time box accordingly based on your sprint length.

Superficial Commitments

Let's say a team has a decent sprint, and after inspecting the increment during the sprint review, the team members gather for the retrospective. They collaborate on a list of ideas that could help them improve the way they work. Then they select two items that they think will help them improve during the next sprint:

1. Update the definition of done to include code reviews before checking code into source control.
2. Restrict coffee talk to outside the team room because it's distracting to some team members.

The next sprint starts, but the team completely ignores the two improvements they committed to. Despite the Scrum master reminding them about it, they've become lost in the work and are too busy to implement the code review change. And coffee chatter happens frequently in the team room—in fact, it's more noticeable than ever.

This pattern continues for several sprints. Eventually code quality goes down and the production bug rate spikes. The team is also experiencing a rise in conflicts. They aren't improving. In fact, in some ways, they're moving backward.

It's important that the Scrum team live by the Scrum values in everything they do. Team members must truly commit to the improvements that the team identifies in the retrospective. Team members must commit to one another on making the improvements. Ignoring these commitments is a clear sign that the team hasn't embraced a mindset of continuous improvement.

If your team has fallen into this trap of creating superficial commitments but not following through with them, you can use a few techniques to try to fix that. During the next retrospective, begin by asking members how the team did with their previous improvement items. Remember, these improvement experiments are sprint backlog items, so the team should consider them each day during the daily scrum and perhaps even discuss them during the sprint review.

If the team didn't make progress, then you need to talk about why. Did the team forget to account for the time it would take to complete the improvement item during sprint planning? Perhaps it was unclear how the team would make or measure the progress of the experiment? In any case, the team needs to have an open and respectful discussion to help team members commit to improving during their next sprint.

Joe asks:

How many improvements are reasonable?

An ambitious team might identify many improvements they wish to work on in the next sprint, but committing to an excessive amount might mean that none actually get accomplished. Keep in mind that improvements impact the team's capacity during the sprint. Taking on more than one or two improvement items may not lead to the business outcomes that the product owner and stakeholders are expecting. Scrum teams should select just a few key improvements that they believe they can implement during the next sprint. Work as a team to determine the right number of improvements for each sprint.

Meaningless Improvements

Let's suppose the Scrum master facilitates an interactive session with the whole team in attendance, and team members respond with some ideas for how they can get better. Here's what they came up with:

- Have more fun
- Go to lunch once a week
- Write better code
- Estimate with more accuracy
- Collaborate

Are these actionable improvements? Will this team really get better if they choose to implement one or two of these items in the next sprint? The items are awfully vague and they don't make for concrete improvements. It's great that the entire Scrum team attended the sprint retrospective and that team members are generating ideas about how to improve—but now it's time to delve into the team's problems to find root causes and measurable goals and objectives for the next sprint.

The retrospective is about examining people, relationships, and technical practices that are inhibiting the team. Let's review the items that the team came up with, and add some follow-up questions you might ask to dig a little deeper into the problems:

- Have more fun
  - Is someone killing all the fun?
  - What's preventing this team from thinking work is fun?
  - When's the last time the team was shown appreciation during a sprint retrospective?
- Go to lunch once a week
  - Is everybody getting along?
  - What are we going to solve by going to lunch?
  - Are we not communicating well enough?
- Write better code
  - What's our definition of quality?
  - Are we writing bad code?
  - Is the architecture still appropriate given what we know now?
  - Should we revisit our definition of done?
- Estimate with more accuracy
  - Why are we estimating?
  - What did we estimate incorrectly?
  - Did our bad estimate set unrealistic expectations?
  - Are there impediments or bad practices that impact our ability to estimate?
- Collaborate
  - Are we helping each other?
  - What happened between the development team and product owner?
  - Are we missing opportunities to work together?
  - Are we appropriately refining the product backlog?

Empty improvements are the easy way to get out of a retrospective quickly, but they invoke no real improvements. The team has to dig deep into relationships and/or technical issues in order to reach its maximum potential. The retrospective should be as serious and professional a meeting as the sprint review. It's the time to bring up issues and attempt to find resolutions.

As a Scrum master, you'll need to flex your facilitation muscles and discover what questions lead to the right conversations to help reveal the deeper issues. Keep digging—your teams need you.