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

In 2016, Todd got a call from a product owner who was struggling. His Scrum team was using five-week sprints, with four weeks designated for work and the last week dedicated to retrospectives and planning. You might be thinking this way of doing sprints isn't actually Scrum, and you'd be right. Per The Scrum Guide, sprints are limited in duration to one calendar month or less, and committing a week to retrospectives and planning is highly unusual.

The product owner in this situation was having a really tough time. She had no transparency into the sprint backlog and couldn't forecast a release plan or make even basic assumptions about the progress being made on the work. The planning/retrospective week was especially painful for her: There were long meetings, lots of indecision and arguments, and she had little ability to forecast the next sprint. Most importantly to the PO, she had no idea of how to set stakeholder expectations.

As Todd dug in and started asking questions, he discovered that the product owner wasn't the only one in pain. Everybody on the Scrum team was miserable. Development team members felt lost and often didn't know what to work on next. Some had nothing to do the first two weeks of a sprint and then got slammed the last two weeks, putting in overtime to finish their work. The opposite was true for others on the dev team.

Daily scrums were status meetings, and often ran 45 minutes to an hour as the Scrum master showed up with a list of tasks to check on. The Scrum master didn't know Scrum; he was simply thrust into the position because he had a project management background, and the organization's management team had set the expectation that everything needed to be accomplished in an agile way.

The fix was simple and effective (although, due to the organization's culture, it was difficult to implement): They just needed to start *truly* using Scrum.

All the required Scrum events (sprint, sprint planning, daily scrum, sprint review, and sprint retrospective) are time-boxed. The duration of each event should be tailored to your environment. Here are The Scrum Guide's guidelines on how long each event should last:

- Sprint - one month or less
- Sprint Planning - a maximum of eight hours for a one-month sprint, usually shorter for shorter sprint lengths
- Daily Scrum - a maximum of 15 minutes regardless of sprint length

- Sprint Review - a maximum of four hours for a one-month sprint, usually less for shorter sprint lengths
- Sprint Retrospective - a maximum of three hours for a one-month sprint, usually less for shorter sprint lengths

The sprint length determines how often you receive feedback about your product increment, and the cadence and frequency of all the other Scrum events. The sprint is a powerful tool for a Scrum team that is often misunderstood and misused. Consider the following benefits of sprints:

- The sprint time box creates focus on doing only the things that are essential to completing the sprint goal, and getting features released to your customers. Any distractions that are not related to your sprint goal become an easy “no”—you don’t have time to waste.
- Working and thinking in sprints will expose many problems with the way your Scrum team currently works. Every bottleneck, inefficient process, and ineffective practice will come to light as the team tries to get a feature completed, tested, integrated, and deployed by the end of the sprint. This provides excellent opportunities for your team to identify and resolve impediments that could have gone undetected or unresolved if you were using another framework or methodology.
- The sprint represents a commitment to Scrum. By working in sprints, the team has decided to provide multiple opportunities to inspect and adapt progress toward the sprint goal. This commitment allows the team to get feedback frequently and maintain alignment with the customers’ needs.
- Working in sprints gives your business partners transparency into the team’s progress, and empowers the stakeholders to manage cost and release schedules with real data, clear information, and product increments to inspect at the end of every sprint.
- Sprints give your team members opportunities to let go of limiting beliefs about what is possible and to try new approaches. While this may be uncomfortable at times, with the help of the Scrum master (you!), they can move through these situations and grow as developers and team members. The momentary discomfort is worth the long-term benefits.

The sprint is the container for all the other Scrum events. Understanding the purpose of the sprint, knowing common pitfalls to avoid, and understanding how to avoid them is pivotal to the success of your Scrum team.

We Need a Special Sprint

Some teams that are new to Scrum have trouble moving away from waterfall-style techniques such as phase gates and big upfront planning. Specialized sprints can be appealing to such teams because they model old, familiar processes and behaviors that have been in place for many years. But if any of the following specialized sprints sound familiar or even just tempting, your Scrum team or organization may have underlying issues that you need to work on.

- Sprint Zero - Also called architecture or planning sprints, the intent of this type of sprint is to create a robust plan, often in the form of a product backlog, that the team can use to get started.
- Design Sprints - The result of these sprints is a design output such as user experience or architecture. There's no business value delivered during these sprints, only a vision of what future business value *might* look like, such as user interface mockups or a database design.
- Development Sprints - Sprints where the only thing achieved is writing code. All other tasks, such as design, testing, and integration, are put on hold or handled by other teams.
- Test Sprints - Sprints where code written by another team is tested. Bugs and issues that are found are reported back to the team that created them.
- Integration Sprints - The output is code merges or attempts at completing integrations with other systems that the system is dependent upon.
- Hardening Sprints - These are for code clean up, bug fixes, integration, testing and anything left over, to make sure the system is ready for production.
- Release Sprints - Often used as the final phase, a release sprint is where software that has been designed, written, and tested meets the infrastructure team for deployment into production.
- Bug Sprints - After code is shipped into production and bugs start rolling in, these sprints are designed to patch and fix them, and then deploy those fixes back into production.

These specialized sprints all map back to waterfall-style techniques. In a waterfall world, you perform big up-front design (BUFD) work to define the product and requirements in a design phase. Then you move into an architecture and infrastructure phase, followed by a development phase. Hopefully, the testing, integration, and hardening phases come next. Then you finish up with a release phase and ship the product to production. During a post-production phase, you squash the bugs as they're reported.

Scrum is different. The sprint is a boundary. Within a sprint, the Scrum team performs sprint planning, the daily scrum, the development work, the sprint review, and the sprint retrospective. The outcome of a sprint is a production-ready increment of the product. These increments can be a major competitive advantage because, in four weeks or less, you have production-ready software. The next sprint begins right after the end of the previous sprint. *All* of the phases of each project occur within a sprint. If it helps, think of a sprint as a mini-project where the work is asynchronously executed by the development team. By the end, your team must yield a valuable and potentially releasable increment, no matter how small.

Sprint zero is one of the most popular specialized sprints. Teams often claim they need a sprint zero to take care of things that needs to be done before the first sprint of a project can begin. For example, a team might claim that they need a full product backlog before they begin the first sprint.

The problem with having a sprint zero is that the team establishes a precedent of allowing special types of sprints that don't deliver business value. If the development team's goal for a sprint is creating a product backlog, they're obviously not delivering a potentially releasable increment during that sprint. When business value isn't on the line, teams can be tempted to extend sprint zero a few days, weeks, or months. As a Scrum master, if you allow these behaviors, you're opening the door to the team using similar arguments in the future when delivery is on the line.

Joe asks:

What do I need to start a sprint?

Scrum doesn't prescribe what you need to have in place to begin your first sprint. All you need to get started with sprint planning is a product owner, a development team, and a few high-level product backlog items. Is this ideal? No. But these roles and artifacts are enough to get started initially.

Once you get started, you'll find out a lot of things that you had no way of knowing —valuable info that no amount of planning or studying could have unearthed. So don't fear getting started, fear *not* starting at all.

Figuring out why your team is asking for specialized sprints is the first step in removing these events from your Scrum practice. Here are some common underlying issues that can cause teams to request specific, specialized sprints:

- Sprint zero is a sign of a team that's looking to do too much analysis prior to getting started, and that is not open to learning during the first sprint. Teams also use sprint zeros to build a pause into the software development process to catch up on testing, technical debt, or to simply get an easier week to recharge and prepare for the next sprint.
- Design sprints hint at a team that's uncomfortable working with incomplete information. A hard truth in the software development world is that you can't perfectly plan complexity. You will experiment, learn, and adapt throughout each sprint.
- Test, integration, and hardening sprints are attempts to cram quality into the product toward the end of the development cycle. Quality must be a primary consideration on a Scrum team. The development team tests, integrates, and improves the product every day of the sprint—not just at the end. Scrum requires that, each sprint, the team outputs a potentially releasable increment, and that requires testing.
- Release sprints give the Scrum team time to move the latest product increment to production and verify that everything still works as expected. Often the team needs this phased approach due to manual deployment and regression testing practices. Any time the team spends manually deploying a product is time they're not spending building something of value that the business wants. To help your team move past these antiquated practices and improve efficiency, map out your release process as a team, and look for ways to automate repetitive and complicated tasks.
- Bug sprints are a sign of low quality. Slow down, take fewer product backlog items into your sprint, strengthen your definition of done, and stop creating bugs. Sometimes you have to go slow to eventually deliver sooner.

I Believed in Hardening Sprints

by: Todd Miller

Many years ago, as a Scrum master, I actually recommended the use of hardening sprints to a development team that was building a sales engineering tool. They were struggling to get testing completed during sprints. After every three sprints, we would do a hardening sprint prior to a release. I went as far as recommending a hardening backlog where the team would place items that they wanted to refactor and retest.

> After several sprints, the team noticed a detrimental pattern emerging: Our hardening backlog was becoming larger and larger, and we could never complete it during the hardening sprints. We also noticed a sharp increase in the number of bugs being reported in production. During a retrospective, the team decided to ditch hardening sprints because they felt like it was causing a decrease in quality. Instead, they fine-tuned their definition of "done" so that unfinished work no longer accumulated. Harsh lesson learned!

Get to the root of the impediments that specialized sprints are covering up, and use the Scrum values to see your way to using sprints as they were intended: to time-box the end-to-end delivery of value to your customers.

Let's Change the Sprint Length

A sprint's duration is limited to one calendar month or less. Keeping the duration of a sprint consistent during a development effort establishes a cadence. A consistent time box also helps the team provide consistent forecasts to the product owner and stakeholders. And it offers the Scrum team a way to easily interpret a consistent set of data around how the team works and the complexities they're facing.

Once a sprint has started, its length can't change. We don't make them shorter because we completed all of our work done sooner, and we don't make them longer because we're not quite done. Once we decide a sprint length or set a cadence, it's a rhythm, it's a heartbeat, it's how we space out our feedback loops. Consistency is key.

The issue with extending a sprint is that continually extending it just one more day increases the risk of moving in the wrong direction, because you're not getting feedback as often as needed. You also avoid inspecting why the team struggled to deliver value at the end of the sprint. These impediments to delivery won't go away on their own and, if left unresolved, will compound in cost, delays, and risk.

And what if the extra day you request isn't enough? How many times have you seen "just one more day" turn into "just one more month"?

The sprint is a creative constraint. Every issue within our organization that prevents, delays, or inhibits delivery will come forward quickly because of the sprint time box. The urgency to deliver, coupled with the necessity to improve, drives Scrum teams to become creative at solving problems.

Why does this matter? A sprint is a fixed period of time that you've decided to use to capitalize on an opportunity in the marketplace. You're delivering the right feature at the right time for the right customer. You're opening up the opportunity for the product owner to get validation for the assumptions

Joe asks:

But we're so close! Can't we add a few days to the sprint to finish up one last product backlog item?

Nope. Not even if you've hit the end of your two-week sprint and aren't quite done with the product increment. That's a difficult spot, but you have to go to the sprint review and discuss what happened during the sprint and why you don't have a done increment of product to inspect. We do not exceed the time box—no matter what.

There's no such thing as a failed sprint, just an undesired outcome that we can learn from with stakeholders.

they've made about value. The increments created each sprint are opportunities for the product owner (and the organization as a whole) to realize a return on their investment in the product.

You're going to find problems within your processes. Development practices will need refinement. Meanwhile, you're still on the hook to deliver features by the end of the sprint. This positive sense of urgency leads to improvements that allow you to meet the definition of done and deliver features by the end of the sprint.

If you extend a sprint, you rob the team of the opportunity to improve their process, practices, and collaboration. Stick to the time box and use it to drive continuous improvement and delivery.

Ask yourself and the team honestly, "Are we extending the sprint to meet an artificial goal or to meet a metric?" Address the underlying issues of why you're not accomplishing your sprint goals. Removing the impediments to delivery will pay off far better than extending a sprint to game a metric, or make it look like you delivered a done increment within the sprint.

Joe asks:

What is the "right" sprint length?

When the team is deciding on a sprint length, your first consideration should be how long it's safe for the product owner to go without feedback from their stakeholders and customers. In highly complex environments, an overly long feedback loop could be risky.

Your team needs to discuss the product and technical implications of various sprint length options, and determine how long they're comfortable going without feedback, and how long the stakeholders are comfortable going without inspecting the latest product increment. Of course, the sprint duration also has to be technically feasible so that "done" work can be completed in its entirety.

The number of weeks that your team decides on isn't nearly as important as the reasoning behind the choice and the collaborative discussions that led to that decision. Before your team decides on "just right", make sure you've balanced risk, delivering value frequently to your customers, and technical feasibility.